第 13 章 提升法

集成学习的思想“三个臭皮匠，顶个诸葛亮”，此结论成立有前提条件。

如果这些臭皮匠的优缺点完全相同，则无论有多少臭皮匠，也顶不过诸葛亮。

上章介绍的随机森林，其组合策略为尽量使决策树之间不相关(decorrelate)。

能否更主动地寻找不同(互补)的“臭皮匠”？比如，考虑一个臭皮匠的序列：皮匠 1，皮匠 2，皮匠 3，……能否使得每个皮匠正好弥补之前所有皮匠的缺点？

这正是“提升法”(Boosting)的组合策略。

对于随机森林，每棵决策树的作用完全对称，可随便更换决策树的位置。

对于提升法，则每棵决策树的作用并不相同，这些依次而种(grown sequentially)的决策树之间的相对位置不能随意变动。

提升法是一种“序贯集成法”(sequential ensemble approach)。

随机森林使用自助样本(故可计算袋外误差)，而提升法则基于原始样本(故一般无法计算袋外误差)，但可使用不同的观测值权重(observation weights)。

13.1 自适应提升法

最早的提升法为 Freund and Schapire (1996, 1997)提出的**自适应提升法**(Adaptive Boosting，简记 AdaBoost)，仅适用于分类问题。

对于分类问题，考虑依次种M棵树$G_1(\mathbf{x}),\cdots,G_M(\mathbf{x})$。

对于第m棵树中错误分类的观测值，则在之后的第$(m+1)$棵树加大其权重，以此类推。

可通过以下两种方法来加大错分观测值的权重：

(1) 权重更新(reweighting)。在定义基尼指数或信息熵的不纯度函数，以及在计算终节点预测值时，均考虑不同观测值的权重。

(2) 再抽样(resampling)。在种每棵决策树时，都使用从“加权的分布”(weighted distribution)中再抽样得到的数据。需要先将每轮的权重标准化，使得权重之和为 1。

假设训练数据为$\left\{\mathbf{x}_i, y_i\right\}_{i=1}^{n}$。首先考虑二分类问题的 AdaBoost 算法，记响应变量$y \in \left\{-1, 1\right\}$。

“$y=1$”表示正例(positive case)，而“$y=-1$”表示反例(negative case)。

二分类问题的 AdaBoost 算法可分为以下 3 步；而其中第 2 步又分为 5 小步，针对所种决策树$m=1,\cdots,M$进行 for 循环：

1、初始化每个观测值的权重$w_i=1/n,\ i=1,\cdots,n$。这意味着，刚开始所有观测值的权重均相同，等于$1/n$(n为样本容量)。

2、对于决策树$m=1,\cdots,M$，进行以下操作：

(1) 使用观测值i的当前权重w_i，估计第m棵决策树$G_m(\mathbf{x})$；

(2) 根据当前权重w_i，计算第m棵决策树的错分率err_m：

$$err_m \equiv \frac{\sum_{i=1}^{n} w_i I(y_i \neq G_m(\mathbf{x}_i))}{\sum_{i=1}^{n} w_i} \qquad (13.1)$$

其中，分母为所有观测值的权重之和，而分子为错分观测值$(y_i \neq G_m(\mathbf{x}_i))$的权重之和。

(3) 计算正确分类的对数几率(log odds)α_m，即正确分类的概率$(1-err_m)$除以错误分类的概率err_m，再取对数：

$$\alpha_m \equiv \ln\left(\frac{1-err_m}{err_m}\right) \qquad (13.2)$$

(4) 更新观测值的权重：

$$w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot I(y_i \neq G_m(\mathbf{x}_i))\right] \qquad (13.3)$$

(5) 将所有权重标准化，保证权重之和为 1，即

$$w_i \leftarrow \frac{w_i}{\sum_{j=1}^{n} w_j} \qquad (13.4)$$

3、将每棵决策树的预测结果$G_m(\mathbf{x})$，以对数几率α_m为权重，通过加权多数票(weighted majority vote)的方式组合在一起，输出最终预测结果：

$$G(\mathbf{x}) = sign\left[\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right] \quad (13.5)$$

其中，$sign(\cdot)$为“符号函数”(sign function)，即

$$sign(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & if\ z < 0 \end{cases} \quad (13.6)$$

在 AdaBoost 算法的第 2 步中，进一步考察第(4)步观测值权重的变化。假定“弱分类器”(weak classifier)的错分率至少比随机猜测更低，则几率$\left(\frac{1-err_m}{err_m}\right)>1$，故对数几率$\alpha_m=\ln\left(\frac{1-err_m}{err_m}\right)>0$。权重更新公式(13.3)可写为

$$w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot I(y_i \neq G_m(\mathbf{x}_i))\right]=\begin{cases} w_i \cdot \left(\frac{1-err_m}{err_m}\right) & if\ y_i \neq G_m(\mathbf{x}_i) \\ w_i & if\ y_i = G_m(\mathbf{x}_i) \end{cases} \tag{13.7}$$

分类错误的观测值权重增加$\left(\frac{1-err_m}{err_m}\right)>1$倍，而分类正确的观测值权重不变。

为保持所有观测值的权重之和为 1(参见方程(13.4))，分类正确的观测值权重其实相对地缩小。

如果某观测值一直分类错误，则其权重将不断增加，表明算法越来越希望能将此“顽固”或困难(hard)的观测值正确地分类。

AdaBoost 算法对于观测值权重的更新过程，参见图 13.1。

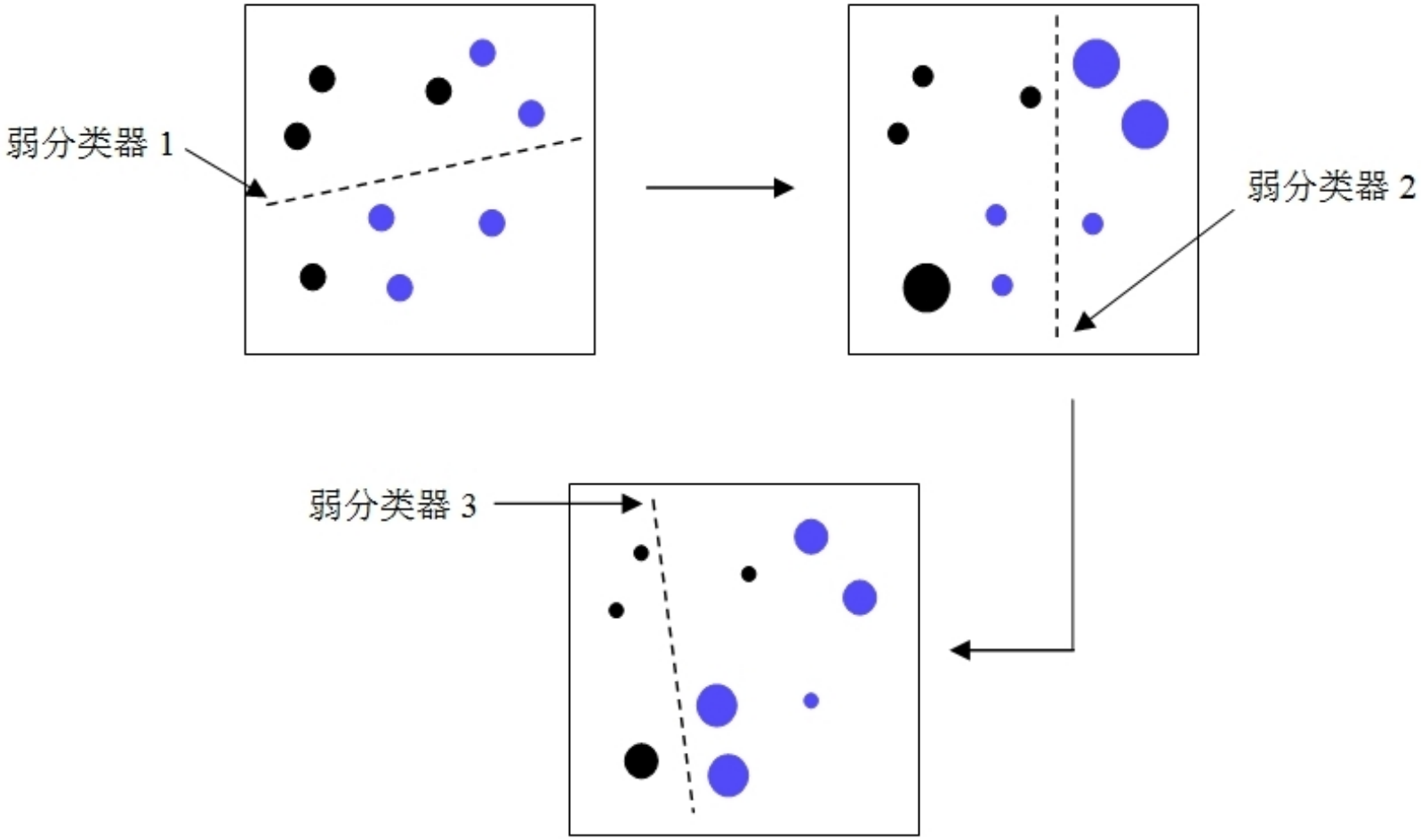

图 13.1 AdaBoost 算法对权重的更新

根据方程(13.5)，集成学习的最后结果以“加权多数票”决定，而权重为正确分类的对数几率$\alpha_m = \ln\left(\frac{1-err_m}{err_m}\right) > 0$。

分类越正确的决策树，在委员会中的投票权重越大。

将二分类问题的 AdaBoost 算法，推广到多分类问题。对于多分类问题，假设$y \in \{1, 2, \cdots, K\}$。

多分类问题的 AdaBoost 算法与二分类问题基本相同。

但最后进行加权多数投票的公式为

$$G(\mathbf{x}) = \underset{y \in \{1, 2, \cdots, K\}}{\operatorname{argmax}} \left[\sum_{m=1}^{M} \alpha_m I(y = G_m(\mathbf{x})) \right] \quad (13.8)$$

其中，给定特征向量$\mathbf{x}$，示性函数$I(y = G_m(\mathbf{x}))$用于判断第m棵树的预测结果$G_m(\mathbf{x})$是否正确。然后再以正确分类的对数几率α_m作为权重，进行加权投票。

直观上，上式分别计算$y = 1, 2, \cdots, K$所得的不同票数，然后以得票最多者胜出。

AdaBoost 算法的作用机制如何？依然可从偏差(bias)与方差(variance)的角度来考察。

一方面，由于每棵树均纠正上一棵树的错误，迫使分类器更加重视特征空间(feature space)中错误分类的区域，故可降低偏差。

另一方面，由于 AdaBoost 的最终预测结果为很多决策树的加权平均，故可达到降低方差的效果。

早期的 AdaBoost 算法经常使用“树桩”(stump)作为弱学习器，即只有一个根节点与两个终节点的决策树。

该决策树仅选择一个变量，做一次分裂，故一般为弱学习器。

例如，仅根据身高的信息判断性别，参见图 13.2。

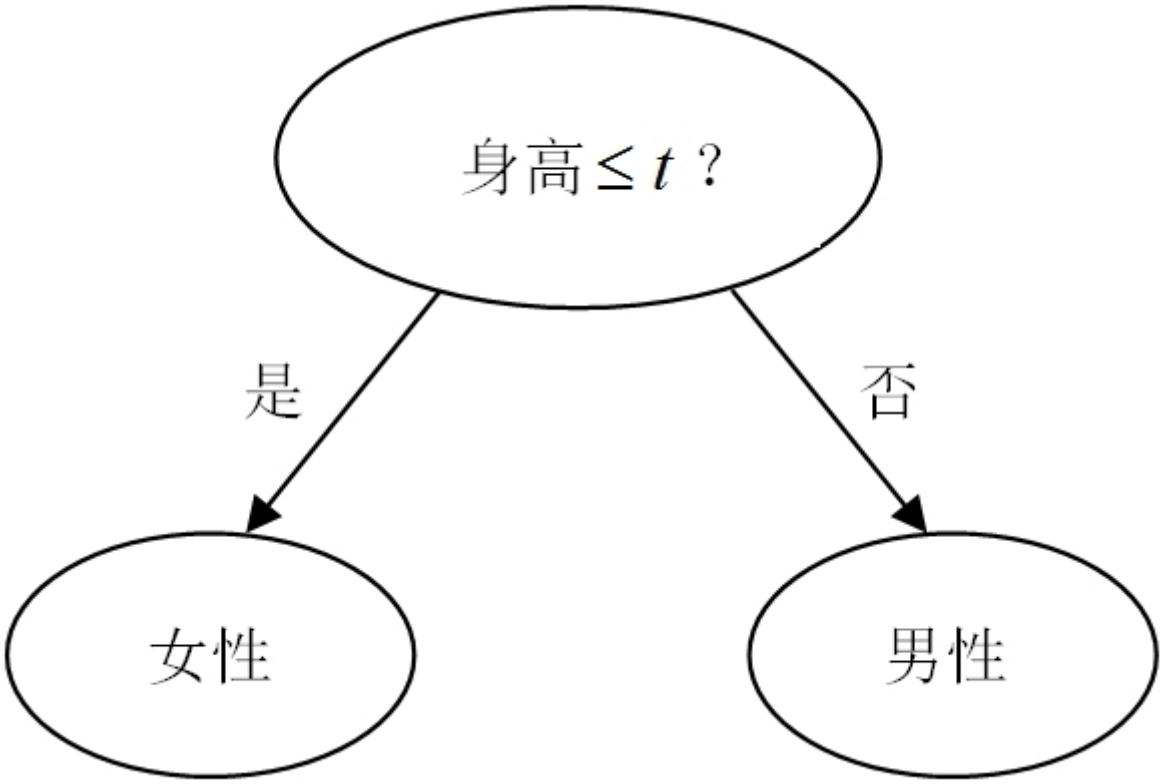

图 13.2 根据身高判断性别的树桩(弱学习器)

由于树桩仅做一次分裂，故其偏差较大；而方差较小，故很难过拟合。

使用树桩作为基学习器是基于这样一种认识，即“慢学习”(learning slowly) 通常效果较好。

但若使用树桩作为基学习器，则无法捕捉变量之间的“交互效应”(interaction effect，类似于线性回归的交互项)，故未必总能达到最好的预测效果。

13.2 AdaBoost 的统计解释

自从 Freund and Schapire (1996)提出 AdaBoost 算法后，即取得很好的预测效果。Friedman et al. (2000)给出 AdaBoost 的统计解释。

AdaBoost 算法可等价地视为**前向分段加法模型**(forward stagewise additive modeling)，并使用**指数损失函数**(exponential loss function)。

对于二分类问题的 AdaBoost 算法，可将最终表达式 $G(\mathbf{x})=sign\left[\sum_{m=1}^{M}\alpha_m G_m(\mathbf{x})\right]$ 中的每一项 $G_m(\mathbf{x})\in\{-1,1\}$ 视为“基函数”(basis function)，类似于泰勒展开式(Taylor expansion)的每一项。

更一般地，考虑将希望学到的函数 $f(\mathbf{x})$ 做“基函数展开”(basis function expansion)，由此得到**加法模型**(additive model)”：

$$f(\mathbf{x})=\sum_{m=1}^{M}\beta_m G(\mathbf{x};\boldsymbol{\gamma}_m) \qquad (13.9)$$

其中，β_m 为“展开系数”(expansion coefficients)。$G(\mathbf{x};\boldsymbol{\gamma}_m)$ 为基函数，而 $\boldsymbol{\gamma}_m$ 为参数向量。

如以决策树为基函数，则 $\boldsymbol{\gamma}_m$ 表示分裂变量、在何处分裂以及终节点的预测值。

为估计展开系数β_m与基函数参数$\boldsymbol{\gamma}_m$，可最小化以下目标函数：

$$\min_{\{\beta_m,\boldsymbol{\gamma}_m\}} \sum_{i=1}^{n} L\left(y_i, \sum_{m=1}^{M} \beta_m G(\mathbf{x}_i;\boldsymbol{\gamma}_m)\right) \quad (13.10)$$

其中，$L\left(y_i, f(\mathbf{x}_i)\right)$为损失函数，比如误差平方$(y_i - f(\mathbf{x}_i))^2$，或 0-1 损失函数$I\left(y_i \neq f(\mathbf{x}_i)\right)$等。

直接求解此最小化问题很困难。例如，若基函数$G(\mathbf{x}_i;\boldsymbol{\gamma}_m)$为决策树，则$G(\mathbf{x}_i;\boldsymbol{\gamma}_m)$无解析表达式。

考虑分步计算，即所谓**前向分段算法**(forward stagewise algorithm)：

1、初始化 $f(\mathbf{x})$为零(函数)，即 $f_0(\mathbf{x})=0$。

2、对于基函数$m=1,\cdots,M$，进行以下 `for` 循环：

(1) 求解最小化问题

$$(\beta_m,\boldsymbol{\gamma}_m)=\underset{\{\beta,\boldsymbol{\gamma}\}}{\operatorname{argmin}}\sum_{i=1}^{n}L\left(y_i,\ f_{m-1}(\mathbf{x}_i)+\beta\cdot G(\mathbf{x}_i;\boldsymbol{\gamma})\right) \quad (13.11)$$

其中，在已知 $f_{m-1}(\mathbf{x}_i)$情况下，选择下一轮的最优展开系数β_m与基函数参数$\boldsymbol{\gamma}_m$。

(2) 更新加法模型(13.9)：

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \beta_m \cdot G(\mathbf{x};\boldsymbol{\gamma}_m) \quad (13.12)$$

须注意“分段”与“分步”的区别。对于“分段”(stagewise)算法，已经加入的项(基函数)，在之后的优化中不再调整。

另一方面，对于“分步”(stepwise)算法，则已经加入的项依然可在后续优化中调整。

例如，在统计学的“前向分步回归”(forward stepwise regression)，每次加入最显著的变量，并重新调整之前已在模型中的变量之回归系数。

命题 13.1 (Friedman et al., 2000)　二分类问题的 AdaBoost 算法，等价于使用指数损失函数的前向分段加法模型。其中，**指数损失函数**(exponential loss function)为

$$L\big(y, f(\mathbf{x})\big) = \exp\big(-y f(\mathbf{x})\big) \qquad (13.13)$$

证明 对于 AdaBoost 算法，基函数为$G_m(\mathbf{x}_i) \in \{-1, 1\}$。

考虑给定 $f_{m-1}(\mathbf{x})$，如何最优化 $f_m(\mathbf{x})$。

将指数损失函数(13.13)代入目标函数(13.10)，并使用函数更新规则(13.12)可得：

$$\min_{\{\beta, G\}} \sum_{i=1}^{n} L\left(y_i,\, f_m(\mathbf{x}_i)\right)$$

$$= \sum_{i=1}^{n} \exp\left[-y_i\, f_m(\mathbf{x}_i)\right] \tag{13.14}$$

$$= \sum_{i=1}^{n} \exp\left[-y_i\left(\, f_{m-1}(\mathbf{x}_i) + \beta \cdot G(\mathbf{x}_i)\right)\right]$$

进一步，目标函数可等价地写为

$$\sum_{i=1}^{n}\exp\Big[-y_i\big(f_{m-1}(\mathbf{x}_i)+\beta\cdot G(\mathbf{x}_i)\big)\Big] \qquad (y_i\text{乘进去})$$

$$=\sum_{i=1}^{n}\exp\big[-y_i f_{m-1}(\mathbf{x}_i)-\beta\cdot y_i G(\mathbf{x}_i)\big] \qquad (\text{分成两项})$$

$$=\sum_{i=1}^{n}\underbrace{\exp\big[-y_i f_{m-1}(\mathbf{x}_i)\big]}_{\equiv w_i^{(m)}}\cdot\exp\big[-\beta\cdot y_i G(\mathbf{x}_i)\big] \qquad (\text{简化})$$

$$\equiv\sum_{i=1}^{n}w_i^{(m)}\cdot\exp\big[-\beta\cdot y_i G(\mathbf{x}_i)\big] \tag{13.15}$$

其中，权重$w_i^{(m)}\equiv\exp\big(-y_i f_{m-1}(\mathbf{x}_i)\big)$，仅与$i$与$m$有关，不依赖于$\beta$或$G$。

实际观测值$y_i \in \{-1,1\}$，而预测值$G(\mathbf{x}_i) \in \{-1,1\}$。

如果预测正确，即$y_i = G(\mathbf{x}_i)$，则有$y_i G(\mathbf{x}_i) = 1$。

反之，如果预测错误，即$y_i \neq G(\mathbf{x}_i)$，则有$y_i G(\mathbf{x}_i) = -1$。

这正是约定$y_i \in \{-1,1\}$的方便之处。

可把样本分为正确分类($y_i = G(\mathbf{x}_i)$)与错误分类($y_i \neq G(\mathbf{x}_i)$)的两部分，并将目标函数写为

$$\sum_{i=1}^{n} w_i^{(m)} \exp\left[-\beta y_i G(\mathbf{x}_i)\right] \qquad \text{(将样本分为两类)}$$

$$= \sum_{y_i = G(\mathbf{x}_i)} w_i^{(m)} \exp\left[-\beta y_i G(\mathbf{x}_i)\right] + \sum_{y_i \neq G(\mathbf{x}_i)} w_i^{(m)} \exp\left[-\beta y_i G(\mathbf{x}_i)\right] \quad \text{(代入} y_i G(\mathbf{x}_i) = \pm 1\text{)}$$

$$= \sum_{y_i = G(\mathbf{x}_i)} w_i^{(m)} \exp\left[-\beta \cdot 1\right] + \sum_{y_i \neq G(\mathbf{x}_i)} w_i^{(m)} \exp\left[-\beta \cdot (-1)\right] \qquad \text{(简化)}$$

$$= e^{-\beta} \sum_{y_i = G(\mathbf{x}_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G(\mathbf{x}_i)} w_i^{(m)}$$

(13.16)

接上式，继续将目标函数变为更适合最优化的形式：

$$e^{-\beta}\sum_{y_i=G(\mathbf{x}_i)} w_i^{(m)} + e^{\beta}\sum_{y_i\neq G(\mathbf{x}_i)} w_i^{(m)} \qquad (\text{加减 } e^{-\beta}\cdot\sum_{y_i\neq G(\mathbf{x}_i)} w_i^{(m)})$$

$$=\left(e^{-\beta}\sum_{y_i=G(\mathbf{x}_i)} w_i^{(m)} + e^{-\beta}\cdot\sum_{y_i\neq G(\mathbf{x}_i)} w_i^{(m)}\right)+\left(e^{\beta}\sum_{y_i\neq G(\mathbf{x}_i)} w_i^{(m)} - e^{-\beta}\cdot\sum_{y_i\neq G(\mathbf{x}_i)} w_i^{(m)}\right)$$

$$=e^{-\beta}\sum_{i=1}^{n} w_i^{(m)} + (e^{\beta}-e^{-\beta})\sum_{y_i\neq G(\mathbf{x}_i)} w_i^{(m)} \qquad (\text{合并同类项})$$

$$=e^{-\beta}\sum_{i=1}^{n} w_i^{(m)} + (e^{\beta}-e^{-\beta})\sum_{i=1}^{n} w_i^{(m)} I(y_i\neq G(\mathbf{x}_i)) \quad (\text{写回全样本的形式})$$

(13.17)

其中，$\sum_{i=1}^{n} w_i^{(m)} I(y_i\neq G(\mathbf{x}_i)) = \sum_{y_i\neq G(\mathbf{x}_i)} w_i^{(m)}$。

针对此目标函数，考虑在给定β的情况下，G_m的最优解：

$$\min_{G} \underbrace{e^{-\beta}\sum_{i=1}^{n} w_i^{(m)}}_{\text{与}G\text{无关}} + \underbrace{(e^{\beta}-e^{-\beta})}_{>0}\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G(\mathbf{x}_i)) \quad (13.18)$$

其中，第一项$e^{-\beta}\sum_{i=1}^{n} w_i^{(m)}$与$G$无关，而第二项的系数$(e^{\beta}-e^{-\beta})>0$。

因此，该最小化问题等价于：

$$\min_{G} \sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G(\mathbf{x}_i)) \quad (13.19)$$

因此，G_m的最优解可写为

$$G_m \equiv \underset{G}{\operatorname{argmin}} \sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G(\mathbf{x}_i)) \qquad (13.20)$$

这意味着，应最小化加权的训练误差。如果以决策树作为基学习器，则此解正是加权的决策树。

最优解G_m并不依赖于β。后面将证明，上式中权重$w_i^{(m)}$的更新规则与 AdaBoost 的权重更新规则(13.3)等价。

将G_m的最优解(13.20)代入目标函数(13.16)，可得一元优化问题：

$$\min_{\beta} \; e^{-\beta} \sum_{y_i=G_m(\mathbf{x}_i)} w_i^{(m)} + e^{\beta} \sum_{y_i \neq G_m(\mathbf{x}_i)} w_i^{(m)} \qquad (13.21)$$

在上式中，对β求导，可得一阶条件：

$$e^{-\beta} \sum_{y_i=G_m(\mathbf{x}_i)} w_i^{(m)} = e^{\beta} \sum_{y_i \neq G_m(\mathbf{x}_i)} w_i^{(m)} \qquad (13.22)$$

上式两边同除权重之和$\sum_{i=1}^{n} w_i^{(m)}$，可得：

$$e^{-\beta}\underbrace{\frac{\sum_{y_i=G_m(\mathbf{x}_i)} w_i^{(m)}}{\sum_{i=1}^{n} w_i^{(m)}}}_{=1-err_m} = e^{\beta}\underbrace{\frac{\sum_{y_i\neq G_m(\mathbf{x}_i)} w_i^{(m)}}{\sum_{i=1}^{n} w_i^{(m)}}}_{=err_m} \qquad (13.23)$$

由此可得：

$$e^{-\beta}(1-err_m) = e^{\beta} err_m \qquad (13.24)$$

移项整理可得：

$$e^{2\beta} = \frac{1-err_m}{err_m} \qquad (13.25)$$

在上式两边取对数，可得β_m的最优解：

$$\beta_m = \frac{1}{2}\ln\left(\frac{1-err_m}{err_m}\right) = \frac{1}{2}\alpha_m \qquad (13.26)$$

前向分段加法模型的最优展开系数β_m为对数几率α_m的一半。

下面证明，前向分段算法的权重$w_i^{(m)} = \exp\left(-y_i f_{m-1}(\mathbf{x}_i)\right)$之更新规则与 AdaBoost 相同。根据$w_i^{m+1}$的定义：

$$
\begin{aligned}
w_i^{m+1} &\equiv \exp\left[-y_i f_m(\mathbf{x}_i)\right] \qquad \text{(代入函数更新规则)} \\
&= \exp\left[-y_i\left(f_{m-1}(\mathbf{x}_i)+\beta_m \cdot G_m(\mathbf{x}_i)\right)\right] \quad \text{(展开)} \\
&= \underbrace{\exp\left[-y_i f_{m-1}(\mathbf{x}_i)\right]}_{\equiv w_i^m} \cdot \exp\left[-y_i \beta_m \cdot G_m(\mathbf{x}_i)\right] \quad (w_i^m\text{的定义}) \\
&= w_i^m \cdot \exp\left[-\beta_m \cdot y_i G_m(\mathbf{x}_i)\right] \quad \text{(参见下文)} \\
&= w_i^m \cdot \exp\left[\beta_m\left(2I(y_i \neq G(\mathbf{x}_i))-1\right)\right] \qquad \text{(展开)} \\
&= w_i^m \cdot \exp\left[2\beta_m I(y_i \neq G(\mathbf{x}_i))-\beta_m\right] \qquad (\text{代入} 2\beta_m=\alpha_m) \quad (13.27) \\
&= w_i^m \cdot \exp\left[\alpha_m I(y_i \neq G(\mathbf{x}_i))\right] \cdot \underbrace{e^{-\beta_m}}_{\text{与 } i \text{ 无关}}
\end{aligned}
$$

其中，$-y_i G_m(\mathbf{x}_i) = \begin{cases} -1 & if\ y_i = G_m(\mathbf{x}_i) \\ 1 & if\ y_i \neq G_m(\mathbf{x}_i) \end{cases}$，而

$2I(y_i \neq G(\mathbf{x}_i)) - 1 = \begin{cases} -1 & if\ y_i = G(\mathbf{x}_i) \\ 1 & if\ y_i \neq G(\mathbf{x}_i) \end{cases}$，故二者相等。

由于$e^{-\beta_m} > 0$与观测值i无关，故上式与 AdaBoost 的权重更新规则(13.3)在本质上等价。 根据方程(13.9)，加法模型的最终估计结果为

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m \cdot G_m(\mathbf{x}) \qquad (13.28)$$

因此，前向分段加法模型的决策规则也与 AdaBoost 等价：

$$
\begin{aligned}
sign\left[f(\mathbf{x})\right] &= sign\left[\sum_{m=1}^{M}\beta_m \cdot G_m(\mathbf{x})\right] \qquad (\text{代入}\beta_m = \alpha_m / 2) \\
&= sign\left[\sum_{m=1}^{M}\frac{1}{2}\alpha_m \cdot G_m(\mathbf{x})\right] \\
&= sign\left[\sum_{m=1}^{M}\alpha_m \cdot G_m(\mathbf{x})\right]
\end{aligned}
\tag{13.29}
$$

综上所述，对于二分类问题，带指数损失函数的前向分段加法模型，等价于 AdaBoost 算法。 ■

13.3 回归问题的提升法

发现 AdaBoost 算法的数学本质后，即开启了改进与推广 AdaBoost 之门。

AdaBoost 算法使用指数损失函数。在前向分段加法模型中，使用其他损失函数，即可得到不同的算法。

最初的 AdaBoost 算法仅适用于分类问题，因为它使用指数损失函数。

对于回归问题，自然地可考虑使用“误差平方”(squared loss)的损失函数：

$$L\big(y, f(\mathbf{x})\big) = \big[y - f(\mathbf{x})\big]^2 \qquad (13.30)$$

将前向分段加法模型代入此损失函数可得：

$$\begin{aligned}\min_{\beta, G}\ & L\left(y_i,\ f_{m-1}(\mathbf{x}_i)+\beta\cdot G(\mathbf{x}_i;\boldsymbol{\gamma})\right)\\ &=\left[y_i-f_{m-1}(\mathbf{x}_i)-\beta\cdot G(\mathbf{x}_i;\boldsymbol{\gamma})\right]^2 \qquad (13.31)\\ &=\left[r_i^{(m)}-\beta\cdot G(\mathbf{x}_i;\boldsymbol{\gamma})\right]^2\end{aligned}$$

其中，$r_i^{(m)}\equiv y_i-f_{m-1}(\mathbf{x}_i)$为当前阶段模型的回归残差(residual)。这意味着，只要以当前残差$r_i^{(m)}$为响应变量，对特征向量$\mathbf{x}_i$进行回归即可。这种方法称为**回归问题的提升法**，或“2-范数提升法”(L_2boosting)。

在以残差$r_i^{(m)}$为响应变量对特征向量$\mathbf{x}_i$进行“回归”时，既可进行线性回归，也可使用回归树。

如果使用回归树，则称为**回归提升树**(boosted regression tree)。在估计回归提升树时，有较多的调节参数需要考虑，有时也称为“超参数”(hyperparameters)：

(1) 决策树的数量 M。虽然提升法不容易过拟合，但也可能因M太大而过拟合。这是提升法与随机森林的区别之一。

一般可使用交叉验证选择M。

(2) 决策树的分裂次数 *d*，称为**交互深度**(interaction depth)。

例如，$d=1$，则为仅分裂 1 次的树桩。此时无法考虑变量之间的交互效应(interaction effects)，相当于没有交互项(interaction term)的线性回归。

如果$d=2$，决策树分裂两次，则可包含两个变量之间的交互效应；

如果$d=3$，决策树分裂三次，则可包含三个变量之间的交互效应；以此类推。

一般建议选择$d=4\sim8$，以充分捕捉(多个)变量之间的交互效应。

(3) 学习率(learning rate) $0<\eta<1$。通常选择$\eta=0.1$或0.01，以进一步降低学习速度，并可使用更多决策树(M 更大)来拟合函数。

“学习率”也称为**收缩参数**(shrinkage parameter)。

设定较小的学习率，也是一种正则化方法(regularization)，可避免过拟合。

上述调节参数之间，也会相互影响。

例如，如果所设的学习率η很小，则通常需要更大的M 。

针对训练数据$\{\mathbf{x}_i, y_i\}_{i=1}^n$，想学得函数$f(\mathbf{x})$，则回归提升树的具体算法如下：

1、初始化$f(\mathbf{x})=0$，则残差就是响应变量$r_i = y_i \ (i=1,\cdots,n)$。

2、对于决策树$m=1,\cdots,M$，进行以下 `for` 循环：

(1) 使用数据$\{\mathbf{x}_i, r_i\}_{i=1}^n$，估计拥有$d$个分裂($d+1$个终节点)的决策树$G_m(\mathbf{x})$；

(2) 函数更新：

$$f(\mathbf{x}) \leftarrow f(\mathbf{x}) + \eta G_m(\mathbf{x}) \qquad (13.32)$$

其中，$0<\eta<1$为学习率。

(3) 残差更新：

$$r_i \leftarrow r_i - \eta G_m(\mathbf{x}) \qquad (13.33)$$

在上式中，由于函数$f(\mathbf{x})$增加$\eta G_m(\mathbf{x})$，故残差r_i减少$\eta G_m(\mathbf{x})$。

3、输出结果：

$$f(\mathbf{x}) = \sum_{m=1}^{M} \eta G_m(\mathbf{x}) \qquad (13.34)$$

在上式中，将每次学习的增量$\eta G_m(\mathbf{x})$进行加总，即为最终学得的函数$f(\mathbf{x})$。

下面使用模拟数据，考察在提升法中加入更多决策树的作用。

首先，导入所需模块：

```
In [1]: import numpy as np
   ...: import pandas as pd
   ...: import matplotlib.pyplot as plt
   ...: import seaborn as sns
   ...: from sklearn.ensemble import
            GradientBoostingRegressor
```

其次，假设特征变量x服从$[0,1)$区间的均匀分布，随机抽取 500 个观测值；而响应变量的数据生成过程为$y=\sin(2\pi x)+\varepsilon$，其中$\varepsilon \sim N(0,0.1^2)$：

```
In [2]: np.random.seed(1)
   ...: x = np.random.uniform(0, 1, 500)
   ...: y = np.sin(2 * np.pi * x) + np.random.normal(0,
                    0.1, 500)
   ...: data = pd.DataFrame(x, columns=['x'])
   ...: data['y'] = y
   ...: w = np.linspace(0, 1, 100)
```

其中，np.sin()与 np.pi 分别为 Numpy 的正弦函数与圆周率π。

更直观地，画散点图与$f(x)=\sin(2\pi x)$的函数曲线：

```
In [3]: sns.scatterplot(x='x', y='y', s=20, data=data,
                        alpha=0.3)
   ...: plt.plot(w, np.sin(2 * np.pi * w))
   ...: plt.title('Data Generating Process')
```

其中，第 1 个命令的参数“s=20”用于指定图标尺寸(marker size)，结果参见图 13.3。

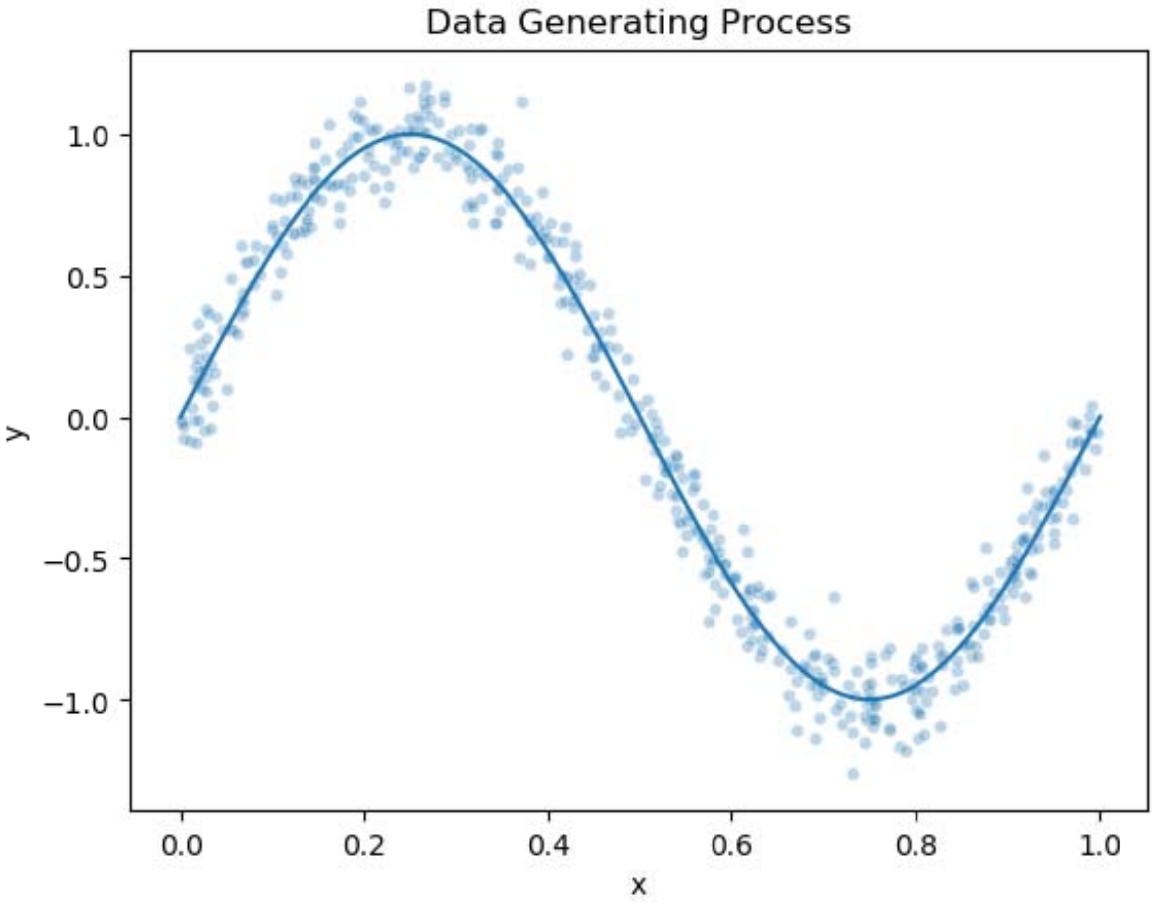

图 13.3 模拟数据的生成过程

下面，使用 sklearn 的 `GradientBoostingRegressor` 类，估计回归提升树。

特别地，使用树桩(分裂次数$d=1$)，通过 `for` 循环，让决策树数目M分别等于 1, 10, 100, 1000，画图展示函数拟合的效果：

```
In [4]: for i, m in zip([1, 2, 3, 4], [1, 10, 100, 1000]):
   ...:     model = GradientBoostingRegressor
                      (n_estimators=m, max_depth=1,
                    learning_rate=1, random_state=123)
   ...:     model.fit(x.reshape(-1, 1), y)
   ...:     pred = model.predict(w.reshape(-1, 1))
   ...:     plt.subplot(2, 2, i)
```

```
...:     plt.plot(w, np.sin(2 * np.pi * w), 'k',
                  linewidth=1)
...:     plt.plot(w, pred, 'b')
...:     plt.text(0.65, 0.8, f'M = {m}')
...: plt.subplots_adjust(wspace=0.4, hspace=0.4)
```

其中，GradientBoostingRegressor()的参数“max_depth=1”表示仅分裂 1 次(即树桩)；

参数“learning_rate=1”将学习率设为 1(默认学习率为 0.1)，结果参见图 13.4。

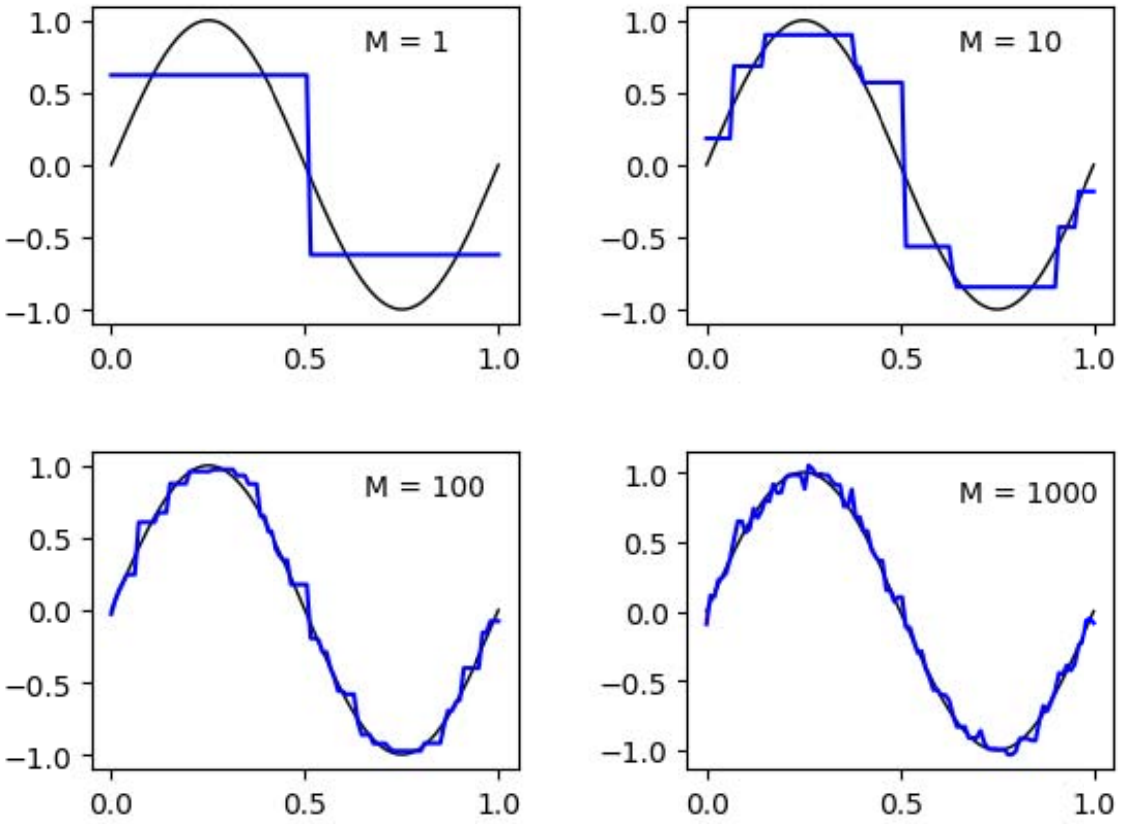

图 13.4 回归提升树的拟合效果

13.4 回归问题的其他损失函数

对于回归问题的提升法，也可以使用其他损失函数。

比如，更为稳健的“绝对损失函数”(absolute loss)，也称为**拉普拉斯损失函数**(Laplace loss)：

$$L\big(y, f(\mathbf{x})\big) = \big|y - f(\mathbf{x})\big| \qquad (13.35)$$

其中，由于损失函数为误差的绝对值(L_1loss)，故不易受到“极端值”(outliers)的影响。如果使用平方损失函数(L_2loss)，则较易受极端值影响。

对于回归问题，另一可供选择的损失函数为统计学中的**胡贝尔损失函数**(Huber loss)：

$$L\big(y, f(\mathbf{x})\big)=\begin{cases}\dfrac{1}{2}\big(y-f(\mathbf{x})\big)^2 & if\ \ \big|y-f(\mathbf{x})\big|\le\delta \\ \delta\Big(\big|y-f(\mathbf{x})\big|-\delta/2\Big) & if\ \ \big|y-f(\mathbf{x})\big|>\delta\end{cases} \tag{13.36}$$

其中，$\delta>0$为调节参数，可通过交叉验证确定。

胡贝尔损失函数综合了平方损失(L_2loss)与绝对损失(L_1loss)。

当误差的绝对值小于或等于δ时，使用平方损失函数$\frac{1}{2}\left(y-f(\mathbf{x})\right)^2$。

当误差的绝对值大于δ时，则使用绝对损失函数$\delta\left(\left|y-f(\mathbf{x})\right|-\delta/2\right)$，以避免受到极端值的太大影响。

胡贝尔损失函数综合了平方损失与绝对损失函数的优点。

通过引入新的调节参数δ，使得算法更具灵活性。

13.5 梯度提升法

Friedman (2001)将 AdaBoost 推广到更一般的**梯度提升法**(Gradient Boosting Machine，简记 GBM)。

GBM 的主要创新在于，提出以非参方法估计基函数，并在“函数空间”(function space)使用“梯度下降”(gradient descent) 进行近似求解。

对于训练数据$\{\mathbf{x}_i, y_i\}_{i=1}^n$，若以函数$F(\mathbf{x})$来预测$y$，则在总体中的“期望损失函数”(expected loss function)为

$$\mathrm{E}_{y,\mathbf{x}} L(y, F(\mathbf{x})) \qquad (13.37)$$

其中，$L(y, F(\mathbf{x}))$为给定的损失函数，而期望算子$\mathrm{E}_{y,\mathbf{x}}(\cdot)$同时对$y$与$\mathbf{x}$求期望。

我们的问题是，在“函数空间”(function space)，即由所有可能的函数$F(\mathbf{x})$组成之集合，找到最优函数$F^*(\mathbf{x})$，能使期望损失函数最小化：

$$F^*(\mathbf{x}) \equiv \underset{F}{\operatorname{argmin}}\ \mathrm{E}_{y,\mathbf{x}}\, L(y, F(\mathbf{x})) \qquad (13.38)$$

其中，最优化针对函数空间的函数$F(\cdot)$进行。

根据迭代期望定律，可将目标函数简化，即先给定$\mathbf{x}$，针对y求条件期望$\mathrm{E}_y(\cdot \mid \mathbf{x})$，所得结果为$\mathbf{x}$的函数；然后再对$\mathbf{x}$求期望$\mathrm{E}_{\mathbf{x}}(\cdot)$：

$$
\begin{aligned}
F^*(\mathbf{x}) &\equiv \underset{F}{\operatorname{argmin}}\ \mathrm{E}_{y,\mathbf{x}}\, L(y, F(\mathbf{x})) \\
&= \underset{F}{\operatorname{argmin}}\ \mathrm{E}_{\mathbf{x}}\Big[\mathrm{E}_{y}(L(y, F(\mathbf{x}))\,|\,\mathbf{x})\Big]
\end{aligned}
\tag{13.39}
$$

因此，对于任意给定$\mathbf{x}$，此最小化问题可等价地写为

$$
\min_{F}\ \mathrm{E}_{y}(L(y, F(\mathbf{x}))\,|\,\mathbf{x}) \tag{13.40}
$$

使用非参数方法(nonparametric approach)，将$F(\mathbf{x})$在每个$\mathbf{x}$的取值均视为“参数”(parameter)。

函数$F(\mathbf{x})$可视为无穷维向量，故有无穷多“参数”。函数空间其实是一个无穷维的向量空间。

在理论上，可考虑在此无穷维的函数空间进行梯度下降，即所谓**函数梯度下降**(functional gradient descent)。仍然假设前向分段加法模型：

$$F^*(\mathbf{x})=\sum_{m=1}^{M} f_m(\mathbf{x}) \qquad (13.41)$$

其中，为保持与 Friedman (2001)的符号一致，用 $f_m(\mathbf{x})$表示加法模型的第m项，其作用类似于上文的$\beta_m G_m(\mathbf{x})$。

如果给定$\mathbf{x}$，则目标函数(13.40)的(无穷维)梯度向量事实上只是一维标量：

$$g_m(\mathbf{x}) \equiv \left[\frac{\partial \mathrm{E}_y(L(y,F(\mathbf{x}))\,|\,\mathbf{x})}{\partial F(\mathbf{x})} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \tag{13.42}$$

其中，偏导数的评估在$F_{m-1}(\mathbf{x}) = \sum_{k=1}^{m-1} f_k(\mathbf{x})$(上一阶段的函数估计)处进行。

使用梯度下降法，则当前阶段的函数变化为负梯度方向：

$$f_m(\mathbf{x}) = -\rho_m g_m(\mathbf{x}) \tag{13.43}$$

其中，$-\rho_m$为“步长”(step size)，也称为“学习率”(learning rate)。

在无穷维的函数空间进行梯度下降，其实是不可行的(infeasible)，因为我们并没有无穷多的数据。

根据观测数据$\{\mathbf{x}_i, y_i\}_{i=1}^{n}$，对应于方程(13.42)的样本负梯度为

$$-g_m(\mathbf{x}_i) = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \quad (i=1,\cdots,n) \qquad (13.44)$$

其中，$-g_m(\mathbf{x}_i)$为在观测值$\mathbf{x}_i$处的负梯度方向，称为**准残差**(pseudo residuals)。

同时考虑所有观测值($i=1,\cdots,n$)，则上式成为n维的负梯度向量，或准残差向量：

$$-\mathbf{g}_m(\mathbf{x}) \equiv \begin{pmatrix} -g_m(\mathbf{x}_1) \\ \vdots \\ -g_m(\mathbf{x}_n) \end{pmatrix} \qquad (13.45)$$

依然无法计算不在样本内的其他$\mathbf{x}$处的梯度，因为没有相应的y观测值。

为了解决此问题，注意到负梯度方向$-\mathbf{g}_m(\mathbf{x}_i)$还须是基函数的一员：

$$f_m(\mathbf{x}) \equiv \beta_m h(\mathbf{x};\mathbf{a}_m) \qquad (13.46)$$

其中，$h(\mathbf{x};\mathbf{a}_m)$为基函数(由基学习器所决定)，带有参数$\mathbf{a}_m$。

基函数$h(\mathbf{x};\mathbf{a}_m)$类似于上文的$G(\mathbf{x};\boldsymbol{\gamma}_m)$，只是使用了不同的符号(与文献保持一致)。

具体来说，选择与负梯度向量$-\mathbf{g}_m(\mathbf{x})=\left(-g_m(\mathbf{x}_1),\cdots,-g_m(\mathbf{x}_n)\right)'$最为接近的基函数向量$\left(h(\mathbf{x}_1;\mathbf{a}_m),\cdots,h(\mathbf{x}_n;\mathbf{a}_m)\right)'$。

这其实是最小二乘问题，即以准残差$-g_m(\mathbf{x}_i)$为响应变量，对$h(\mathbf{x}_i;\mathbf{a})$作线性投影(linear projection)：

$$\mathbf{a}_m \equiv \underset{\mathbf{a},\beta}{\operatorname{argmin}} \sum_{i=1}^{n}\left[-g_m(\mathbf{x}_i)-\beta h(\mathbf{x}_i;\mathbf{a})\right]^2 \qquad (13.47)$$

求得最优的$h(\mathbf{x}_i;\mathbf{a}_m)$之后，可使用下式进行函数更新：

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x};\mathbf{a}_m) \qquad (13.48)$$

其中，步长ρ_m可通过“直线搜索”(line search)来确定，即

$$\rho_m \equiv \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \rho \cdot h(\mathbf{x}_i;\mathbf{a}_m)\right) \qquad (13.49)$$

综上所述，梯度提升法的算法可总结如下：

1、初始化$F_0(x)=\underset{c\in\mathbb{R}}{\operatorname{argmin}}\sum_{i=1}^{n}L(y_i,c)$，则$F_0(x)$为最优的常值函数(constant function)。

2、对于基函数$m=1,\cdots,M$，进行以下 `for` 循环：

(1) 计算准残差：

$$r_i^{(m)}=-\left[\frac{\partial L(y_i,F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}\qquad(i=1,\cdots,n)\qquad(13.50)$$

(2) 将准残差$r_i^{(m)}$对$\mathbf{x}$进行回归：

$$\mathbf{a}_m = \underset{\mathbf{a},\,\beta}{\operatorname{argmin}} \sum_{i=1}^{n} \left[r_i^{(m)} - \beta h(\mathbf{x}_i;\mathbf{a}) \right]^2 \qquad (13.51)$$

(3) 计算最优步长：

$$\rho_m = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \rho \cdot h(\mathbf{x}_i;\mathbf{a}_m) \right) \qquad (13.52)$$

(4) 更新函数：

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x};\mathbf{a}_m) \qquad (13.53)$$

3、输出结果$F_M(\mathbf{x})$。

例 作为一个简单的“现实检查”(reality check)，下面将梯度提升法应用于回归问题。损失函数为误差平方函数$L\big(y,F(\mathbf{x})\big)=\frac{1}{2}\big[y-F(\mathbf{x})\big]^2$。

将平方损失函数代入准残差的表达式(13.50)可得：

$$r_i^{(m)} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = -\left[\frac{\partial \frac{1}{2}\left[y_i - F(\mathbf{x}_i)\right]^2}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$$

$$= -\frac{1}{2}\cdot 2\left[y_i - F(\mathbf{x}_i)\right]\cdot(-1)\Big|_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = \left[y_i - F(\mathbf{x}_i)\right]\Big|_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$$

$$= y_i - F_{m-1}(\mathbf{x}_i) \tag{13.54}$$

对于回归问题，准残差$r_i^{(m)}$就是真正的残差$\left[y_i - F_{m-1}(\mathbf{x}_i)\right]$。

进一步，计算最优步长：

$$\begin{aligned}\rho_m &\equiv \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{n}\left[y_i - F_{m-1}(\mathbf{x}_i) - \rho \cdot h(\mathbf{x}_i;\mathbf{a}_m)\right]^2 \\ &= \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{n}\left[r_i^{(m)} - \rho \cdot h(\mathbf{x}_i;\mathbf{a}_m)\right]^2 \equiv \beta_m\end{aligned} \tag{13.55}$$

最优步长其实就是展开系数β_m。

当基学习器$h(\mathbf{x}_i;\mathbf{a}_m)$为决策树时，梯度提升法就是上文的回归提升树(boosted regression tree)。

13.6 二分类问题的逻辑损失函数

针对二分类问题，AdaBoost 使用如下指数损失函数(exponential loss)：

$$L\left(y,\ f(\mathbf{x})\right)=\exp\left(-yf(\mathbf{x})\right) \qquad (13.56)$$

在预测时，以$sign\left[\ f(\mathbf{x})\right]$预测$y\in\left\{-1,1\right\}$，故“$yf(\mathbf{x})>0$”意味着预测正确。

反之，“$yf(\mathbf{x})<0$”则意味着预测错误。

推而广之，$yf(\mathbf{x})$越大，则预测越正确。

为此，称“$yf(\mathbf{x})$”为**裕度**(余裕程度，margin)。

裕度越大，则预测越正确；反之，则预测越错误。

可使用**负裕度**(negative margin) $-yf(\mathbf{x})$度量预测的错误程度。

指数损失函数为负裕度的指数函数，它对于负裕度的惩罚增长很快，为指数增长(exponential growth)，参见图 13.5。

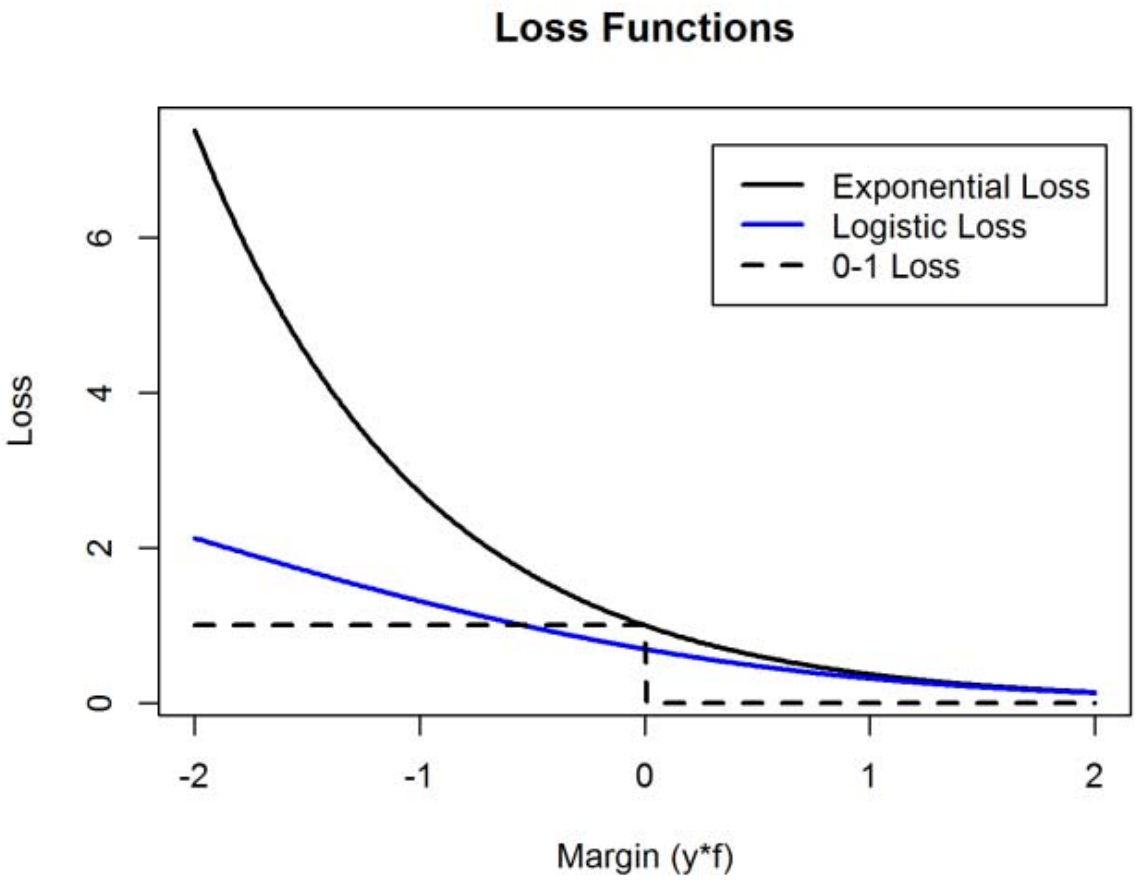

图 13.5 三种损失函数的比较

这意味着，算法对于较大的预测错误极为挑剔。

如果数据中的噪音较大(比如，贝叶斯错误率较高)或分类标签有误，则 AdaBoost 的稳健性较差。

在当代的提升法实践中，已不太常用 AdaBoost 的指数损失函数，而一般建议使用如下**逻辑损失函数**(logistic loss)：

$$L(y,\ f(x)) = \ln\left(1+e^{-yf(x)}\right) \qquad (13.57)$$

事实上，上式就是逻辑模型(Logit)的对数似然函数之负数，故也称为“二项偏离度”(Binomial deviance)；证明如下。

假定$y_i \in \{-1,1\}$，且服从 Logit 模型，则有：

$$P(y_i \mid \mathbf{x}_i)=\begin{cases}\dfrac{1}{1+e^{-f(\mathbf{x}_i)}} & if\ y_i=1\\ \dfrac{1}{1+e^{f(\mathbf{x}_i)}} & if\ y_i=-1\end{cases} \tag{13.58}$$

其中，两个条件概率之和为$\dfrac{1}{1+e^{-f(\mathbf{x}_i)}}+\dfrac{1}{1+e^{f(\mathbf{x}_i)}}=1$。

考虑到$y_i=1$或-1，故上式可写为

$$P(y_i \mid \mathbf{x}_i) = \begin{cases} \dfrac{1}{1+e^{-y_i f(\mathbf{x}_i)}} & if\ \ y_i = 1 \\ \dfrac{1}{1+e^{-y_i f(\mathbf{x}_i)}} & if\ \ y_i = -1 \end{cases} \tag{13.59}$$

此时，无论$y_i = 1$或-1，条件概率的表达式都一样，故可统一写为

$$P(y_i \mid \mathbf{x}_i) = \frac{1}{1+e^{-y_i f(\mathbf{x}_i)}} \qquad (i = 1, \cdots, n) \tag{13.60}$$

对于样本数据$\{\mathbf{x}_i, y_i\}_{i=1}^n$，在独立同分布(iid)的假设下，整个样本的似然函数为

$$\prod_{i=1}^{n} P(y_i \mid \mathbf{x}_i) = \prod_{i=1}^{n} \frac{1}{1+e^{-y_i f(\mathbf{x}_i)}} = \prod_{i=1}^{n} \left(1+e^{-y_i f(\mathbf{x}_i)}\right)^{-1} \qquad (13.61)$$

因此，样本数据的对数似然函数之负数为

$$\sum_{i=1}^{n} \ln\left(1+e^{-y_i f(\mathbf{x}_i)}\right) \qquad (13.62)$$

上式正是逻辑损失函数(13.57)。

逻辑损失函数对于负裕度($-yf(\mathbf{x})$)的惩罚比指数损失函数更为温和。

“0-1 损失函数”也称为**错分损失函数**(misclassification loss)，即如果分类正确(裕度为正)，则损失为 0；反之，如果分类错误(裕度为负)，则损失为 1。

0-1 损失函数为阶梯函数，并不光滑。逻辑损失函数与指数损失函数均可视为对 0-1 损失函数的光滑近似，而逻辑损失函数的近似效果更优。

13.7 多分类问题的交叉熵损失函数

对于多分类问题，可使用多项逻辑(Multinomial Logit)的对数似然函数之负数作为损失函数。

假设响应变量y的取值可分为K类，即$y \in \{1,2,\cdots,K\}$。

给定特征向量$\mathbf{x}_i$，假设“$y_i = k$”$(k = 1,\cdots,K)$的条件概率为

$$\mathrm{P}(y_i = k \mid \mathbf{x}_i) = \frac{\exp\{f_k(\mathbf{x}_i)\}}{\sum_{l=1}^{K}\exp\{f_l(\mathbf{x}_i)\}} \quad (k = 1,\cdots,K) \qquad (13.63)$$

整个样本的对数似然函数之负数为：

$$-\sum_{i=1}^{n}\sum_{k=1}^{K}\left[I(y_i=k)\cdot\ln \mathrm{P}(y_i=k\,|\,\mathbf{x}_i)\right] \qquad (13.64)$$

其中，$I(\cdot)$为示性函数。给定$\left\{f_1(\mathbf{x}),\cdots,f_K(\mathbf{x})\right\}$，则多分类问题的损失函数可写为

$$L\left(\left\{y_k,\,f_k(\mathbf{x})\right\}_{k=1}^{K}\right)=-\sum_{k=1}^{K}y_k\ln p_k(\mathbf{x}) \qquad (13.65)$$

其中，$y_k\equiv I(y=k)$为虚拟变量，而$p_k(\mathbf{x})\equiv \mathrm{P}(y=k\,|\,\mathbf{x})$为条件概率。

在计算机或机器学习领域，此损失函数也称为**交叉熵损失函数**(cross-entropy loss function)。

“交叉熵”是信息熵概念的推广，它度量样本的实际分布与预测分布之间的距离，参见本章附录。

由于逻辑模型(Logit)是多项逻辑模型(Multinomial Logit)的特例，故逻辑损失函数也称为**二值交叉熵损失函数**(binary cross-entropy loss function)。

13.8 随机梯度提升

在梯度提升法的基础上，Friedman (2002)进一步提出**随机梯度提升**(Stochastic Gradient Boosting)。

在进行随机梯度提升时，每次仅随机抽取部分子样本 (比如，二分之一样本)，称为**子抽样**(subsampling)，然后计算准残差，拟合基学习器。

与有放回且保持样本容量不变的自助抽样不同，子抽样为“无放回”(without replacement)抽样，且仅抽取一部分样本。

从数据的二维表结构来看，子抽样相当于随机抽取数据框的某些行，故也称为**行子抽样**(row subsampling)。

相对而言，随机森林的“随机特征选择”(random feature selection)，则为**列子抽样**(column subsampling)，即随机抽取部分变量。

子抽样的好处在于，每次仅随机抽取部分样本数据，故可进一步降低算法的学习速度，以预防过拟合，而且计算更快(因为每次仅用部分样本)。

子抽样使得每次用于估计的子样本不尽相同，故可降低基学习器之间的相关性，从而减少方差，提高预测准确率。

子抽样给算法带来的随机性，也有助于算法跳出损失函数可能存在的局部最小值。

由于子抽样仅使用部分数据估计决策树，故存在“袋外观测值”(out-of-bag observations)，可计算“袋外误差”(out-of-bag errors)，作为对测试误差的便捷估计。

在使用随机梯度下降时，如何选择子抽样的比例？

这涉及偏差与方差的权衡。在一个极端，如果子抽样的比例很小，则每次仅用少部分数据估计决策树，导致偏差较大。另一方面，如果子抽样的比例很大(接近于 1)，则偏差较小，但无法达到降低方差的效果。

子抽样比例也是一个调节参数，可通过交叉验证来确定，这增加了算法的灵活性。

提升法起源于 AdaBoost 算法。

AdaBoost 通过加大错误分类样本点的权重(reweighting)，迫使以后的基学习器修正之前基学习器的错误。

梯度提升法(GBM)则通过拟合之前基学习器的(准)残差，使得以后的基学习器修正之前基学习器的错误。

AdaBoost 仅适用于分类问题，为 GBM 的特例。

GBM 为通用算法，适用于一般的损失函数，还引入学习率、随机梯度提升，故一般预测效果更好。

13.9 回归提升树的 Python 案例

使用波士顿房价数据 boston(参见第 4 章)，演示回归提升树的 Python 操作。

* 详见教材，以及配套 Python 程序（现场演示）。

13.10 二分类提升树的 Python 案例

使用 Hastie et al. (2009)的垃圾邮件数据 `spam` (参见第 8 章)，演示二分类问题的梯度提升法。

* 详见教材，以及配套 Python 程序（现场演示）。

13.11 多分类提升树的 Python 案例

使用玻璃种类数据 `Glass` (参见第 6 章)，演示多分类问题的梯度提升法。

* 详见教材，以及配套 Python 程序（现场演示）。

13.12 XGBoost 算法

在使用梯度提升法时，若样本容量较大或为大数据，则推荐使用陈天奇团队开发的 Python 模块 xgboost(Chen and Guestrin, 2016)。

其中，xgboost 表示 Extreme Gradient Boosting(极端梯度提升)。

xgboost 模块依然使用梯度提升法，但在具体算法上作了许多改进，包括使用牛顿法计算下降方向，改进决策树的算法，使用稀疏矩阵(sparse matrix)等，使得运算速度大幅提升。

与随机森林类似，xgboost 也允许进行“列子抽样”(column subsampling)，即“随机特征选择”(random feature selection)。

另外，xgboost 还可用线性回归作为基学习器。

分别以波士顿房价数据 `boston`(参见第 4 章)与 `spam` 数据(参见第 8 章)为例进行演示。

* 详见教材，以及配套 Python 程序（现场演示）。

附录

A 13.1 交叉熵损失函数

假设某随机变量共分K类，取值分别为$1,\cdots,K$，而其相应的概率为$(p_1,\cdots,p_K)$，其中$p_k \geq 0$，且$\sum_{k=1}^{K} p_k = 1$。

信息熵(information entropy)或“熵”的定义为

$$\text{Entropy}(p_1,\cdots,p_K) \equiv -\sum_{k=1}^{K} p_k \log p_k \qquad (13.66)$$

特别地，如果$K=2$，则信息熵为

$$\text{Entropy}(p,1-p)\equiv-\left[p\log p+(1-p)\log(1-p)\right] \quad (13.67)$$

根据信息理论(information theory)，对于概率分布$(p_1,\cdots,p_K)$，信息熵是在传送其状态信息时，平均所需的最少字节数。

但我们通常并不知道真实分布$(p_1,\cdots,p_K)$，而使用样本数据来估计$(\hat{p}_1,\cdots,\hat{p}_K)$。

根据估计的概率分布$(\hat{p}_1, \cdots, \hat{p}_K)$进行信息传输，平均所需要的字节数，即为**交叉熵**(cross-entropy)：

$$cross\text{-}entropy \equiv -\sum_{k=1}^{K} p_k \log \hat{p}_k \qquad (13.68)$$

交叉熵一定大于或等于熵，因为前者使用“错误”的概率分布，故需要更多的字节数来传递信息。

将交叉熵减去熵，即为估计的概率分布$(\hat{p}_1, \cdots, \hat{p}_K)$离开真实分布$(p_1, \cdots, p_K)$的所谓 KL 距离(Kullback–Leibler divergence)：

$$\mathrm{KL}(\mathbf{p}\|\hat{\mathbf{p}}) \equiv -\sum_{k=1}^{K} p_k \log \hat{p}_k + \sum_{k=1}^{K} p_k \log p_k = \sum_{k=1}^{K} p_k \log\left(\frac{p_k}{\hat{p}_k}\right) \geq 0 \tag{13.69}$$

其中，$\mathbf{p} \equiv (p_1 \cdots p_K)'$，而$\hat{\mathbf{p}} \equiv (\hat{p}_1 \cdots \hat{p}_K)'$。

直观上，交叉熵与 KL 距离从信息的角度衡量两个概率分布之间的距离。

特别地，如果$K = 2$，则交叉熵为

$$-\left[p \log \hat{p} + (1-p)\log(1-\hat{p})\right] \tag{13.70}$$

对于样本数据$i=1,\cdots,n$，则整个样本的交叉熵为：

$$-\sum_{i=1}^{n}\left[p_i \log \hat{p}_i+(1-p_i)\log(1-\hat{p}_i)\right] \qquad (13.71)$$

这正是**二值交叉熵损失函数**(binary cross-entropy loss function)。

最小化交叉熵损失函数，等价于最大化对数似然函数(即最大似然估计)。

不失一般性，考虑二分类问题，对于实际观测到的第i个观测值，则p_i为 0 或 1(两类中必居其一)。

第i个观测值的似然函数可写为

$$\begin{cases} \hat{p}_i & if\ \ p_i = 1 \\ 1-\hat{p}_i & if\ \ p_i = 0 \end{cases} \tag{13.72}$$

写为更紧凑的形式：

$$\hat{p}_i^{\,p_i}(1-\hat{p}_i)^{1-p_i} \tag{13.73}$$

对于相互独立的样本，整个样本的似然函数为

$$L = \prod_{i=1}^{n} \hat{p}_i^{\,p_i}(1-\hat{p}_i)^{1-p_i} \tag{13.74}$$

故对数似然函数为

$$\max \ \ln L=\sum_{i=1}^{n}\left[p_i \log \hat{p}_i+(1-p_i)\log(1-\hat{p}_i)\right] \qquad (13.75)$$

最大化对数似然函数等价于最小化其负数：

$$\min \ -\ln L=-\sum_{i=1}^{n}\left[p_i \log \hat{p}_i+(1-p_i)\log(1-\hat{p}_i)\right] \qquad (13.76)$$

上式正是整个样本的交叉熵。