第十二章　Boosting 方 法

12.1 简介

12.1 简 介

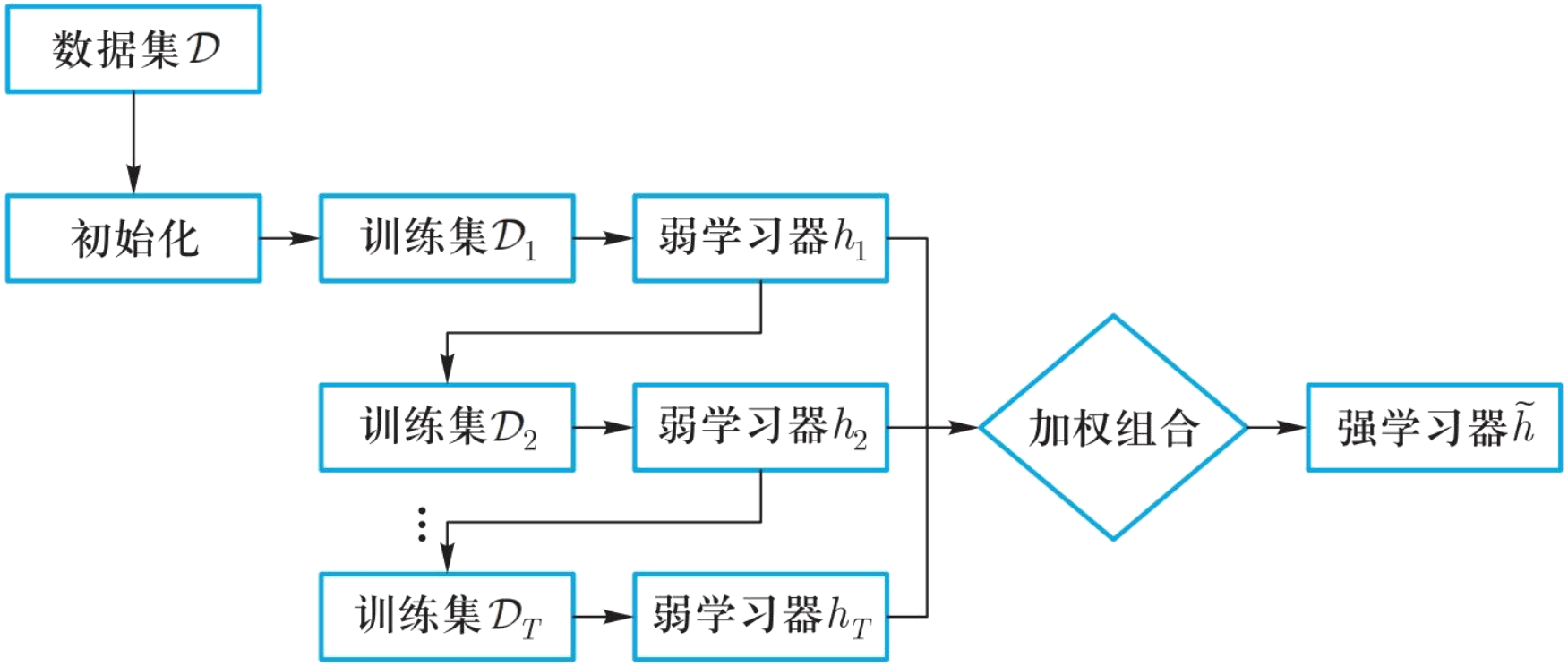
- Boosting 是另一种集成学习方法, 它通过迭代训练一系列弱学习器, 每个学习器都试图纠正前一个学习器的错误. 在每一轮中, 对之前错误分类的样本增加权重, 使其在下一轮中更受关注. 与随机森林选择随机特征和直接投票不同, Boosting 算法使用所有特征进行训练, 对预测结果进行加权投票.
- 如何根据历史数据, 建立精确的预测模型, 是机器学习方法研究的主要目的之一. 例如, 我们拟构建一个可以区分垃圾邮件和正常邮件的电子邮件过滤器. 通常机器学习方法解决此问题的思路如下：首先收集尽可能多的垃圾电子邮件和非垃圾电子邮件的案例. 然后, 应用已有的机器学习算法对收集到的邮件和其对应的标签进行训练, 建立一个分类准则, 基于已知数据建立的准则可以对新的、未加标签的电子邮件进行自动的分类. 一个自然的期望是建立的分类 (预测) 准则能够对新的电子邮件做出精确的预测.
- 通常, 建立一个高度准确的预测或分类准则往往是困难的. 但是, 建立系列相对准确、比随机猜测稍好的经验法则就容易实现得多. 对上例的电子邮件问题, 可以建立这样一个简单规则：如果电子邮件中出现“立即购买”的短语, 那么就预测它是垃圾邮件.

12.1 简 介

- Boosting 方法正是基于这样一种考虑：找到许多粗糙的、比随机猜测稍好的经验准则要比找到单一的、高度精确的预测规则容易得多. 这些粗糙的、比随机猜测稍好的经验准则或算法, 通常被称为弱学习算法或者基学习算法.
- Boosting 算法反复调用这些弱学习算法, 每次让其学习训练样本的不同子集. 更准确地说, 是训练数据的不同分布或权重. 每次调用时, 基学习算法都会生成一个新的弱学习算法 (学习器). 重复建立多个弱学习算法后, Boosting 算法最终将这些弱学习算法 (学习器) 有效组合成最后的强学习算法 (强学习器), 强学习器有望比任何一个弱学习器都准确得多.
- 因此, Boosting 的**核心思想**就是：对于一个复杂的任务而言, 有效地综合多个专家的预测进而所得出的新的预测, 要优于其中任何一个专家的单独预测. 即通常所说的“**三个臭皮匠顶个诸葛亮**”的道理.

12.1 简 介

- Boosting 集成学习方法基本流程如图 12.1 所示.



12.1 简 介

- 当然, 要使上述 Boosting 方法切实可行, 还有以下**两个基本问题**需要解决:
  - 1. 在每一轮训练中, 如何改变训练数据的权值或概率分布;
  - 2. 最终, 如何将所得到的弱分类器组合成为一个强学习器.
- 关于第 1 个问题, Boosting 算法家族中的代表性算法 AdaBoost 的处理方式是: 提高那些被前一轮弱学习器错误分类样本的权值, 而降低那些被正确分类样本的权值. 这样一来, 那些没有得到正确分类的数据, 由于其权值的加大而受到后一轮弱学习器的更大关注. 于是得到的系列弱学习器之间具有“互补”的特点. 至于第 2 个问题, AdaBoost 算法通常采取加权投票的方法.具体地, 增加分类误差率小的弱学习器的权值, 使其在表决中起较大作用; 减少分类误差率大的弱学习器的权值, 使其在表决中起较小的作用.

12.1.1 Boosting 方法起源

- Boosting 方法起源于一个纯理论性的公开问题, 即所谓的 Boosting **公开问题**.
- 1984 年, Valiant[135] 在研究概率近似正确 (probably approximately correct, PAC) 学习框架时, 给出了强可学习 (或称可学习) 和弱可学习的概念.1988 年, Kearns 和 Valiant[136] 在研究 PAC 学习模型时, 针对以上弱可学习与强可学习提出如下公开问题: 是否可以将性能仅略好于随机猜测的弱学习算法“Boosting”提升为具有任意精确度的强学习算法? 即 Boosting 公开问题. 此问题非常重要, 因为获得一个弱学习器要比获得一个强学习器要容易得多. 如果该问题的答案是肯定的, 那么任何弱学习器都有可能被提升为强学习器. 这一思想对今后机器学习算法尤其是集成学习的发展产生了深远的影响.
- 1989年,Schapire给出了肯定的答复, 在PAC框架下, 强可学习与弱可学习是等价的, 即:一个概念是强可学习的充要条件是这个概念是弱可学习的.同时, 他在文章中给出的构造性证明也成为最早的Boosting算法[137]. 随后,在1990年, Freund开发了一个效率更高的且具有某种最优性的Boosting 算法[138].Drucker等人利用这些早期的Boosting方法在光学字符识别(optical character recognition,OCR)任务上进行了首次实验验证[139].然而, 由于上述算法需要提前知晓弱学习器的错误率上界, 这通常在实际应用中是未知的.因此上述 Boosting 算法并不具备实际应用性.

12.1.1 Boosting 方法起源

- Freund 与 Schapire 在随后的研究中发现, “Online”学习与 Boosting 问题之间存在着极大的共性. 他们将其与加权投票的相关研究成果进行融合,并在 Boosting 问题中进行相应推广, 得到了著名的 AdaBoost 算法. 特别地,此算法不需要提前预知弱学习器的分类精度等相关的任何先验知识, 在实践中获得了极大成功. 凭借此工作, Freund 和 Schapire 获得了 2003 年度理论计算机的最高奖——哥德尔奖 (Godel prize). AdaBoost 一举成为最具影响力的集成算法之一, 被评为数据挖掘十大算法之一[140].

12.1.2 AdaBoost 算法

1. AdaBoost 通用算法

- AdaBoost 算法可以做出非常精准的预测, 其过程却非常简单. 现举例如下:
- 以二元问题为例, 设 $\mathcal{X}$ 为自变量组成的样本空间, 其中的样本都是从分布 $\mathcal{D}$ 中随机抽取, 且满足独立同分布性; 设 $\mathcal{X}$ 由 $\mathcal{X}_1$、$\mathcal{X}_2$ 和 $\mathcal{X}_3$ 三部分组成,每个部分占 1/3, 通过随机猜测工作的学习器在这个问题上有 0.5 的分类误差. 我们想在这个问题上得到一个精确的 (例如零误差) 学习器. 可借助的只有一个弱学习器 h_1, 它在样本空间 $\mathcal{X}_1$ 和 $\mathcal{X}_2$ 中有正确的分类, 在 $\mathcal{X}_3$ 有错误的分类. 如何将此弱学习器 h_1 “Boosting”为强学习器呢?
- 一个自然的想法就是纠正 h_1 所犯的错误. 首先, 通过 $\mathcal{D}$ 派生出新的分布 $\mathcal{D}_1$. 例如通过提高那些被 h_1 错误学习样本的权值, 降低那些被 h_1 正确预测的样本的权值诱导出新分布 $\mathcal{D}_1$. 显然在 $\mathcal{D}_1$ 上, h_1 的错误被彰显. 然后用D1 训练得到学习器 h_2. 此时得到的学习器 h_2 极有可能也是一个弱学习器.假设它在 $\mathcal{X}_1$ 和 $\mathcal{X}_3$ 中有正确的预测, 在$\mathcal{X}_2$ 有错误的预测.通过以某种适当的方式组合 h_1 和 h_2, 组合的学习器将在$\mathcal{X}_1$ 中具有正确的预测, 并且可能在$\mathcal{X}_2$ 和 $\mathcal{X}_3$ 中仍有错误.

12.1.2 AdaBoost 算法

- 类似地, 为了使组合学习器的错误彰显, 我们再次派生出一个新的分布 $\mathcal{D}_2$, 并从 $\mathcal{D}_2$ 训练出新的学习器 h_3, 使 h_3 在 $\mathcal{X}_2$ 和 $\mathcal{X}_3$ 有正确的预测. 最后, 通过组合 h_1、h_2 和 h_3, 就得到一个强学习器, 因为在 $\mathcal{X}_1$、$\mathcal{X}_2$ 和 $\mathcal{X}_3$ 的每个空间中, 至少有两个学习器是正确的.

- 简而言之, Boosting 方法就是顺序训练一族弱学习器, 并将它们组合以形成强学习器来进行预测. 训练过程中, 让后建立的学习器更多地关注前序学习器的错误预测样本. 通用 Boosting 算法如下:

Boosting 通用算法

输入: 样本权值分布 $\mathcal{D}$;
基学习算法 $\mathcal{L}$;
学习轮数 T.

过程:

1. $\mathcal{D}_1 = \mathcal{D}$; % 初始化分布
2. **for** $t = 1, 2, \cdots, T$:
3. $h_t = \mathcal{L}(\mathcal{D}_t)$; % 依分布训练弱学习器
4. $\mathrm{err}_{\mathcal{D}_t} = P_{\mathcal{D}_t}(h_t(\boldsymbol{X}) \neq Y)$; % 度量误差
5. $\mathcal{D}_{t+1} = \mathrm{Adjust\_Distribution}(\mathcal{D}_t, \mathrm{err}_{\mathcal{D}_t})$; % 更新分布
6. **end**

输出: $\mathcal{H}(\boldsymbol{X}) = \mathrm{Combine\_Outputs}(\{h_1(\boldsymbol{X}), \cdots, h_T(\boldsymbol{X})\})$

12.1.2 AdaBoost 算法

2. AdaBoost 具体算法

- 对上述 Boosting 通用算法, 将 Adjust Distribution(·, ·) 和 CombineOutputs (, · · · ,) 取成不同的形式, 则会衍生出各种类型的 Boosting 算法, 如AdaBoost.M1、AdaBoost.MR、FilterBoost、GentleBoost、GradientBoost、Log-itBoost 等, 进而形成庞大的 Boosting 算法族. 我们下面重点介绍 AdaBoost算法.

- 假设$\mathcal{X}\subset\mathbf{R}^{n\times p}$是自变量生成的样本空间,$\mathcal{Y}$是标签集, 通常假设$\mathcal{Y}=\{-1,1\}$. 给定一个二元训练数据集$\{(\boldsymbol{X}_1,Y_1),(\boldsymbol{X}_2,Y_2),\cdots,(\boldsymbol{X}_n,Y_n)\}$ 每个训练数据由自变量样本与标签组成, 其中$\boldsymbol{X}_i\in\mathcal{X}$为自变量样本空间,$Y_i\in\mathcal{Y}$为标签, $i=1,2,\cdots,n$. AdaBoost 算法如下.

AdaBoost 算法

输入： 训练数据集 $\{(\boldsymbol{X}_1,Y_1),(\boldsymbol{X}_2,Y_2),\cdots,(\boldsymbol{X}_n,Y_n)\}$;
基学习算法 $\mathcal{L}$;
学习轮数 T.

过程：

1. $\mathcal{D}_1(\boldsymbol{X}_i)=1/n$; % 初始化权重分布
2. **for** $t=1,2,\cdots,T$:
3. $h_t(\boldsymbol{X})=\mathcal{L}(\mathcal{D},\mathcal{D}_t)$, 且 $h_t(\boldsymbol{X}):\mathcal{X}\to\{-1,1\}$; % 依分布训练输出弱学习器
4. $\mathrm{err}_{\mathcal{D}_t}=\sum_{i=1}^{n}P_{\boldsymbol{X}_i\sim\mathcal{D}_t}(h_t(\boldsymbol{X}_i)\neq Y_i)$; % 度量误差
5. **if** $\mathrm{err}_{\mathcal{D}_t}\geqslant 0.5$ **then break**
6. $\alpha_t=\frac{1}{2}\ln\left(\frac{1-\mathrm{err}_{\mathcal{D}_t}}{\mathrm{err}_{\mathcal{D}_t}}\right)$; % 计算弱学习器 h_t 的权重
7. $\mathcal{D}_{t+1}(\boldsymbol{X}_i)=\frac{\mathcal{D}_t(\boldsymbol{X}_i)}{Z_t}\exp(-\alpha_tY_ih_t(\boldsymbol{X}_i))$, 其中
$Z_t=\sum_{i=1}^{n}\mathcal{D}_t(\boldsymbol{X}_i)\exp(-\alpha_tY_ih_t(\boldsymbol{X}_i))$; % 更新分布, Z_t 为规范化因子
8. **end**

输出： $\mathcal{H}(\boldsymbol{X})=\mathrm{sign}\left(\sum_{t=1}^{T}\alpha_th_t(\boldsymbol{X})\right)$

12.1.2 AdaBoost 算法

注 12.1 上述算法通常被称为“离散型 AdaBoost”, 因为弱学习器 $h_t(\boldsymbol{X})$返回离散的标签. 若弱学习器返回的预测值为连续实数, 则可以对 AdaBoost做恰当的修改, 参见文献 [141]. 此外, 上述算法中的关键量 h_t 和 α_t 的选取原理将在下一章给出具体的解释.

注 12.2 对上述 AdaBoost 算法做一些简单说明:

- (1) 为保证第 1 步能够输出基本学习器 $h_1(\boldsymbol{X})$, 我们需要假设原始训练数据具有已知的权值分布. 为简单起见, 通常假定训练数据具有均匀分布.
- (2) 根据已有的符号表示, 可以将训练数据权值的分布写成如下形式:

$$\mathcal{D}_t = \left(\omega_1^{(t)}, \omega_2^{(t)}, \cdots, \omega_i^{(t)}, \cdots, \omega_n^{(t)}\right), \quad t = 1, 2, \cdots, T, \tag{12.1.1}$$

其中, $\omega_i^{(1)} = \frac{1}{n}, i = 1, 2, \cdots, n.$

- (3) 计算基本学习器 $h_t(\boldsymbol{X})$ 在分布为 $\mathcal{D}_t$ 的训练数据集上的误差率:

$$\mathrm{err}_{\mathcal{D}_t} = \sum_{i=1}^{n} P_{\boldsymbol{X}_i \sim \mathcal{D}_t}\left(h_t\left(\boldsymbol{X}_i\right) \neq Y_i\right) = \sum_{h_t\left(\boldsymbol{X}_i\right) \neq Y_i} \omega_i^{(t)}. \tag{12.1.2}$$

此式表明弱学习器 $h_t(\boldsymbol{X})$ 在加权训练数据集上的分类误差等于被$h_t(\boldsymbol{X})$ 误分的样本权值之和.

12.1.2 AdaBoost 算法

- (4) 上述算法给出弱学习器的系数表达式为

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\mathrm{err}_{\mathcal{D}_t}}{\mathrm{err}_{\mathcal{D}_t}}\right). \tag{12.1.3}$$

显然当 $\mathrm{err}_{\mathcal{D}_t} \leqslant 0.5$ 时, 其系数 $\alpha_t \geqslant 0$, 且 α_t 为 $\mathrm{err}_{\mathcal{D}_t}$ 的减函数. 所以误差率 $\mathrm{err}_{\mathcal{D}_t}$越小的学习器其权重 α_t 就越大.

- (5) 根据算法中的第 7 个式子以及式 (12.1.1) 可知, 更新权值分布可以写成

$$\omega_i^{(t+1)} = \frac{\omega_i^{(t)}}{Z_t}\exp\left(-\alpha_t Y_t h_t\left(\boldsymbol{X}_i\right)\right) = \begin{cases} \dfrac{\omega_i^{(t)}}{Z_t}\mathrm{e}^{-\alpha}, & h_t\left(\boldsymbol{X}_i\right) = Y_i, \\ \dfrac{\omega_i^{(t)}}{Z_t}\mathrm{e}^{\alpha_t}, & h_t\left(\boldsymbol{X}_i\right) \neq Y_i. \end{cases} \tag{12.1.4}$$

12.1.2 AdaBoost 算法

其中

$$
\begin{aligned}
Z_t &= \sum_{i=1}^{n} \mathcal{D}_t(\boldsymbol{X}_i)\exp\left(-\alpha_t Y_t h_t(\boldsymbol{X}_i)\right) \\
&= \sum_{i=1}^{n} \omega_i^{(t)} \exp\left(-\alpha_t Y_t h_t(\boldsymbol{X}_i)\right) \qquad (12.1.5) \\
&= \sum_{Y_i = h_t(\boldsymbol{X}_i)} \omega_i^{(t)} \mathrm{e}^{-\alpha_t} + \sum_{Y_i \neq h_t(\boldsymbol{X}_i)_i} \omega_i^{(t)} \mathrm{e}^{\alpha_t}.
\end{aligned}
$$

式 (12.1.4) 说明, 若样本 $\boldsymbol{X}_i$ 被正确学习, 即 $h_t(\boldsymbol{X}_i) = Y_i$, 则其权重会减小;若 $\boldsymbol{X}_i$ 被错误学习, 即 $h_t(\boldsymbol{X}_i) \neq Y_i$, 则其权重会增大. 这样, 误分学习样本在下一轮学习中就会受到基学习器的更大关注而起到更大作用.

- (6) 记

$$
\tilde{h}_T(\boldsymbol{X}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{X}), \qquad (12.1.6)
$$

12.1.2 AdaBoost 算法

则 AdaBoost 的最后输出结果为

$$\mathcal{H}(\boldsymbol{X}) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{X})\right). \tag{12.1.7}$$

此式表明对于任何数据的分类采用加权投票表决的方法, 其中 $\mathcal{H}(\boldsymbol{X})$ 的符号决定数据 $\boldsymbol{X}$ 的类. 需要注意的是, 基本学习器 $h_t(\boldsymbol{X})$ 的系数 α_t 未必具有归一性.

- (7) AdaBoost 具备以下两个特点:
- (a) AdaBoost 不更新训练数据, 只是不断更新训练数据的权值分布, 使得训练数据在基本分类器的学习中起不同的作用.
- (b) AdaBoost 最终分类器 $\tilde{h}_T(\boldsymbol{X})$ 的构成是利用基本分类器 h_t, $t = 1, 2, \cdots, T$的线性组合构建, 因而可视其为加法模型.

12.1.3 AdaBoost 实例

例 12.1 给定如表 12.1 所示训练数据. 假设弱学习器 (或称阈值函数)由 $X < v$ 或 $X \geqslant v$ 产生, 其阈值 v 使该学习器在训练数据集上分类误差率最低. 试用 AdaBoost 算法学习一个强学习器.

序号	1	2	3	4	5	6	7	8	9	10
X	1	2	3	4	5	6	7	8	9	10
Y	1	1	1	−1	1	−1	−1	1	−1	−1

解 初始化数据权值分布

$$\mathcal{D}_t = \left(\omega_1^{(t)}, \omega_2^{(t)}, \cdots, \omega_i^{(t)}, \cdots, \omega_n^{(t)}\right),$$

$$\omega_1^{(t)} = 0.1, \quad i = 1, 2, \cdots, 10.$$

▶ 1. 对 $t = 1$:

(1) 在权值分布为 D1 的训练数据上, 阈值 v 取 3.5 时分类误差率最低,故基本分类器为

12.1.3 AdaBoost 实例

$$h_1(X)=\begin{cases}1, & X<3.5,\\ -1, & X\geq 3.5.\end{cases}$$

(2) $h_1(X)$ 在训练数据集上的误差率：$\mathrm{eer}_{\mathcal{D}_1}=\sum_{i=1}^{n}P_{X_i\sim\mathcal{D}_1}\left(h_1(X_i)\neq Y_i\right)=0.2.$

(3) 计算 h1(X) 的系数：$\alpha_1=\frac{1}{2}\ln\frac{1-\mathrm{eer}_{\mathcal{D}_1}}{\mathrm{eer}_{\mathcal{D}_1}}=0.6931.$

(4) 更新训练数据的权值分布：

$$\mathcal{D}_2=\left(\omega_1^{(2)},\omega_2^{(2)},\cdots,\omega_n^{(2)}\right),$$

$$\omega_i^{(2)}=\frac{\omega_i^{(1)}}{Z_1}\exp\left(-\alpha_1 Y_i h_1(X_i)\right),i=1,2,\cdots,10,$$

$$\mathcal{D}_2=(0.0625,0.0625,0.0625,0.0625,0.2500,0.0625,0.0625,0.2500,0.0625,0.0625),$$

$$\tilde{h}_1(X)=0.693h_1(X).$$

分类器 sign($\tilde{h}_1$ (X)) 在训练数据集上有 2 个误分类点.

12.1.3 AdaBoost 实例

▶ 2. 对 $t=2$:

(1) 在权值分布为 $\mathcal{D}_2$ 的训练数据上, 阈值 v 是 8.5 时分类误差率最低,基本分类器为

$$h_2(X)=\begin{cases}1, & X<8.5,\\ -1, & X\geq 8.5.\end{cases}$$

(2) $h_2(X)$ 在训练数据集上的误差率 $\mathrm{eer}_{\mathcal{D}_2}=0.1875.$

(3) 计算 $\alpha_2=0.7332.$

(4) 更新训练数据权值分布:

$$\mathcal{D}_3=(0.0385,0.0385,0.0385,0.1667,0.1539,0.1667,0.1667,0.1539,0.0385,0.0385),$$

$$\tilde{h}_2(X)=0.693h_1(X)+0.7332h_2(X).$$

分类器 sign($\tilde{h}_2(X)$) 在训练数据集上有 3 个误分类点.

12.1.3 AdaBoost 实例

▶ 3. 对 $t=3$:

(1) 在权值分布为 $\mathcal{D}_3$ 的训练数据上, 阈值 v 是 5.5 时分类误差率最低,基本分类器为

$$h_3(X)=\begin{cases}1, & X<5.5,\\ -1, & X\geq 5.5.\end{cases}$$

(2) $h_3(X)$ 在训练数据集上的误差率 $\mathrm{eer}_{\mathcal{D}_3}=0.3206.$

(3) 计算 $\alpha_3=0.3755.$

(4) 更新训练数据权值分布:

$$\mathcal{D}_3=(0.028,0.028,0.028,0.260,0.113,0.123,0.123,0.240,0.028,0.028),$$

$$\tilde{h}_2(X)=0.4236h_1(X)+0.6496h_2(X)+0.3755h_3(X).$$

分类器 sign($\tilde{h}_3(X)$) 在训练数据集上有 0 个误分类点.

12.1.3 AdaBoost 实例

- 于是最终分类器为

$$\mathcal{H}(X)=\operatorname{sign}\left(\tilde{h}_3(X)\right)=\operatorname{sign}\left\{0.4236h_1(X)+0.6496h_2(X)+0.3755h_3(X)\right\}.$$

12.2　AdaBoost 算法的误差分析

12.2.1 daBoost 算法的训练误差

- AdaBoost 最基本的理论性质涉及其减少训练误差的能力, 即训练集上的误差率. Freund 和 Schapire 给出了误差界[142]:

- **定理 12.1** 若 AdaBoost 在每一轮迭代中生成弱分类器 h_t 的误差率是 $\mathrm{eer}_{\mathcal{D}_t}$, $t = 1, 2, \cdots, T$, 则最终分类器 $\mathcal{H}(\boldsymbol{X})$ 的训练误差率

$$\mathrm{err}_{\mathcal{D}} = \sum_{i=1}^{n} P_{\boldsymbol{X}_i \sim \mathcal{D}} \left(\mathcal{H}\left(\boldsymbol{X}_i\right) \neq Y_i \right)$$

  - 满足

$$\mathrm{err}_{\mathcal{D}} \leq 2^T \prod_{t=1}^{T} \sqrt{\mathrm{err}_{\mathcal{D}_t} \left(1 - \mathrm{err}_{\mathcal{D}_t}\right)}.$$

- 此定理的证明具有一定的技巧性与特殊性, 因而在后续研究中极少被采用. 随后 Schapire 和 Singer 进行了更深入研究[143], 得到如下结论:

12.2.1 daBoost 算法的训练误差

- **定理 12.2** AdaBoost 算法最终分类器的训练误差满足

$$\frac{1}{n}\sum_{i=1}^{n} I\left(\mathcal{H}\left(\boldsymbol{X}_i\right) \neq Y_i\right) \leq \frac{1}{n}\sum_{i=1}^{n} \exp\left(-Y_i \tilde{h}_T\left(\boldsymbol{X}_i\right)\right) = \prod_{t=1}^{T} Z_t, \tag{12.2.1}$$

  - 其中 $I()$ 是示性函数, Z_t, $\tilde{h}_T(\boldsymbol{X})$ 和 $\mathcal{H}(\boldsymbol{X})$ 分别见式 (12.1.5), (12.1.6) 和(12.1.7).

- **证明** 当 $\tilde{h}_T(\boldsymbol{X}_i) = Y_i$ 时, $Y_i \tilde{h}_T(\boldsymbol{X}_i) < 0$, 因而 $\exp(-Y_i \tilde{h}_T(\boldsymbol{X}_i)) \geqslant 1$.由此可知 (12.2.1) 的不等式成立.

- 下面证明 (12.2.1) 中等式是成立的. 首先, 由式 (12.1.4) 可知:

$$\omega_i^{(t)} \exp\left(-\alpha_t Y_i h_t\left(\boldsymbol{X}_i\right)\right) = Z_t \omega_i^{(t+1)} \qquad i = 1, 2, \cdots, n.$$

  - 根据上式得到:

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{n} \exp\left(-Y_i \tilde{h}_T\left(\boldsymbol{X}_i\right)\right) &= \frac{1}{n}\sum_{i=1}^{n} \exp\left(-\sum_{i=1}^{n} \alpha_t Y_i h_t\left(\boldsymbol{X}_i\right)\right) \\ &= \sum_{i=1}^{n} \omega_i^{(1)} \prod_{t=1}^{T} \exp\left(-\alpha_t Y_i h_t\left(\boldsymbol{X}_i\right)\right)\end{aligned}$$

12.2.1 daBoost 算法的训练误差

$$
\begin{aligned}
&= Z_1 \sum_{i=1}^{n} \omega_i^{(2)} \prod_{t=2}^{T} \exp\left(-\alpha_t Y_i h_t\left(\boldsymbol{X}_i\right)\right) \\
&= Z_1 Z_2 \sum_{i=1}^{n} \omega_i^{(3)} \prod_{t=3}^{T} \exp\left(-\alpha_t Y_i h_t\left(\boldsymbol{X}_i\right)\right) \qquad (12.2.2) \\
&= \cdots \\
&= Z_1 Z_2 \cdots Z_{T-1} \sum_{i=1}^{n} \omega_i^{(T)} \prod_{t=3}^{T} \exp\left(-\alpha_T Y_i h_T\left(\boldsymbol{X}_i\right)\right).
\end{aligned}
$$

- 再由式 (12.1.5) 中 Z_t 的定义可知:

$$
Z_T = \sum_{i=1}^{n} \omega_i^{(T)} \exp\left(-\alpha_T Y_i h_T\left(\boldsymbol{X}_i\right)\right), \qquad (12.2.3)
$$

- 证毕.

12.2.1 daBoost 算法的训练误差

- **注 12.3** (1) 定理 12.2的左边是训练误差, 右边是所有归一化因子 $Z_t(t=1,2,\cdots,T)$ 的乘积. 通过上式可以知道, 要想得到对训练样本拟合精度高的强学习器, 需要选择弱学习器 $h_t(\boldsymbol{X})$ 以及其权值 α_t, 使 $\prod_{t=1}^{T} Z_t$ 最小化.虽然上式给出的是相对松弛的训练误差界, 但因其更好的解释性和可操作性,可得到更广泛的应用.

  (2) 要实现 $\prod_{t=1}^{T} Z_t$ 的最小化, 每加入一个新的弱学习器, 都可能要修改已有弱学习器的集成方式, 其复杂度太高. AdaBoost 的思想是不改变已有预测准则的形式, 以线性加和的方式加入新的弱学习器, 只最小化当前迭代的归一化因子 Z_t.

- **定理 12.3**

$$\prod_{t=1}^{T} Z_t = 2^T \prod_{t=1}^{T} \sqrt{\mathrm{err}_{\mathcal{D}_t}\left(1-\mathrm{err}_{\mathcal{D}_t}\right)} = \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right), \qquad (12.2.4)$$

  ▶ 其中 $\gamma_t = \frac{1}{2} - \mathrm{eer}_{\mathcal{D}_t}$, 称为 h_t 的边界 (edge).

- 本定理证明过程略, 感兴趣的读者可参考文献 [144].

12.2.1 daBoost 算法的训练误差

- **推论 12.1** 若存在 $\gamma > 0$, 对所有 t 有 $\gamma_t \geqslant \gamma$, 则

$$\frac{1}{n}\sum_{i=1}^{n} I\left(\mathcal{H}\left(\boldsymbol{X}_i\right) \neq Y_i\right) \leq \exp\left(-2T\gamma^2\right). \tag{12.2.5}$$

- 因为 $\mathrm{eer}_{\mathcal{D}_t} = \frac{1}{2} - \gamma_t$, 说明边界 γ_t 度量第 t 个弱学习器 h_t 的误差率比随机猜测的错误率好.
- 此推论告诉我们这样一个事实：只要弱学习器的学习能力比随机猜测稍好 $\left(\gamma_t = \frac{1}{2} - \mathrm{eer}_{\mathcal{D}_t} \geq \gamma > 0\right)$, 那么通过 AdaBoost 输出的强学习器 $\mathcal{H}$ 的训练误差就是随着轮数 T 呈指数下降的. 例如令 $\gamma = 0.1$, 即所有弱学习器的误差率不超过 0.4, 则上式表明强预测准则 $\mathcal{H}$ 的训练误差率至多是

$$\left(\sqrt{1 - 4 \times 0.1^2}\right)^T \approx \left(0.9798\right)^T.$$

- 但其实 AdaBoost 算法及其分析并不需要知道这个下界 γ, 而且能适应弱学习器各自的训练误差率, 由此获得了自适应 Boosting 的名称, 即 AdaBoost(Ada 是 Adaptive 的简写). 如果某些 γ_t 很大, 那么训练误差界的减少将会更大.

12.2.1 daBoost 算法的训练误差

- 此外, 通过此推论可知：若想使训练误差率 $\mathrm{err}_{\mathcal{D}} \leqslant \varepsilon$, 则可让训练轮数 T 为

$$\frac{1}{2\gamma^2}\ln\frac{1}{\varepsilon}.$$

  - 上述式子说明可以通过有限个学习器就可以使得训练误差足够小.

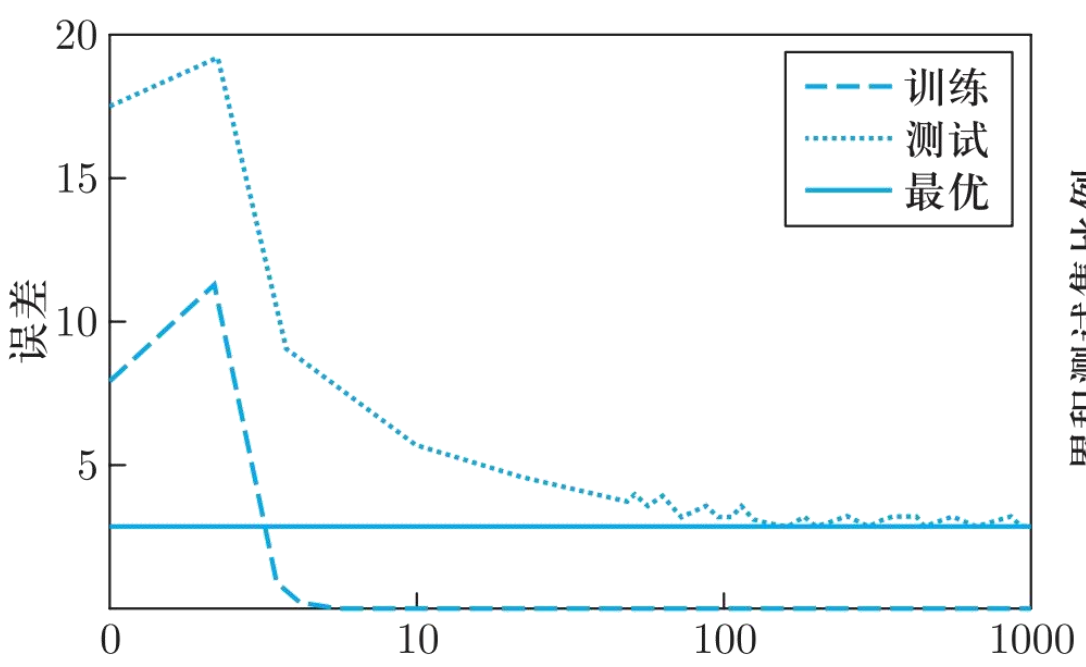
12.2.2 AdaBoost 算法的泛化误差

- 定理 12.1, 12.2 和 12.3 讨论了 AdaBoost 算法训练误差的下界. 然而, 在机器学习中我们真正关心的是它在测试数据上的泛化能力. 事实上, 任一种能够降低训练误差的算法不一定有资格作为 Boosting 算法. Boosting 算法是一种可以使泛化误差 (generalization error) 任意接近零的算法. 直观来说, 它是一种对测试数据具有接近完美的预测能力的学习算法. 当然, 以上结论的成立需要对算法提供合理规模的训练样本, 所使用的弱学习算法也一直有比随机猜测好的弱学习器.

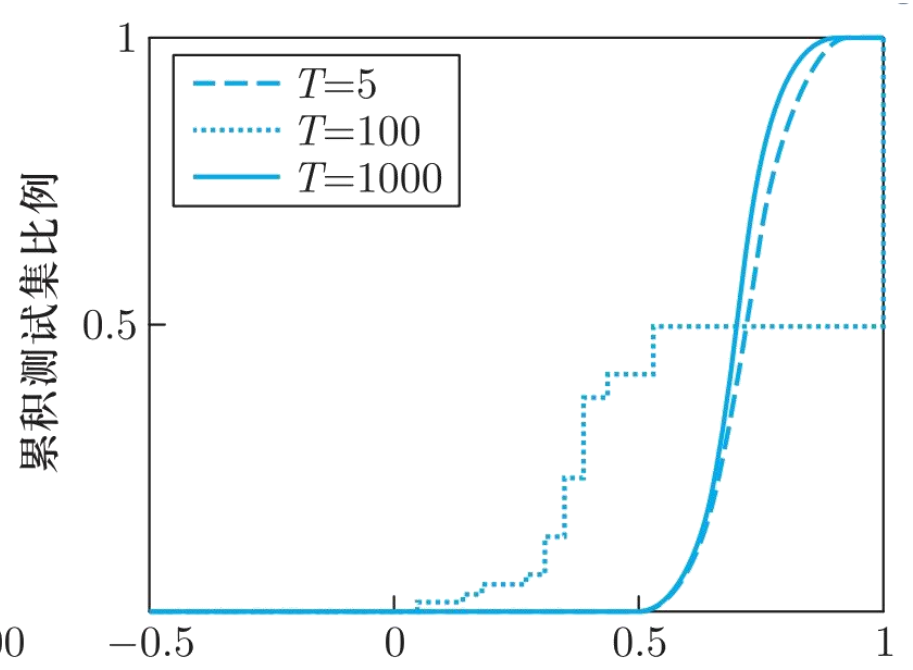
- 许多实验都表明 AdaBoost 算法在迭代次数很高时似乎并不容易出现过拟合. 特别地, 在训练误差率降到零以后, 继续增大学习轮数 T, AdaBoost 的测试误差率在某种程度上仍在降低. 例如, Schapire 绘制了效果曲线, 如图12.2(a) 所示. 对于 AdaBoost 算法, 在最初几轮训练误差率已降至零后的很长一段时间内, 测试误差率仍持续降低. 从表面上看, 随着更多子分类器的加入, 集成分类器形式趋于复杂, 分类精度却仍在提高, 似乎违背了科学研究中的“奥卡姆剃刀”原理. 因此, 如何解释 AdaBoost 为什么不容易过拟合成为了 AdaBoost 算法中最迷人的理论问题, 吸引了大量关注.

12.2.2 AdaBoost 算法的泛化误差

- 为了解释这一现象, 分析 AdaBoost 算法的强泛化能力, Schapire 将统计学习中分类间隔 (例如支持向量机中定义的间隔) 的相关分析理论引入 AdaBoost 算法[145]. 由此, Schapire et al. 指出, AdaBoost 之所以没有过拟合, 是因为在训练误差达到零后, 强分类器随着学习轮数增加, 其间隔还在增大, 因而泛化误差仍在减小. 这也成了目前最流行的分析方法.



(a) 训练误差与测试误差



(b) 间隔

12.2.2 AdaBoost 算法的泛化误差

- 根据泛化误差依赖于最小间隔 θ 的值. 我们可以最大化最小间隔来得到更紧的泛化误差. 这也正是 Breiman 设计 Arc-gv 算法的主要思想. 在每轮迭代中, Arc-gv 根据上述思想更新 α_t 形式如下:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1+\gamma_t}{1-\gamma_t}\right) - \frac{1}{2}\ln\left(\frac{1+\theta_t}{1-\theta_t}\right),$$

  - 其中 $\gamma_t = \frac{1}{2} - \mathrm{eer}_{\mathcal{D}_t}$, θ_t 是截止当前学习轮次的组合分类器的最小间隔.

- 与 AdaBoost 相比, Arc-gv 能够得到更大的最小分类间隔以及更好的分类间隔分布. 而且, 基于间隔理论, Breiman 给出了更紧的泛化误差界[146].

- 按照理论分析, Arc-gv 具有更强的泛化能力, Arc-gv 应当比 AdaBoost 性能好. 然而 Breiman 在实验中发现: 尽管 Arc-gv 能够得到比 AdaBoost 更大的最小间隔, 但 Arc-gv 的测试误差率却总是高于 AdaBoost. 因此, Breiman对 Schapire 的间隔分析理论提出了质疑, 使得 AdaBoost 的间隔分析方法受到了极大的挑战.

12.2.2 AdaBoost 算法的泛化误差

- 随后的几年, 虽然有诸多学者给出了 AdaBoost 更紧致的泛化误差界, 并指出其实是分类间隔分布影响着 AdaBoost 的泛化误差界. 但是关于 Breiman的质疑, 都没有给出正面回答. 七年后, Reyzin 和 Schapire 发现了一个有趣的现象. 考虑到泛化误差界与间隔 θ、学习轮数 T 以及子学习器的复杂度相关, 因此为了研究间隔的影响, 需要固定另外的两个因素. Breiman 在比较Arc-gv 和 AdaBoost 时, 在实验中以 CART 为弱分类器, 通过固定叶节点数目来控制弱分类器复杂度. 然而, Reyzin 和 Schapire 却进一步发现, 当叶节点数目相同时, Arc-gv 与 AdaBoost 生成的决策树有着很大的不同[147]. Arc-gv生成的树通常拥有较大的深度, 而 AdaBoost 生成的树则宽度较大. 一般情况下, 更深的决策树由于进行了更多分裂, 可能具有更大的模型复杂度. 在这种情况下比较 Arc-gv 与 AdaBoost 有失公平. Reyzin 和 Schapire 重新进行了Breiman 的实验, 但使用复杂度相同的决策树作为基分类器. 此时, AdaBoost比 Arc-gv 的间隔分布要优. 虽然 Reyzin 和 Schapire 指出分类间隔分布是影响算法泛化能力的关键, 但并没有给出比 Breiman 更紧致地泛化误差界.

- 为彻底解决此类问题, 需要通过分类间隔分布给出 AdaBoost 算法更紧致的泛化误差界. 而且, 要想正面回答 Breiman 的问题, 这个泛化误差界应该比 Breiman 给出的基于最小分类间隔的泛化误差界更紧致.

12.2.2 AdaBoost 算法的泛化误差

- 也就是说, 是分类间隔分布, 而不是最小分类间隔决定了算法的泛化能力. 2008 年, 王立威等构造了一种与分类间隔分布有关, 与最小分类间隔几乎无关的平衡分类间隔(equilibrium margin, Emargin), 给出了基于平衡分类间隔更紧致的 Emargin界[148]. 通过实验证明：与 Arc-gv 相比, AdaBoost 有着更大的平衡分类间隔以及更低的测试误差率, 实现了实验测试与理论分析结果的一致. 但是证明过程中考虑了比 Breiman 和 Schapire 模型更多的信息, 因而无法直接说明平衡分类间隔比最小间隔更本质.
- 为更进一步解决这个问题, 2013 年, 周志华进行了更深入的研究[149]. 首先, 他们引入了第 k 间隔界 (the kth margin bound), 并且研究其与最小间隔界与 Emargin 界的关系. 然后, 通过改进 Bernstein 界最终给出了更好的泛化误差界. 总之, 周志华他们的研究结果不仅正面回答了 Breiman 的疑问, 合理解释了 AdaBoost 不会过拟合的原因, 而且进一步巩固了间隔分析理论知识结构.

12.3　AdaBoost 算法原理探析

12.3 AdaBoost 算法原理探析

- AdaBoost 算法最初的方法融合了“Online”学习与加权投票的思想, 但随着研究的深入, 不同学者从不同的视域提出了各种新的理论模型, 越来越多地将 Boosting 算法设计与优化理论联系起来, 从不同的视角解释AdaBoost算法的原理和有效性. 这些不同的视域分析也为新算法的设计提供了更广阔的思路. 本章将分别从几种主流的视域对 AdaBoost 算法进行分析, 并对其算法的原理进行讨论.

12.3.1 损失函数最小化视域

- 多数统计与机器学习问题都可视为对目标函数或者损失函数的优化. 例如, 最简单的一元线性回归问题. 给定样本 $\{(\boldsymbol{X}_1, Y_1), \cdots, (\boldsymbol{X}_n, Y_n)\}$.最小二乘估计的目标是找到参数 $\beta = (\beta_1, \boldsymbol{\beta}_2)^{\mathrm{T}}$, 使其最小化残差平方和

$$\mathrm{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(Y_i - \beta_1 - \boldsymbol{X}_i^{\mathrm{T}} \beta_2 \right)^2,$$

  - 其中, RSS($\boldsymbol{\beta}$) 就是损失函数. 许多其他方法, 如神经网络、最大似然估计、支持向量机、逻辑斯谛回归等, 都可视为某种程度下的目标函数优化问题.

- 现在的问题是：AdaBoost 是否也可视为是最优化目标函数的过程？后面的研究表明, AdaBoost 确实可视为优化某种损失函数的一种算法.

1. 指数损失函数

- 那么 AdaBoost 关联的损失函数如何定义呢？我们前面得到分类器的误差率为

$$\frac{1}{n} \sum_{i=1}^{n} I\left(\mathcal{H}\left(\boldsymbol{X}_i \right) \neq Y_i \right), \tag{12.3.1}$$

12.3.1 损失函数最小化视域

- 那么 AdaBoost 算法就是最小化 (12.3.1) 所示的目标函数吗？其实并不是, 而是最小化式 (12.3.1) 的一个上界. 因为 $\mathcal{H}(\boldsymbol{X}) = \text{sign}(\tilde{h}_T(\boldsymbol{X}))$, 则

$$\frac{1}{n}\sum_{i=1}^{n} I\left(\mathcal{H}\left(\boldsymbol{X}_i\right) \neq Y_i\right) = \frac{1}{n}\sum_{i=1}^{n} I\left(\text{sign}\left(\tilde{h}_T\left(\boldsymbol{X}_i\right)\right) \neq Y_i\right) = \frac{1}{n}\sum_{i=1}^{n} I\left(\tilde{h}_T\left(\boldsymbol{X}_i\right) Y_i \leq 0\right). \quad (12.3.2)$$

- 利用 $I(x \leqslant 0) \leqslant \mathrm{e}^{-x}$, 则知误差率上界可取为

$$\mathscr{L}_{\exp}\left(\tilde{h} \middle| \mathcal{D}\right) = E_{\boldsymbol{X}\sim\mathcal{D}}\left(\mathrm{e}^{-Y\tilde{h}_T(\boldsymbol{X})}\right), \quad (12.3.3)$$

- 其中, $Y\,\tilde{h}_T(\boldsymbol{X})$ 称为假定问题的分类间隔.

- 要使上述损失函数达到最小, 关键要解决以下两个问题:
  - (1) 如何确定一系列的弱学习器 h_t?
  - (2) 如何确定合适的权重 α_t?

- 利用式 (12.1.6) 可知, 只需求

$$\mathscr{L}_{\exp}\left(\alpha_t h_t \middle| \mathcal{D}_t\right) = E_{\boldsymbol{X}\sim\mathcal{D}_t}\left(\mathrm{e}^{-Y\alpha_t h_t}\right) \quad (12.3.4)$$

12.3.1 损失函数最小化视域

- 在分布 $\mathcal{D}_t$ 下关于权重 α_t 和学习器 h_t 的最小值.

$$\begin{aligned}
&E_{\boldsymbol{X}\sim\mathcal{D}_t}\left(\mathrm{e}^{-Y\alpha_t h_t}\right)\\
&=E_{\boldsymbol{X}\sim\mathcal{D}_t}\left[\mathrm{e}^{-Y\alpha_t}I\left(Y=h_t\left(\boldsymbol{X}\right)\right)+\mathrm{e}^{\alpha_t}I\left(Y\neq h_t\left(\boldsymbol{X}\right)\right)\right]\\
&=\mathrm{e}^{-\alpha_t}P_{\boldsymbol{X}\sim\mathcal{D}_t}\left(Y=h_t\left(\boldsymbol{X}\right)\right)+\mathrm{e}^{\alpha_t}P_{\boldsymbol{X}\sim\mathcal{D}_t}\left(Y\neq h_t\left(\boldsymbol{X}\right)\right)\\
&=\mathrm{e}^{-\alpha_t}\left(1-\mathrm{err}_{\mathcal{D}_t}\right)+\mathrm{e}^{\alpha_t}\mathrm{err}_{\mathcal{D}_t},
\end{aligned}\tag{12.3.5}$$

- 其中 $\mathrm{err}_{\mathcal{D}_t}=P_{\boldsymbol{X}\sim\mathcal{D}_t}\left(Y\neq h_t\left(\boldsymbol{X}\right)\right)$. 为得到最优 α_t, 将指数损失函数求导令其为零, 则

$$\frac{\partial\mathscr{L}_{\exp}\left(\alpha_t h_t\left|\mathcal{D}_t\right.\right)}{\partial\alpha_t}=-\mathrm{e}^{-\alpha_t}\left(1-\mathrm{err}_{\mathcal{D}_t}\right)+\mathrm{e}^{\alpha_t}\mathrm{err}_{\mathcal{D}_t}=0,\tag{12.3.6}$$

- 解得

$$\alpha_t=\frac{1}{2}\ln\left(\frac{1-\mathrm{err}_{\mathcal{D}_t}}{\mathrm{err}_{\mathcal{D}_t}}\right).\tag{12.3.7}$$

12.3.1 损失函数最小化视域

- 这正是 AdaBoost 算法中计算 α_t 的方法. 下面讨论 h_t 的选取.
- 假设已获得了一系列弱学习器并知其权重, 则组合形成 $\tilde{h}_{t-1}$. 关于 h_t 的理想选择就是纠正 $\tilde{h}_{t-1}$ 的错误, 最小化指数损失函数

$$\begin{aligned}
&\mathscr{L}_{\exp}\left(\tilde{h}_{t-1}+h_t \,|\mathcal{D}\right) \\
&=E_{\boldsymbol{X}\sim\mathcal{D}}\left[\mathrm{e}^{-Y\left(\tilde{h}_{t-1}(\boldsymbol{X})+h_t(\boldsymbol{X})\right)}\right] \\
&=E_{\boldsymbol{X}\sim\mathcal{D}}\left[\mathrm{e}^{-Y\tilde{h}_{t-1}(\boldsymbol{X})}\cdot\mathrm{e}^{-Yh_t(\boldsymbol{X})}\right].
\end{aligned} \tag{12.3.8}$$

  - 对 $\mathrm{e}^{-Yh_t(\boldsymbol{X})}$ 利用泰勒展式, 则

$$\begin{aligned}
&\mathscr{L}_{\exp}\left(\tilde{h}_{t-1}+h_t \,|\mathcal{D}\right) \\
&\approx E_{\boldsymbol{X}\sim\mathcal{D}}\left[\mathrm{e}^{-Y\tilde{h}_{t-1}(\boldsymbol{X})}\left(1-Yh_t(\boldsymbol{X})+\frac{Y^2h_t(\boldsymbol{X})^2}{2}\right)\right] \\
&=E_{\boldsymbol{X}\sim\mathcal{D}}\left[\mathrm{e}^{-Yh_{t-1}(\boldsymbol{X})}\left(1-Yh_t(\boldsymbol{X})+\frac{1}{2}\right)\right].
\end{aligned} \tag{12.3.9}$$

12.3.1 损失函数最小化视域

- 因此, h_t 的理想选择为

$$
\begin{aligned}
h_t(X) &= \arg\min_h \mathscr{L}_{\exp}\left(\tilde{h}_{t-1} + h_t \,|\, \mathcal{D}\right) \\
&= \arg\min_h E_{X\sim\mathcal{D}}\left[\mathrm{e}^{-Y\tilde{h}_{t-1}(X)}\left(1 - Yh_t(X) + \frac{1}{2}\right)\right] \\
&= \arg\max_h E_{X\sim\mathcal{D}}\left[\mathrm{e}^{-Y\tilde{h}_{t-1}(X)} Yh_t(X)\right] \\
&= \arg\max_h E_{X\sim\mathcal{D}}\left[\frac{\mathrm{e}^{-Y\tilde{h}_{t-1}(X)}}{E_{X\sim\mathcal{D}}\left[\mathrm{e}^{-Y\tilde{h}_{t-1}(X)}\right]} Yh_t(X)\right].
\end{aligned}
\tag{12.3.10}
$$

- 记

$$
\mathcal{D}_t(X) = \frac{\mathcal{D}(X)\mathrm{e}^{-Y\tilde{h}_{t-1}(X)}}{E_{X\sim\mathcal{D}}\left[\mathrm{e}^{-Y\tilde{h}_{t-1}(X)}\right]}, \tag{12.3.11}
$$

12.3.1 损失函数最小化视域

- 则利用期望的定义, 可得

$$h_t(\boldsymbol{X}) = \arg\max_h E_{\boldsymbol{X}\sim\mathcal{D}}\left[\frac{\mathrm{e}^{-Y\tilde{h}_{t-1}(\boldsymbol{X})}}{E_{\boldsymbol{X}\sim\mathcal{D}}\left[\mathrm{e}^{-Y\tilde{h}_{t-1}(\boldsymbol{X})}\right]} Yh_t(\boldsymbol{X})\right] \tag{12.3.12}$$
$$= \arg\max_h E_{\boldsymbol{X}\sim\mathcal{D}_t}\left[Yh_t(\boldsymbol{X})\right].$$

- 又由于 $Y\,h_t(\boldsymbol{X}) = 1 - 2I(Y \neq h_t(\boldsymbol{X}))$, 因而 $h_t(\boldsymbol{X})$ 满足

$$h_t(\boldsymbol{X}) = \arg\min_h E_{\boldsymbol{X}\sim\mathcal{D}_t}\left[I\left(Y \neq h_t(\boldsymbol{X})\right)\right]. \tag{12.3.13}$$

- 通过式 (12.3.11), 可得

$$\mathcal{D}_{t+1}(\boldsymbol{X}) = \frac{\mathcal{D}(\boldsymbol{X})\mathrm{e}^{-Y\tilde{h}_t(\boldsymbol{X})}}{E_{\boldsymbol{X}\sim\mathcal{D}}\left[\mathrm{e}^{-Y\tilde{h}_t(\boldsymbol{X})}\right]} = \frac{\mathcal{D}(\boldsymbol{X})\mathrm{e}^{-Y\tilde{h}_{t-1}(\boldsymbol{X})}\cdot\mathrm{e}^{-Y\alpha_t h_t(\boldsymbol{X})}}{E_{\boldsymbol{X}\sim\mathcal{D}}\left[\mathrm{e}^{-Y\tilde{h}_t(\boldsymbol{X})}\right]} = \mathcal{D}_t(\boldsymbol{X})\cdot\mathrm{e}^{-Y\alpha_t h_t(\boldsymbol{X})}\frac{E_{\boldsymbol{X}\sim\mathcal{D}}\left[\mathrm{e}^{-Y\tilde{h}_{t-1}(\boldsymbol{X})}\right]}{E_{\boldsymbol{X}\sim\mathcal{D}}\left[\mathrm{e}^{-Y\tilde{h}_t(\boldsymbol{X})}\right]}. \tag{12.3.14}$$

12.3.1 损失函数最小化视域

- 这恰好是 AdaBoost 算法更新分布的方法.

注 12.4 从优化损失函数的角度分析 AdaBoost 算法, 有以下优点:

- (1) 帮助我们明确 AdaBoost 算法的学习目标, 有助于理解 AdaBoost 的理论原理及相关性质.
- (2) 实现学习目标 (损失函数最小化) 与实现目标 (数值方法) 之间的“解耦”(decoupling). 一方面可以修改目标函数以适应新的学习模型, 另一方面可以引入快速实用的数值方法. 从而可以建立其他类型的 AdaBoost 算法[150]. 如优化任意可导函数的 AnyBoost; 优化基于分类间隔损失函数的MarginBoost 等[151].

2. 逻辑斯谛损失函数

- 对于 AdaBoost 算法, 如注 12.4 所言, 可以根据需要使用不同的损失函数.
- 对于指数损失函数 $\mathrm{e}^{-Y\tilde{h}_T(X)}$, 其优点是性质好且容易处理. 但若预测错误, 则 $Y\tilde{h}_T(\boldsymbol{X})$ 为负, 从而 $\mathrm{e}^{-Y\tilde{h}_T(X)}$为指数增长. 这意味着此时对扰动很敏感, 即稳健性较差.

12.3.1 损失函数最小化视域

- 为此, 在当代 Boosting 方法中, 常用逻辑斯谛损失函数:

$$\mathscr{L}_{\log}\left(\tilde{h}\middle|\mathcal{D}\right)=E_{\boldsymbol{X}\sim\mathcal{D}}\left[\ln\left(1+\mathrm{e}^{-Y\tilde{h}_T(\boldsymbol{X})}\right)\right]. \tag{12.3.15}$$

  - 显然根据指数函数与对数函数的性质知式 (12.3.3) 与式 (12.3.15) 会被相同的函数最小化. 事实上, 可以通过极大似然的思想来解释 $\ln\left(1+\mathrm{e}^{-Y\tilde{h}_T(\boldsymbol{X})}\right)$ 的来源. 已知 $Y_i\in\{-1,1\}$, 且服从 Logit 模型, 则有:

$$P\left(Y_i\middle|\boldsymbol{X}_i\right)=\begin{cases}\dfrac{1}{1+\mathrm{e}^{-\tilde{h}_T(\boldsymbol{X}_i)}}, & Y_i=1,\\ \dfrac{1}{1+\mathrm{e}^{\tilde{h}_T(\boldsymbol{X}_i)}}, & Y_i=-1.\end{cases}$$

  - 考虑到 Y_i 的取值为 -1 或者 1, 将上式合并成

$$P\left(Y_i\middle|\boldsymbol{X}_i\right)=\begin{cases}\dfrac{1}{1+\mathrm{e}^{-Y_i\tilde{h}_T(\boldsymbol{X}_i)}}, & Y_i=1,\\ \dfrac{1}{1+\mathrm{e}^{-Y_i\tilde{h}_T(\boldsymbol{X}_i)}}, & Y_i=-1.\end{cases}$$

12.3.1 损失函数最小化视域

- 即条件概率的表达式统一为

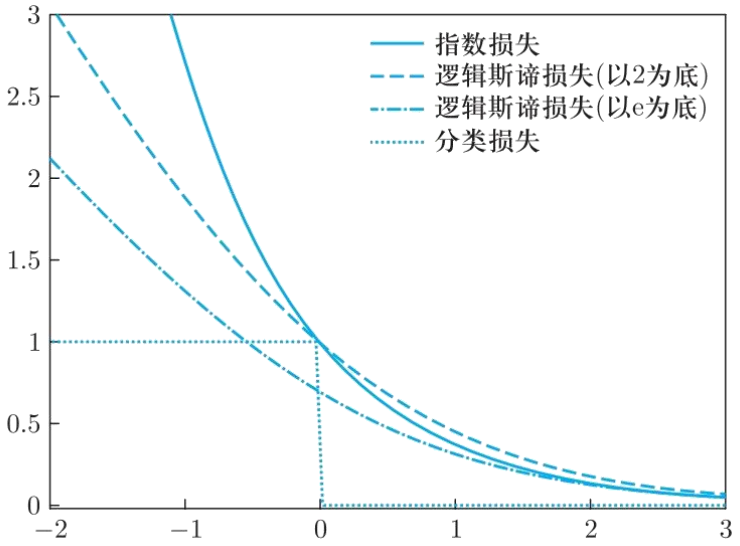
$$P\left(Y_i \middle| \boldsymbol{X}_i\right) = \frac{1}{1+\mathrm{e}^{-Y_i \tilde{h}_T(\boldsymbol{X}_i)}}, \tag{12.3.15}$$

- 从而最大似然函数为

$$\prod_{i=1}^{n} P\left(Y_i \middle| \boldsymbol{X}_i\right) = \prod_{i=1}^{n} \frac{1}{1+\mathrm{e}^{-Y_i \tilde{h}_T(\boldsymbol{X}_i)}} = \prod_{i=1}^{n} \left(1+\mathrm{e}^{-Y_i \tilde{h}_T(\boldsymbol{X}_i)}\right)^{-1}. \tag{12.3.16}$$

- 上式 (12.3.16) 的最大值等价成取负对数的最小值, 即式 (12.3.15).

注 12.5 指数损失函数与逻辑斯谛损失函数都可视作对 0–1 损失函数的光滑近似. 相比较而言, 逻辑斯谛的近似程度要更好. 见图 12.3.



12.3.1 损失函数最小化视域

- Friedman et al. 认为, 与其用 AdaBoost 中的拟牛顿更新策略, 不如通过梯度下降的方式优化逻辑斯谛损失函数, 并据此给出了 LogitBoost 算法[141].

LogitBoost 算法

输入： 训练数据集 $\{(\boldsymbol{X}_1, Y_1), (\boldsymbol{X}_2, Y_2), \cdots, (\boldsymbol{X}_n, Y_n)\}$;
基学习算法 $\mathcal{L}$;
学习轮数 T.

过程：

1. $Y_0(\boldsymbol{X}_i) = Y_i$; % 初始化目标
2. $\widetilde{h}_0(\boldsymbol{X}) = 0$; % 初始化函数
3. **for** $t = 1, \cdots, T$:
4. $p_t(\boldsymbol{X}) = \dfrac{1}{1 + \mathrm{e}^{-2\widetilde{h}_{t-1}(\boldsymbol{X})}}$; % 计算概率
5. $Y_t(\boldsymbol{X}) = \dfrac{Y_{t-1}(\boldsymbol{X}) - p_t(\boldsymbol{X})}{p_t(\boldsymbol{X})(1 - p_t(\boldsymbol{X}))}$; % 更新目标
6. $\mathcal{D}_t(\boldsymbol{X}) = p_t(\boldsymbol{X})(1 - p_t(\boldsymbol{X}))$; % 更新权重
7. $h_t = \mathcal{L}(\mathcal{D}, Y_t, \mathcal{D}_t)$; % 训练分类器 h_t 在分布 $\mathcal{D}_t$ 下拟合样本集中的 $Y_t()$
8. $\widetilde{h}_t = \widetilde{h}_{t-1} + \dfrac{1}{2} h_t$; % 更新组合分类器
9. **end**

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

12.3.2 向前逐段可加视域

1. 向前逐段算法

- 对于二元分类问题的 AdaBoost 算法, 其最终表达式为

$$\mathcal{H}(\boldsymbol{X}) = \operatorname{sign}\left(\tilde{h}_T(\boldsymbol{X})\right) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{X})\right).$$

  - 可将 $h_t(\boldsymbol{X}) \in \{-1, 1\}$ 视为基函数, 类似于泰勒展式的函数项. 不失一般性,将希望学到的函数 $f(x)$ 做基函数展开, 可以得到加法模型:

$$f(x) = \sum_{t=1}^{T} \alpha_t h(x; \gamma_t), \tag{12.3.17}$$

  - 其中, α_t 为展开系数, $h(x; \gamma_t)$ 为基函数, γ_t 为参数.

- 为估计展开式系数 α_t 与参数 γ_t, 可以最小化以下目标函数:

$$\min_{\alpha_t's, \gamma_t's} \sum_{i=1}^{n} L\left(Y_i, \sum_{t=1}^{T} \alpha_t h(\boldsymbol{X}_i; \gamma_t)\right), \tag{12.3.18}$$

12.3.2 向前逐段可加视域

▶ 其中, $L(Y_i, f(\boldsymbol{X}_i))$ 为损失函数. 例如$L(Y_i, f(\boldsymbol{X}_i)) = (Y_i - f(\boldsymbol{X}_i))^2$, $\mathrm{L}(Y_i f(\boldsymbol{X}_i)) = |Y_i - f(\boldsymbol{X}_i)|$ 或者 0−1 损失函数 $I(Y_i \neq f(\boldsymbol{X}_i))$.

- 显然, 这是一个复杂的优化问题. 向前分段算法 (forward stagewise algorithm) 求解这一优化问题的想法是: 利用加性模型的性质, 考虑从向前的分步计算, 每一步只学习一个基函数及其系数, 逐步逼近优化目标函数式(12.3.18). 具体地, 每步只需优化如下损失函数:

$$\min_{\alpha,\gamma} \sum_{i=1}^{n} L\left(Y_i, \sum_{t=1}^{T} \alpha h\left(\boldsymbol{X}_i; \gamma_t\right)\right), \tag{12.3.19}$$

2. 向前分段算法与 AdaBoost 算法

- 由向前分段算法可以推导出 AdaBoost, 用如下定理叙述这一关系.

- **定理 12.4** 二元分类问题的 AdaBoost 算法是向前分段加法算法的特例. 其中模型是由基本分类器组成的加法模型, 损失函数为

$$L\left(Y, f\left(\boldsymbol{X}\right)\right) = \mathrm{e}^{-Yf(\boldsymbol{X})}.$$

12.3.2 向前逐段可加视域

- 该定理的证明关键是找出 $h_t(\boldsymbol{X})$ 和 α_t, 其证明过程类似于前面的推导过程, 可参见文献 [19].

12.4　Boosting 算法的演化

12.4 Boosting 算法的演化

- 基于 AdaBoost 的理论剖析, 人们对 AdaBoost 算法进行了各种改进与推广, 并进行了广泛应用.

12.4.1 回归问题的 Boosting 算法

- 最初的 AdaBoost 算法是针对分类问题设计的, 但其算法思想极其具有一般性, 因而可将其应用于其他问题, 下而重点介绍回归问题的 LSBoost 方法.
- 对于回归问题, 一般采用均方误差为损失函数:

$$\mathscr{L}_{\text{mse}} = \left(Y - \tilde{h}_T\left(\boldsymbol{X}\right)\right)^2, \tag{12.4.1}$$

  - 将 12.3.2 节中的向前逐段加法模型代入该损失函数如下, 并且最小化该损失函数可得 α_t,γ_t:

$$\begin{aligned}
&\min_{\alpha,\gamma}\sum_{i=1}^{n}\mathscr{L}_{\text{mse}}\left(Y_i,\tilde{h}_{t-1}\left(\boldsymbol{X}_i\right)+\alpha\cdot h\left(\boldsymbol{X}_i;\gamma\right)\right)\\
&=\min_{\alpha,\gamma}\sum_{i=1}^{n}\left(Y_i-\tilde{h}_{t-1}\left(\boldsymbol{X}_i\right)-\alpha\cdot h\left(\boldsymbol{X}_i;\gamma\right)\right)^2\\
&=\min_{\alpha,\gamma}\sum_{i=1}^{n}\left(r_{ti}-\alpha\cdot h\left(\boldsymbol{X}_i;\gamma\right)\right)^2
\end{aligned} \tag{12.4.2}$$

12.4.1 回归问题的 Boosting 算法

- 其中, r_{ti} 为当前阶段模型的残差. 因而算法的更新思想是以当前残差 r_{ti} 为因变量, 对自变量 $h(\boldsymbol{X}_i; \gamma)$ 进行回归即可. 进行回归时, 可以采用普通的线性回归方法, 也可以使用如下所述的回归树.

1. 树模型

- 分类与回归树模型 (classification and regression tree, CART) 主要由Quinlan、 Breiman et al. 创立[67] [131], 是一种基本的分类与回归方法. 用于分类问题时, 称为分类树 (classification tree); 用于回归问题, 称为回归树 (regression tree). 这些是前面介绍过的内容.

- 对于分类问题, 我们回忆例 12.1 的做法. 基本分类器可以看作是由一个根节点连接两个叶节点的简单决策树, 常称为决策树桩 (decision stump). 最终的强分类器形式为 $\tilde{h}_T(x)=\sum_{t=1}^{T} h_t(x)$ 抽象成一般形式应用于回归问题,则回归 Boosting 树模型可以表示以决策树为基函数的加法模型:

$$\tilde{h}_T(\boldsymbol{X})=\sum_{t=1}^{T} G(\boldsymbol{X};\Theta_t), \tag{12.4.3}$$

- 其中, $G(\boldsymbol{X}; \Theta_t)$ 表示决策树, Θ_t 为决策树的参数, T 为树的数量.

12.4.1 回归问题的 Boosting 算法

2. 回归问题的 Boosting 树算法

- 假设 $\mathcal{X}$ 为输入空间, $\mathcal{Y}$ 为输出空间, $\{(\boldsymbol{X}_1, Y_1),(\boldsymbol{X}_2, Y_2), \cdots ,(\boldsymbol{X}_n, Y_n)\}$ 为训练数据集.
- 一棵回归树由输入空间的一个划分以及在划分单元上的取值两部分决定. 假设已将输入空间 $\mathcal{X}$ 划分为 K 个单元: $R_1, \cdots , R_K$, 并且在每个单元 R_k 上的取值为 c_k, 于是回归树具有如下形式:

$$G\left(\boldsymbol{X};\Theta_t\right)=\sum_{k=1}^{K} c_k I\left(\boldsymbol{X}\in R_k\right), \tag{12.4.4}$$

  - 其中, 参数 $\Theta=\{(R_1, c_1),(R_2, c_2), \cdots ,(R_K, c_K)\}$ 表示树的区域划分和各区域上的常数. K 是回归树的复杂度, 即叶节点个数.
- 回归问题的 Boosting 树算法采用向前分段算法. 首先确定初始 Boosting树 $\tilde{h}_0\left(\boldsymbol{X}\right)$, 设 $\tilde{h}_{t-1}\left(\boldsymbol{X}\right)$ 为当前模型, 则第 t 步的模型是

$$\tilde{h}_t\left(\boldsymbol{X}\right)=\tilde{h}_{t-1}\left(\boldsymbol{X}\right)+G\left(\boldsymbol{X};\Theta_t\right),$$

12.4.1 回归问题的 Boosting 算法

- 其中 Θ_t 的选取遵循损失最小化原则, 即

$$\Theta_t = \arg\min_{\Theta} \sum_{i=1}^{n} \mathscr{L}\left(Y_i, \tilde{h}_{t-1}\left(\boldsymbol{X}_i\right) + G\left(\boldsymbol{X}_i; \Theta\right)\right)$$

- 由于树的线性组合可以很好地拟合训练数据, 所以回归问题的 Boosting 树是一个很高效的学习方法. 此算法代码如下:

回归问题的 Boosting 树算法

输入: 训练数据集 $\{(\boldsymbol{X}_1, Y_1), (\boldsymbol{X}_2, Y_2), \cdots, (\boldsymbol{X}_n, Y_n)\}$;
损失函数 $\mathscr{L}_{\text{mse}}(Y, \tilde{h}_T(\boldsymbol{X}))$, 基函数集 $\{G(\boldsymbol{X}; \Theta_t)\}$;
学习轮数 T.

过程:

1. $\tilde{h}_0(\boldsymbol{X}) \equiv 0$; % 初始化基函数
2. **for** $t = 1, \cdots, T$:
3. $\Theta_t = \underset{\Theta}{\text{argmin}} \sum_{i=1}^{n} \mathscr{L}_{\text{mse}}(Y_i, \tilde{h}_{t-1}(\boldsymbol{X}_i) + G(\boldsymbol{X}_i; \Theta))$; % 拟合残差得到回归树
4. $\tilde{h}_t(\boldsymbol{X}) = \tilde{h}_{t-1}(\boldsymbol{X}) + G(\boldsymbol{X}; \Theta_t)$; % 更新加法模型
5. **end**

输出: $\tilde{h}_T(\boldsymbol{X}) = \sum_{t=1}^{T} G(\boldsymbol{X}; \Theta_t)$.

12.4.1 回归问题的 Boosting 算法

3. 回归问题其他损失函数

- 对于回归问题的损失函数, 可以根据需要取成不同的形式.
  - (1) 拉普拉斯 (Laplace) 损失函数:

$$\mathscr{L}_{ll}\left(Y,\tilde{h}_T\left(\boldsymbol{X}\right)\right)=\left|Y-\tilde{h}_T\left(\boldsymbol{X}\right)\right|.$$

  - 相比较于平方损失函数 $\mathscr{L}_{\text{mse}}$, 拉普拉斯损失函数不易受到异常值的影响, 因而更稳健.
  - (2) 胡伯 (Huber) 损失函数:

$$\mathscr{L}_{ll}\left(Y,\tilde{h}_T\left(\boldsymbol{X}\right)\right)=\begin{cases}\dfrac{1}{2}\left[Y-\tilde{h}_T\left(\boldsymbol{X}\right)\right]^2, & \left|Y-\tilde{h}_T\left(\boldsymbol{X}\right)\right|\le\delta,\\ \delta\left[\left|Y-\tilde{h}_T\left(\boldsymbol{X}\right)\right|-\dfrac{\delta}{2}\right], & \left|Y-\tilde{h}_T\left(\boldsymbol{X}\right)\right|>\delta.\end{cases}$$

  - 其中, δ 为松弛参数. 通过以上表达式可以看出, 当误差绝对值较小时, 使用平方损失函数; 若误差绝对值较大, 则使用绝对值损失函数. 因此, 胡伯损失函数结合了平方损失函数与绝对值损失函数的优点.

12.4.2 梯度 Boosting 方法

1. 梯度 Boosting 算法

- 根据前面的理论剖析, AdaBoost 可视为加性模型中利用坐标下降法优化指数型损失函数的过程. 在此基础上, Friedman 提出了梯度 Boosting 算法(gradient boosting), 简称为 GBM 算法[152]. GBM 思想：以非参数方法估计基函数, 并在“函数空间”使用梯度下降法求解.
- 假设训练数据 $\mathcal{D}=\{(\boldsymbol{X}_1, Y_1), \cdots, (\boldsymbol{X}_n, Y_n)\}$, 若以函数 $\tilde{h}_T(\boldsymbol{X})$来预测 Y, 则在总体中的平均损失 (expected loss function) 为

$$E_{Y,\boldsymbol{X}}\mathscr{L}\left[Y, \tilde{h}_T(\boldsymbol{X})\right],$$

  - 其中 $\mathscr{L}[\cdot,\cdot]$ 为损失函数. 目标变成找出能够最小化损失的函数 $\tilde{h}^*(\boldsymbol{X})$, 即

$$\tilde{h}^*(\boldsymbol{X}) = \arg\min_{\tilde{h}} E_{Y,\boldsymbol{X}}\mathscr{L}\left[Y, \tilde{h}_T(\boldsymbol{X})\right].$$

12.4.2 梯度 Boosting 方法

- 利用期望的性质将上式写成

$$\tilde{h}^*(\boldsymbol{X}) = \arg\min_{\tilde{h}} E_{Y,\boldsymbol{X}} \mathscr{L}\left[Y, \tilde{h}_T(\boldsymbol{X})\right]$$

$$= \arg\min_{\tilde{h}} E_{\boldsymbol{X}}\left(E_Y\left(\mathscr{L}\left(Y, \tilde{h}_T(\boldsymbol{X})\right)\middle|\boldsymbol{X}\right)\right),$$

- 因而最小化问题等价为

$$\min_{\tilde{h}} E_Y\left[\mathscr{L}\left(Y, \tilde{h}_T(\boldsymbol{X})\right)\middle|\boldsymbol{X}\right]. \tag{12.4.5}$$

- 使用非参数思想, 将 $\tilde{h}_T(x)$ 在每个 $\boldsymbol{X}$ 的取值均看作参数. 由此, $\tilde{h}_T(x)$ 可视为无穷维向量, 故有无穷多参数. 在函数空间上, 进行“泛函梯度下降”方法找最优解.
- 假设模型为加法模型:

$$\tilde{h}^*(\boldsymbol{X}) = \sum_{t=1}^{T} f_t(\boldsymbol{X}),$$

12.4.2 梯度 Boosting 方法

- 利用梯度下降算法, 则

$$f_t(\boldsymbol{X}) = -\rho_t g_t(\boldsymbol{X}), \tag{12.4.6}$$

- 其中 ρ_t 为步幅, 也称为学习率 (learning rate), 且

$$g_t(\boldsymbol{X}) = \left[\frac{\partial E_Y\left(\mathscr{L}\left(Y, \tilde{h}_T(\boldsymbol{X})\right)|\boldsymbol{X}\right)}{\partial \tilde{h}_T(\boldsymbol{X})}\right]_{\tilde{h}(\boldsymbol{X}_i)=\tilde{h}_{t-1}(\boldsymbol{X}_i)}, \tag{12.4.7}$$

$$\tilde{h}_{t-1}(\boldsymbol{X}_i) = \sum_{j=1}^{t-1} f_i(\boldsymbol{X}).$$

- 假设泛函正则性足够好, 能够交换微积分次序, 则

$$g_t(\boldsymbol{X}) = \left[\frac{\partial \mathscr{L}\left(Y, \tilde{h}_T(\boldsymbol{X})\right)}{\partial \tilde{h}_T(\boldsymbol{X})}|\boldsymbol{X}\right]_{\tilde{h}(\boldsymbol{X}_i)=\tilde{h}_{t-1}(\boldsymbol{X}_i)}. \tag{12.4.8}$$

12.4.2 梯度 Boosting 方法

- ρ_t 通过下式给出

$$\rho_t = \arg\min_{\rho} E_{Y,X}\mathscr{L}\left(Y, \tilde{h}_{t-1}(\boldsymbol{X}) - \rho g_t(\boldsymbol{X})\right). \tag{12.4.9}$$

- 因为训练数据的有限性, 这种非参数统计的方法在实际应用中是行不通的. 为解决此问题, 首先假设

$$\tilde{h}_T(\boldsymbol{X}; \boldsymbol{\alpha}, \boldsymbol{\gamma}) = \sum_{t=1}^{T} \alpha_t h(\boldsymbol{X}; \gamma_t), \tag{12.4.10}$$

- 其中 $h(\boldsymbol{X}; \gamma_t)$ 由基学习器决定. 根据训练数据, 则其梯度方向为

$$g_t(\boldsymbol{X}_i) = \left[\frac{\partial \mathscr{L}\left(Y_i, \tilde{h}_T(\boldsymbol{X}_i)\right)}{\partial \tilde{h}_T(\boldsymbol{X}_i)}\right]_{\tilde{h}(\boldsymbol{X}_i)=\tilde{h}_{t-1}(\boldsymbol{X}_i)}, \quad i = 1, 2, \cdots, n, \tag{12.4.11}$$

- 因而, 选择与负梯度方向最为接近的弱学习器, 即

$$\gamma_t = \arg\min_{\gamma} \sum_{i=1}^{n} \left[-g_t(\boldsymbol{X}_i) - h(\boldsymbol{X}_i; \gamma)\right]^2. \tag{12.4.12}$$

12.4.2 梯度 Boosting 方法

- 求得最优 $h(\boldsymbol{X}_i; \gamma_t)$ 之后, 函数更新为

$$\tilde{h}_t = \tilde{h}_{t-1} + \alpha_t h\left(\boldsymbol{X}; \gamma_t\right), \tag{12.4.13}$$

  - 其中, 步幅 ρ_t 利用直线搜索 (line search) 取为

$$\alpha_t = \arg\min_{\alpha} \sum_{i=1}^{n} \mathscr{L}\left(Y_i, \tilde{h}_{t-1}\left(\boldsymbol{X}_i\right) + \alpha \cdot h\left(\boldsymbol{X}_i; \gamma_t\right)\right). \tag{12.4.14}$$

12.4.2 梯度 Boosting 方法

2. GBDT 方法

- 以 GBM 算法为基础, 结合树的思想给出梯度提升决策树 (gradient boost_x0002_ing decision tree, GBDT) 方法. GBDT 以决策树为弱学习器的学习模型, 利用梯度 Boosting 的方法, 完成机器学习任务. 直接给出算法如下:

GBDT 算法

输入：训练数据集 $D=\{(\boldsymbol{X}_1,Y_1),(\boldsymbol{X}_2,Y_2),\cdots,(\boldsymbol{X}_n,Y_n)\}$;

损失函数 $\mathscr{L}(\cdot,\cdot)$;

学习轮数 T.

过程：

1. $\widetilde{h}_0(\boldsymbol{X})=\underset{c}{\operatorname{argmin}}\sum_{i=1}^{n}\mathscr{L}(Y_i,c)$; % 构造初始学习器
2. **for** $t=1,2,\cdots,T$:
3. $r_{ti}=-\left[\dfrac{\partial\mathscr{L}(Y_i,\widetilde{h}_T(\boldsymbol{X}_i))}{\partial\widetilde{h}_T(\boldsymbol{X}_i)}\right]_{\widetilde{h}(\boldsymbol{X}_i)=\widetilde{h}_{t-1}(\boldsymbol{X}_i)}$; % 计算样本负梯度
4. 利用 $D_t=\{(\boldsymbol{X}_1,r_{t1}),\cdots,(\boldsymbol{X}_n,r_{tn})\}$ % 学习得到新的叶节点 R_{tj}
5. $c_{tj}=\underset{c}{\operatorname{argmin}}\sum_{i\in R_{tj}}\mathscr{L}(Y_i,\widetilde{h}_{t-1}(\boldsymbol{X}_i)+c)$,

   R_{tj} 表示第 t 棵树的第 j 个叶节点区域, $j=1,2,\cdots,J$. % 生成决策树
6. $\widetilde{h}_t(\boldsymbol{X})=\widetilde{h}_{t-1}(\boldsymbol{X})+\sum_{j=1}^{J}c_{tj}I(\boldsymbol{X}\in R_{tj})$; % 更新决策树
7. **end**

输出：$\widetilde{h}_T(\boldsymbol{X})=\sum_{t=1}^{T}\sum_{j=1}^{J}c_{tj}I(\boldsymbol{X}\in R_{tj})$.

12.4.2 梯度 Boosting 方法

注 12.6 对上述 GBDT 算法做一些说明:

- (1) 算法第 1 步构造初始学习器, 即只有一个根节点的初始决策树.
- (2) 对于决策树模型, 损失函数的常用方式有均方误差损失函数、拉普拉斯损失函数、胡伯损失函数等.
- (3) 算法第 3 步计算出 rti, 将其作为残差估计, 称其为拟残差 (pseudo residuals).
- (4) 算法第 4 步建立新的训练样本集 D_t:

$$D_t = \left\{\left(\boldsymbol{X}_1, r_{t1}\right), \cdots, \left(\boldsymbol{X}_n, r_{tn}\right)\right\}, \tag{12.4.15}$$

  并用 D_t 作为训练样本集构造一棵决策树, 取此决策树为第 $t+1$ 个基学习器.

- (5) 算法第 5 步表示基学习器 $\tilde{h}_T\left(\boldsymbol{X}\right)$ 中每个叶节点的输出均使得上一轮迭代所取得模型的预测误差达到最小.

注 12.7 将 GBDT 应用于回归问题. 假设损失函数取为均方损失函数, 则

$$\mathscr{L}\left(Y, \tilde{h}_T\left(\boldsymbol{X}\right)\right) = \frac{1}{2}\left[Y - \tilde{h}_T\left(\boldsymbol{X}\right)\right]^2,$$

12.4.2 梯度 Boosting 方法

- 此时步骤 3 中拟残差的表达式变为

$$
\begin{aligned}
r_{ti} &= -\left[\frac{\partial \mathscr{L}\left[Y_i, \tilde{h}_T\left(\boldsymbol{X}_i\right)\right]}{\partial \tilde{h}_T\left(\boldsymbol{X}_i\right)}\right]_{\tilde{h}\left(\boldsymbol{X}_i\right)=\tilde{h}_{t-1}\left(\boldsymbol{X}_i\right)} \\
&= -\left[\frac{\partial \frac{1}{2}\left[Y_i - \tilde{h}_T\left(\boldsymbol{X}_i\right)\right]^2}{\partial \tilde{h}_T\left(\boldsymbol{X}_i\right)}\right]_{\tilde{h}\left(\boldsymbol{X}_i\right)=\tilde{h}_{t-1}\left(\boldsymbol{X}_i\right)} \\
&= \left[Y_i - \tilde{h}_T\left(\boldsymbol{X}_i\right)\right]_{\tilde{h}\left(\boldsymbol{X}_i\right)=\tilde{h}_{t-1}\left(\boldsymbol{X}_i\right)} \\
&= Y_i - \tilde{h}_{t-1}\left(\boldsymbol{X}_i\right).
\end{aligned}
\tag{12.4.16}
$$

12.4.2 梯度 Boosting 方法

- 此时对于回归问题, 拟残差 r_{ti} 就是真正的残差 $Y_i - \tilde{h}_{t-1}(\boldsymbol{X}_i)$. 进一步计算最优步长

$$\begin{aligned}\alpha_t &= \arg\min_{\alpha} \sum_{i=1}^{n} \left(Y_i - \tilde{h}_{t-1}(\boldsymbol{X}_i) - \alpha \cdot h(\boldsymbol{X}_i; \gamma_t)\right)^2 \\ &= \arg\min_{\alpha} \sum_{i=1}^{n} \left(r_{ti} - \alpha \cdot h(\boldsymbol{X}_i; \gamma_t)\right)^2\end{aligned} \tag{12.4.17}$$

- 因此当基学习器为决策树时, GBDT 恰好是前面的回归 Boosting 方法.

12.5 AdaBoost 算法实践

实践代码