《机器学习》课程系列

提升方法*

武汉纺织大学数学与计算机学院

杜小勤

2020/06/06

Contents

*本系列文档属于讲义性质，仅用于学习目的。Last updated on: June 13, 2020。

1 概述

提升 (Boosting) 方法是一种应用广泛的统计学习方法，它将多个学习器有机地组合在一起，形成一个性能更高的整体学习器。

例如，在分类问题上，通过给学习器分配不同的权重以及动态调整训练样本的权重，并将多个分类器进行线性组合，使得整体分类器的性能得到了有效的提高。

实际上，这也符合我们的常识与认知——对于一个复杂问题的决策，综合考虑多个专家的决策或通过多数表决方式，其效果往往要好于个别专家的单独决策。俗语“三个臭皮匠顶个诸葛亮”，说的就是这个道理。

提升方法的理论基础是，1984 年 Kearns 和 Valiant 提出的强可学习 (Strongly Learnable) 和弱可学习 (Weakly Learnable) 概念以及 1989 年 Schapire 证明的“在 PAC 学习的框架下，一个概念是强可学习的充分必要条件是，这个概念也是弱可学习的”——PAC 框架下强可学习与弱可学习的等价性原理。

在概率近似正确 (Probably Approximately Correct, PAC) 框架中，一个概念，如果存在一个多项式的学习算法能够学习它，并且正确率较高，则称这个概念是“强可学习的”；而如果一个多项式学习算法的学习正确率仅比随机猜测略好，则称这个概念是“弱可学习的”。

那么，一个非常自然的问题是，如果已经得到了一个仅比随机猜测略好的弱学习算法，能否将它提升为强学习算法呢？或者，在多数情况下，相对而言，得到一个弱学习算法通常要比得到一个强学习算法容易得多，这是否意味着，我们可以不必直接去获取强学习算法，而是通过提升弱学习算法，就可以得到与强学习算法一样的性能呢？

答案是肯定的，在下文，我们将从理论上表明，通过组合弱学习器，确实可以得到强学习器。但是，首先，从定性的角度来分析，要实现这一点，需要解决哪些问题。

我们已经知道，弱学习器的性能仅仅比随机学习器略好，这意味着，必定存在着单个弱学习器无法处理的实例样本点。那么，这些没有被正确处理的实例样本点，必然要交给后面的弱学习器继续处理。从优先级的角度看，未被正确处理

的实例样本点，其优先级 (紧迫程度) 应该要比那些已经被正确处理的实例样本点的优先级高。这可以通过动态地改变每个实例样本点的权值系数来实现——提高那些被前面的弱学习器处理错误的实例样本点的权值，降低那些已经被正确处理的实例样本点的权值。因此，那些未被正确处理的实例样本点，由于其权值变大，而成为后续弱学习器优先解决的目标。上述过程持续进行，理论上，当所有的实例样本点都被一系列的弱学习器“分而治之”而得到正确处理时，我们将会得到一个强学习器。本质上，提升方法是一种分治算法。

还存在一个问题，如何组合一系列弱学习器？可以考虑下面的方式：

- 多专家组合

  这是一种并行结构，所有的弱分类器都给出各自的预测结果，然后通过“组合器”将这些预测结果转换为最终结果，例如投票 (Voting) 及其变种、混合专家模型等。

  例如，可以使用带权值的多数表决的方式——所有的弱学习器参与表决；正确率高的弱学习器权值大，正确率低的弱学习器权值小。

- 多级组合

  这是一种串行结构——下一个分类器只在前一个分类器预测不够准确的实例上进行训练或检测，例如级联算法 (Cascading)。

最后一个问题是，如何获得不同的弱学习器。有以下几种方式可以考虑：

- 使用不同的弱学习算法，便得到不同的基本学习器：参数估计算法、非参数估计算法等；
- 使用相同的弱学习算法，但配置不同的参数：具有不同 K 值的 K-Means 算法或具有不同结构的神经网络等；
- 针对同一个输入对象，采用不同的表示方法，可以凸显事物的不同特征；
- 使用不同的训练集或不同的样本权值，前者称为装袋 (Boostrap Aggregating/Bagging)，后者称为提升 (Boosting)。

  每个基本模型使用从总体样本中随机抽样得到的不同数据集，再进行训练而得到，这种通过重抽样而得到不同训练数据集的过程被称为装袋。

提供给每个基本模型训练的数据集，采用不同的权重——增加被错误处理的样本权重，使得新模型重点关注这些样本，这种方法被称为提升。

装袋和提升，是构建组合模型的两种最主要的方法。所谓的组合模型，指的是由多个基本模型构成的模型，组合模型的预测效果，往往比任意一个基本模型的效果都要好。

本章，将讨论提升方法。

2 AdaBoost 模型

AdaBoost 模型是提升方法中最具代表性的算法之一，它将实例样本点的动态权值调整与弱分类器的线性组合完美而有效地集成在一起。

下面，将从加法模型 (Additive Model) 或线性组合模型、指数损失函数与前向分步算法 (Forward Stagewise Algorithm) 的角度，推导出 AdaBoost 模型。

2.1 加法模型与前向分步算法

所谓加法模型，指的是如下形式的模型：

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \beta_m b_m(\boldsymbol{x}; \boldsymbol{\gamma}_m) \tag{1}$$

其中，$b_m(\boldsymbol{x}; \boldsymbol{\gamma}_m)$ 为基函数或基学习器，β_m 为基函数的权重系数，$\boldsymbol{\gamma}_m$ 为基函数 $b_m(\boldsymbol{x}; \boldsymbol{\gamma}_m)$ 的参数[1]，M 表示基函数的个数。

对于给定的训练数据集 $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$(其中 $y_i = \{+1, -1\}$)，在损失函数 $L(y, f(\boldsymbol{x}))$ 给定的情况下，加法模型 $f(\boldsymbol{x})$ 的求解，可表

[1]需要注意的是，此处，$\boldsymbol{\gamma}_m$ 特定为一个与训练集相关的权重向量，并非基函数 $b_m(\boldsymbol{x}; \boldsymbol{\gamma}_m)$ 本身的模型参数，它的分量 $\gamma_{m,i}$ 表示基函数分配给第 i 个实例样本点的“优先级”权重。在迭代过程中，提升模型将依据每个实例样本点的分类结果动态地调整 $\gamma_{m,i}$。另一方面，提升模型要求基函数 $b_m(\boldsymbol{x}; \boldsymbol{\gamma}_m)$ 能对特定的数据分布进行学习。但是，$b_m(\boldsymbol{x}; \boldsymbol{\gamma}_m)$ 的最优化学习过程，对提升模型而言，是一个黑箱。因此，它自身的优化参数不在此列出。从下文的分析可以看出这一点。此处，为显式地表达所有实例样本点的优先级权重，故特意将 $\boldsymbol{\gamma}_m$ 列出。此外，一般情况下，$\boldsymbol{\gamma}_m$ 也并非优化对象。例如，在 AdaBoost 模型中，它自然地来自于每个实例样本点的指数损失值。相关详情，参见下文。

示为如下的最优化问题：

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{N} L\left(y_i, f\left(\boldsymbol{x}_i\right)\right) = \min_{\boldsymbol{\beta}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b_m\left(\boldsymbol{x}_i; \boldsymbol{\gamma}_m\right)\right) \tag{2}$$

其中，基函数 $b_m(\boldsymbol{x};\boldsymbol{\gamma}_m)$ 是已知的，但在每轮迭代时，仍需针对特定的数据分布进行最优化，以使某种度量下的分类误差率最小。因此，该最优化问题的求解目标是，确定最优的权重参数 $\boldsymbol{\beta}$。

对于上述最优化问题，直接优化较为复杂。一种可行的优化方法是，利用加法模型的特点，分阶段从前往后分步进行迭代优化：每次迭代时，只优化当前的 1 个基函数，并最优化求解相应的权重系数 β_m；利用某种度量标准 (例如指数损失函数) 直接求解基函数参数 $\boldsymbol{\gamma}_m$；随着迭代的进行，算法将逐步逼近整体优化的目标函数。我们把这种算法称为前向分步算法。

下面，给出加法模型的前向分步算法的一般形式。

算法 **2.1**（加法模型的前向分步算法）

Input:

Dataset: $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Loss Function: $L(y, f(\boldsymbol{x}))$

Basis Function: $b(\boldsymbol{x};\boldsymbol{\gamma})$

Numbers of Basis Functions: M

Output:

Additive Model: $f(\boldsymbol{x}) = \sum_{m=1}^{M} \beta_m b_m(\boldsymbol{x};\boldsymbol{\gamma}_m)$

Algorithm:

Initialize $\gamma_{m,i}$, (e.g.=$\frac{1}{N}$) $i = 1, 2, \ldots, N$

Initialize $f_0(\boldsymbol{x}) = 0$

for $m = 1, 2, \ldots, M$:

Black Box Optimization for:

$$b_m^*(\boldsymbol{x}, \boldsymbol{\gamma}_m) = \arg\min_{b_m} \sum_{i=1}^{N} \gamma_{m,i} I\left(y_i \neq b_m\left(\boldsymbol{x}_i, \boldsymbol{\gamma}_m\right)\right)$$

$$\beta_m^* = \arg\min_{\beta_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right) + \beta_m b_m^*\left(\boldsymbol{x}_i; \boldsymbol{\gamma}_m\right)\right)$$

$$f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m b_m^*\left(\boldsymbol{x}; \boldsymbol{\gamma}_m\right)$$

Get $\boldsymbol{\gamma}_m$ by $\boldsymbol{\gamma}_{m-1}$ and some metrics, e.g. Exponential Loss Function.

2.2 AdaBoost 模型的推导

2.2.1 指数损失函数

在 AdaBoost 模型中，基函数为基本分类器，损失函数则使用如下的指数函数：

$$L(y, f(\boldsymbol{x})) = \exp\left(-yf(\boldsymbol{x})\right) \tag{3}$$

实际上，对于二分类问题的 0-1 损失函数而言，上述指数损失函数是它的代理损失函数。

在“逻辑回归”一章中，我们已经对各种代理损失函数进行了介绍。此处，再简要地回顾一下。

在二分类学习器中，若 y 的取值为 $\{-1, 1\}$，则支持向量机、逻辑回归、AdaBoost 方法的损失函数分别具有如下形式：

$$\begin{aligned}
L(\boldsymbol{w}) &= \frac{1}{N}\sum_{i=1}^{N}\left[1 - y_i\boldsymbol{w}^T\boldsymbol{x_i}\right]_+ + \lambda\|\boldsymbol{w}\|^2 = \frac{1}{N}\sum_{i=1}^{N}\left[1 - y_i f\left(\boldsymbol{x_i}\right)\right]_+ + \lambda\|\boldsymbol{w}\|^2 \\
L(\boldsymbol{w}) &= \frac{1}{N}\sum_{i=1}^{N}\log\left(1 + e^{-y_i\boldsymbol{w}^T\boldsymbol{x_i}}\right) = \frac{1}{N}\sum_{i=1}^{N}\log\left(1 + e^{-y_i f(\boldsymbol{x_i})}\right) \\
L(\boldsymbol{w}) &= \frac{1}{N}\sum_{i=1}^{N} e^{-y_i\sum\limits_{m=1}^{M}\alpha_m G_m(\boldsymbol{x_i})} = \frac{1}{N}\sum_{i=1}^{N} e^{-y_i f(\boldsymbol{x_i})}
\end{aligned} \tag{4}$$

其中，支持向量机的损失函数中，含有正则项 $\|\boldsymbol{w}\|^2$。对于前面的 2 个公式，$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$；对于第 3 个公式，即 AdaBoost 方法，$f(\boldsymbol{x}) = \sum\limits_{m=1}^{M}\alpha_m G_m(\boldsymbol{x})$，它是 AdaBoost 的最终分类器（加法模型），$G_m(\boldsymbol{x})$ 为基本分类器，α_m 为相应的权重。

下面从单样本的角度考察这些损失函数：

$$\begin{aligned}
l(\boldsymbol{w}) &= \left[1 - yf(\boldsymbol{x})\right]_+ \\
l(\boldsymbol{w}) &= \log\left(1 + e^{-yf(\boldsymbol{x})}\right) \\
l(\boldsymbol{w}) &= e^{-yf(\boldsymbol{x})}
\end{aligned} \tag{5}$$

这 3 种损失函数被分别称为合页损失函数、逻辑回归损失函数与指数损失函数，它们都是 0-1 损失函数的上界，具有非常相似的曲线形状[2]，如图2-1所示。

因此，从损失函数的角度来看，支持向量机、逻辑回归与 AdaBoost 方法使用了不同的代理损失函数 (Surrogate Loss Function) 来定义经验风险函数或结构风

[2]注意：此处，逻辑回归损失函数的 log 以 2 为底。

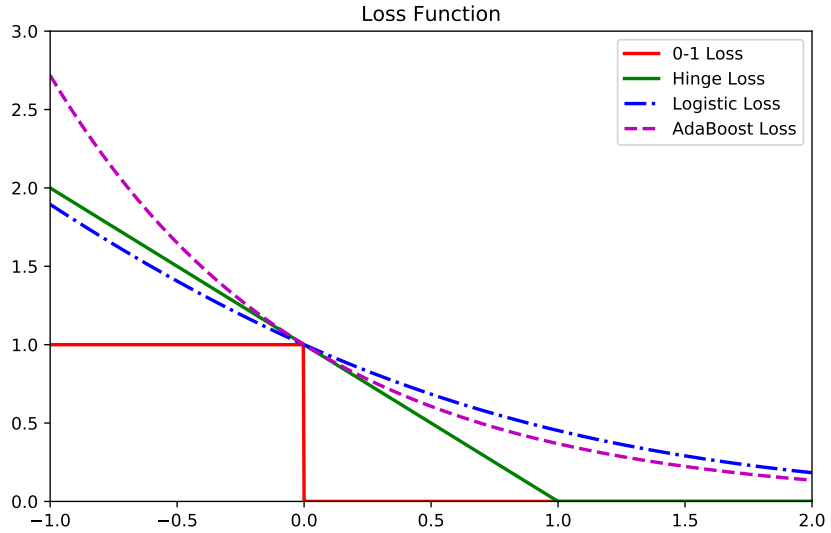

图 2-1: 代理损失函数

险函数，用来表示分类损失，从而实现了分类任务。支持向量机本身具有正则项，而逻辑回归没有正则项，但是可以添加正则项。AdaBoost 方法也没有正则项，可以通过早期停止 (Early Stopping) 技术起到正则项的作用。

2.2.2 基于前向分步算法的 AdaBoost 模型求解

假设 AdaBoost 模型的最终分类器是：

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x}; \boldsymbol{w}_m) \tag{6}$$

其中，$G_m(\boldsymbol{x}; \boldsymbol{w}_m)$ 为第 m 个基本分类器，$\boldsymbol{w}_m$ 为 $G_m(\boldsymbol{x}; \boldsymbol{w}_m)$ 的参数[3]；α_m 为 $G_m(\boldsymbol{x}; \boldsymbol{w}_m)$ 的权重系数；M 表示基本分类器的个数。AdaBoost 模型的损失函数，正如前一节所述，使用如下形式的指数函数：

$$L(y, f(\boldsymbol{x})) = \exp\left(-y f(\boldsymbol{x})\right) \tag{7}$$

[3]需要注意的是，与前文中的一般情形类似，对提升方法而言，基本分类器 $G_m(\boldsymbol{x}; \boldsymbol{w}_m)$ 每轮针对特定数据分布的优化过程是一个黑箱。因此，应该把此处的 $\boldsymbol{w}_m$ 理解成，从当前的基本分类器 $G_m(\boldsymbol{x}; \boldsymbol{w}_m)$ 的视角看，每个实例样本点希望被处理的“优先级”或“紧迫性”权重，即 $\boldsymbol{w}_m$ 不是基本分类器 $G_m(\boldsymbol{x}; \boldsymbol{w}_m)$ 自身的模型参数。对于第2.1节中的基函数参数 γ_m，也应该如此理解，前文已经对此进行了讨论。

因此，根据加法模型的前向分步算法 (2.1) 的一般形式，在第 m 次迭代，AdaBoost 模型的最优化目标为：

$$\begin{aligned}\alpha_m^* &= \arg\min_{\alpha_m} \sum_{i=1}^{N} L\left(y_i, f_m(\boldsymbol{x}_i)\right) \\ &= \arg\min_{\alpha_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right) + \alpha_m G_m\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right)\right) \\ &= \arg\min_{\alpha_m} \sum_{i=1}^{N} \exp\left(-y_i\left(f_{m-1}\left(\boldsymbol{x}_i\right) + \alpha_m G_m\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right)\right)\right)\end{aligned} \tag{8}$$

其中，α_m^* 为第 m 个基本分类器 $G_m(\boldsymbol{x}; \boldsymbol{w}_m)$ 的最优权重系数；$f_{m-1}(\boldsymbol{x})$ 为前 $m-1$ 轮已经得到的加法模型。

继续对 AdaBoost 模型的最优化目标函数 (公式 (8)) 进行变形：

$$\begin{aligned}\alpha_m^* &= \arg\min_{\alpha_m} \sum_{i=1}^{N} \exp\left(-y_i\left(f_{m-1}\left(\boldsymbol{x}_i\right) + \alpha_m G_m\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right)\right)\right) \\ &= \arg\min_{\alpha_m} \sum_{i=1}^{N} \exp\left(-y_i f_{m-1}\left(\boldsymbol{x}_i\right)\right) \exp\left(-y_i \alpha_m G_m\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right)\right)\end{aligned} \tag{9}$$

从上式可以看出，在第 m 轮，最优化求解的目标为 α_m^*。其一般形式 α_m，在公式中已经作为计算项出现，即在子公式 $\exp\left(-y_i \alpha_m G_m\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right)\right)$ 中。然而，$\boldsymbol{w}_m$ 只是以 $G_m\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right)$ 参数项的形式出现，并未以计算项形式出现在公式的其它地方。因此，我们自然会产生这样的疑问：$\boldsymbol{w}_m$ 是什么，其具体计算公式是怎样的？

实际上，仔细分析公式 (9)，可以发现，如果令 $w_{m-1,i} = \exp\left(-y_i f_{m-1}\left(\boldsymbol{x}_i\right)\right)$ 以及 $\boldsymbol{w}_{m-1} = (w_{m-1,1}, w_{m-1,2}, \ldots, w_{m-1,N})$，那么 $\boldsymbol{w}_{m-1}$ 中的每个分量实际上就是第 $m-1$ 轮相应实例样本点 $\boldsymbol{x}_i$ 被 $f_{m-1}\left(\boldsymbol{x}\right)$ 分类后的 (指数) 损失值——对于那些已经得到正确分类的实例样本点而言，该值较低；反之，该值较高。因此，在第 m 轮，如果将该值作为相应实例样本点需要被处理的优先级或紧迫度，那么这应该是十分合理的——那些在前一轮没有得到正确分类的样本点，在本轮将拥有较高的优先级，而被重点关注。

既然 $\boldsymbol{w}_{m-1}$ 与第 $m-1$ 轮得到的加法模型 $f_{m-1}(\boldsymbol{x})$ 相关，那么 $\boldsymbol{w}_m$ 就应该与第 m 轮得到的加法模型 $f_m(\boldsymbol{x})$ 相关，即 $\boldsymbol{w}_{m-1}$ 与 $\boldsymbol{w}_m$ 之间，必然存在某种递推

关系，情况的确如此。利用加法模型关系式，可以在它们之间建立联系：

$$\begin{cases} f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \alpha_m G_m(\boldsymbol{x};\boldsymbol{w}_m) \\ w_{m-1,i} = \exp\left(-y_i f_{m-1}\left(\boldsymbol{x}_i\right)\right) \end{cases} \Rightarrow \quad w_{m,i} = \exp\left(-y_i f_m\left(\boldsymbol{x}_i\right)\right) \quad \Rightarrow$$

$$\begin{aligned} w_{m,i} &= \exp\left(-y_i\left(f_{m-1}\left(\boldsymbol{x}_i\right) + \alpha_m G_m\left(\boldsymbol{x}_i;\boldsymbol{w}_m\right)\right)\right) \\ &= \exp\left(-y_i f_{m-1}\left(\boldsymbol{x}_i\right)\right)\exp\left(-y_i\alpha_m G_m\left(\boldsymbol{x}_i;\boldsymbol{w}_m\right)\right) \\ &= w_{m-1,i}\exp\left(-y_i\alpha_m G_m\left(\boldsymbol{x}_i;\boldsymbol{w}_m\right)\right) \end{aligned} \tag{10}$$

于是，公式 (9) 继续变形为：

$$\alpha_m^* = \arg\min_{\alpha_m}\sum_{i=1}^{N} w_{m-1,i}\exp\left(-y_i\alpha_m G_m\left(\boldsymbol{x}_i;\boldsymbol{w}_m\right)\right) \tag{11}$$

显然，$w_{m-1,i}$ 是一个与当前轮 (第 m 轮) 优化无关的量。

在最优化求解 α_m^* 之前，需要“最优化”针对当前 (第 m 轮) 特定数据分布的基分类器 $G_m^*\left(\boldsymbol{x};\boldsymbol{w}_m\right)$。当然，这一优化过程，对提升模型而言是透明的。

观察公式 (10) 和公式 (11)，可以发现：

$$\begin{aligned} \alpha_m^* &= \arg\min_{\alpha_m}\sum_{i=1}^{N} w_{m-1,i}\exp\left(-y_i\alpha_m G_m\left(\boldsymbol{x}_i;\boldsymbol{w}_m\right)\right) \\ &= \arg\min_{\alpha_m}\sum_{i=1}^{N} w_{m,i} \end{aligned} \tag{12}$$

因此，针对当前特定的数据“分布” $\boldsymbol{w}_{m-1}$[4]，最优基分类器 $G_m^*\left(\boldsymbol{x};\boldsymbol{w}_m\right)$ 由下式得到：

$$G_m^*\left(\boldsymbol{x};\boldsymbol{w}_m\right) = \arg\min_{G_m}\sum_{i=1}^{N} w_{m-1,i} I\left(y_i \neq G_m\left(\boldsymbol{x}_i;\boldsymbol{w}_m\right)\right) \tag{13}$$

其含义显而易见：最优基分类器 $G_m^*\left(\boldsymbol{x};\boldsymbol{w}_m\right)$ 使得第 m 轮训练数据的带权分类误差总和最小。因此，一般要选用支持带权值样本的分类算法作为基分类器，使得基分类器的上述最优化目标得以实现。

下面，开始最优化求解 α_m^*。令公式 (11) 中的目标函数为 $\mathcal{L}(\boldsymbol{\alpha})$，求偏导数

[4]注意，$\boldsymbol{w}_{m-1}$ 是第 $m-1$ 轮分类结束之后得到的新数据“分布”，即本轮需要面对的数据“分布”。而 $\boldsymbol{w}_m$ 是本轮分类结束之后，得到的新数据“分布”，提供给第 $m+1$ 轮的基分类器进行处理。

$\frac{\partial \mathcal{L}(\boldsymbol{\alpha})}{\partial \alpha_m}$，并令其等于 0：

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{\alpha})}{\partial \alpha_m} &= \sum_{i=1}^{N} w_{m-1,i} \exp\left(-y_i \alpha_m G_m^*\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right)\right)\left(-y_i G_m^*\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right)\right) \\ &= -\sum_{i=1}^{N} w_{m-1,i} y_i G_m^*\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right) \exp\left(-y_i \alpha_m G_m^*\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right)\right) = 0\end{aligned} \tag{14}$$

继续求解上式，得到：

$$\begin{aligned}&\sum_{i=1}^{N} w_{m-1,i} y_i G_m^*\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right) \exp\left(-y_i \alpha_m G_m^*\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right)\right) = 0 \quad \Rightarrow \\ &\sum_{y_i = G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)} w_{m-1,i} e^{-\alpha_m} - \sum_{y_i \neq G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)} w_{m-1,i} e^{\alpha_m} = 0 \quad \Rightarrow \\ &\sum_{y_i = G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)} w_{m-1,i} e^{-\alpha_m} = \sum_{y_i \neq G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)} w_{m-1,i} e^{\alpha_m} \quad \Rightarrow \\ &\sum_{y_i = G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)} w_{m-1,i} e^{-\alpha_m} + \sum_{y_i \neq G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)} w_{m-1,i} e^{-\alpha_m} \\ &= \sum_{y_i \neq G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)} w_{m-1,i} e^{\alpha_m} + \sum_{y_i \neq G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)} w_{m-1,i} e^{-\alpha_m} \quad \Rightarrow \\ &e^{-\alpha_m} \sum_{i=1}^{N} w_{m-1,i} = \left(e^{\alpha_m} + e^{-\alpha_m}\right) \sum_{i=1}^{N} w_{m-1,i} I(y_i \neq G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)) \quad \Rightarrow \\ &\frac{e^{-\alpha_m}}{e^{\alpha_m} + e^{-\alpha_m}} = \frac{\sum\limits_{i=1}^{N} w_{m-1,i} I(y_i \neq G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m))}{\sum\limits_{i=1}^{N} w_{m-1,i}} = e_m\end{aligned} \tag{15}$$

其中，e_m 为新设变量，表示 (带权值) 分类误差率。它与 α_m 之间的关系为：

$$e_m = \frac{e^{-\alpha_m}}{e^{\alpha_m} + e^{-\alpha_m}} = \frac{1}{1 + e^{2\alpha_m}} \quad \Rightarrow \quad \alpha_m^* = \alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m} \tag{16}$$

上式的含义是什么？

首先，需要明确的是，关于分类误差率 e_m，它表示的是，在给定第 m 轮最优基分类器 $G_m^*(\boldsymbol{x}; \boldsymbol{w}_m)$ 和第 $m-1$ 轮实例样本点权重向量 $\boldsymbol{w}_{m-1}$ 的情况下，最优基分类器 $G_m^*(\boldsymbol{x}; \boldsymbol{w}_m)$ 对所有实例样本点进行分类所造成的 (带权) 分类错误率。而 α_m^* 的求解公式 (16) 告诉我们，如果将 $1 - e_m$ 当作分类正确的概率，而把 e_m 当做分类错误的概率时，那么给当前最优基分类器 $G_m^*(\boldsymbol{x}; \boldsymbol{w}_m)$ 分配的最佳权重 α_m^* 就是对数几率 $\log \frac{1-e_m}{e_m}$ 的一半，或者说，每一轮最小化指数损失，实质上是在给每个最优基分类器 $G_m^*(\boldsymbol{x}; \boldsymbol{w}_m)$ 分配一个满足对数几率关系的逻辑回归权重参数[5]。

[5]关于对数几率与逻辑回归的相关概念，请阅读“逻辑回归”的相关章节。

通过简单地分析，可以发现，当 $e_m \leqslant \frac{1}{2}$ 时，$\alpha_m^* \geqslant 0$，且 α_m^* 随着 e_m 的减少而增大。因此，分类误差率越小的最优基分类器，在最终的分类器中所占的比重越大，发挥的作用也越大。

现在再来回顾一下公式 (10) 和公式 (11)，并将最优基分类器 $G_m^*(\boldsymbol{x};\boldsymbol{w}_m)$ 代入，得到：

$$\begin{aligned}\alpha_m^* &= \arg\min_{\alpha_m} \sum_{i=1}^{N} w_{m-1,i} \exp\left(-y_i \alpha_m G_m^*\left(\boldsymbol{x}_i;\boldsymbol{w}_m\right)\right) \\ &= \arg\min_{\alpha_m} \sum_{i=1}^{N} w_{m,i}\end{aligned} \tag{17}$$

可以发现，上式表达的含义是，在第 m 轮 (本轮) 中，为最优基分类器 $G_m^*(\boldsymbol{x};\boldsymbol{w}_m)$ 选择的最佳权重 α_m^* 也应该使得分类完成后所有实例样本点的带权值总和最小化。

总之，α_m^* 的选取，要确保如下目标成立：

- 最优基分类器的权重 α_m^* 与其分类误差率成反比关系；
- α_m^* 使得每轮分类后所有实例样本点的带权值总和最小化；

当然，上述 2 个目标是统一的，它们通过指数损失函数完美地结合在一起。

2.3 AdaBoost 算法

下面给出 AdaBoost 算法。

算法 **2.2**（*AdaBoost* 算法）

Input:

Dataset: $T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_N,y_N)\}$

Numbers of Basis Functions: M

Basis Classifier: $G_m(\boldsymbol{x};\boldsymbol{w}_m)$

Output:

Additive Model: $f(\boldsymbol{x})=\sum\limits_{m=1}^{M}\alpha_m G_m^*(\boldsymbol{x};\boldsymbol{w}_m)$

Final Classifier: $G(\boldsymbol{x})=\text{sign}\left(f(\boldsymbol{x})\right)$

Algorithm:

1. Initialize $\boldsymbol{w}_0=(w_{0,1},w_{0,2},\ldots,w_{0,N})$, $w_{0,i}=\frac{1}{N}$
2. Initialize $f_0(\boldsymbol{x})=0$
3. for $m=1,2,\ldots,M$:
4. Black Box Optimization for:
5. $$G_m^*\left(\boldsymbol{x};\boldsymbol{w}_m\right)=\arg\min_{G_m}\sum_{i=1}^{N}w_{m-1,i}I(y_i\neq G_m\left(\boldsymbol{x}_i;\boldsymbol{w}_m\right))$$
6. $$e_m=\frac{\sum\limits_{i=1}^{N}w_{m-1,i}I(y_i\neq G_m^*(\boldsymbol{x}_i;\boldsymbol{w}_m))}{\sum\limits_{i=1}^{N}w_{m-1,i}}=\sum_{i=1}^{N}\frac{w_{m-1,i}}{\sum\limits_{i=1}^{N}w_{m-1,i}}I(y_i\neq G_m^*(\boldsymbol{x}_i;\boldsymbol{w}_m))$$
7. $$\alpha_m^*=\frac{1}{2}\log\frac{1-e_m}{e_m}$$
8. $$\boldsymbol{w}_m=(w_{m,1},w_{m,2},\ldots,w_{m,N})\text{ where:}$$
9. $$w_{m,i}=w_{m-1,i}\exp\left(-y_i\alpha_m G_m^*\left(\boldsymbol{x}_i;\boldsymbol{w}_m\right)\right)$$
10. $$f_m(\boldsymbol{x})=f_{m-1}(\boldsymbol{x})+\alpha_m G_m^*\left(\boldsymbol{x};\boldsymbol{w}_m\right)$$

上面的算法，直接来源于第2.2.2节。在实际应用中，一般要让 $\boldsymbol{w}_m$ 成为一个概率分布。因此，需要为伪代码的第9行引入规范化因子：

$$Z_m=\sum_{j=1}^{N}w_{m-1,j}\exp\left(-y_j\alpha_m G_m^*\left(\boldsymbol{x}_j;\boldsymbol{w}_m\right)\right)\tag{18}$$

从而使得伪代码的第9行变为：

$$w_{m,i}=\frac{w_{m-1,i}\exp\left(-y_i\alpha_m G_m^*\left(\boldsymbol{x}_i;\boldsymbol{w}_m\right)\right)}{Z_m}\tag{19}$$

将 $\boldsymbol{w}_m$ 权重向量规范化为权重概率分布，不会改变第5行的最优化结果。原因是，规范化项 Z_m 对于原 $\boldsymbol{w}_m$ 向量中的所有分量而言是一个常数，因而不会影响到 $G_m(\boldsymbol{x};\boldsymbol{w}_m)$ 的最优化。

至于第6行中的计算公式：

$$\begin{aligned} e_m &= \sum_{i=1}^{N} \frac{w_{m-1,i}}{\sum\limits_{i=1}^{N} w_{m-1,i}} I(y_i \neq G_m^*(\boldsymbol{x}_i;\boldsymbol{w}_m)) \\ &= \sum_{i=1}^{N} \frac{w_{m-1,i}/Z_m}{\sum\limits_{i=1}^{N} w_{m-1,i}/Z_m} I(y_i \neq G_m^*(\boldsymbol{x}_i;\boldsymbol{w}_m)) \end{aligned} \tag{20}$$

上式表明，权重向量 $\boldsymbol{w}_m$ 的概率规范化对 e_m 的计算没有任何影响，且在 $w_{m-1,i}$ 规范化为概率的情况下，有：

$$\begin{aligned} &\sum_{i=1}^{N} w_{m-1,i} = 1 \quad \Rightarrow \\ &e_m = \sum_{i=1}^{N} w_{m-1,i} I(y_i \neq G_m^*(\boldsymbol{x}_i;\boldsymbol{w}_m)) \end{aligned} \tag{21}$$

于是，得到简化版 AdaBoost 算法。

算法 **2.3**（简化版 *AdaBoost* 算法）

Input:

Dataset: $T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_N,y_N)\}$

Numbers of Basis Functions: M

Basis Classifier: $G_m(\boldsymbol{x};\boldsymbol{w}_m)$

Output:

Additive Model: $f(\boldsymbol{x})=\sum\limits_{m=1}^{M}\alpha_m G_m^*(\boldsymbol{x};\boldsymbol{w}_m)$

Final Classifier: $G(\boldsymbol{x})=\text{sign}\left(f(\boldsymbol{x})\right)$

Algorithm:

1. Initialize $\boldsymbol{w}_0=(w_{0,1},w_{0,2},\ldots,w_{0,N})$, $w_{0,i}=\frac{1}{N}$
2. Initialize $f_0(\boldsymbol{x})=0$
3. for $m=1,2,\ldots,M$:
4. Black Box Optimization for:
5. $$G_m^*\left(\boldsymbol{x};\boldsymbol{w}_m\right)=\arg\min_{G_m}\sum_{i=1}^{N}w_{m-1,i}I(y_i\neq G_m\left(\boldsymbol{x}_i;\boldsymbol{w}_m\right))$$
6. $e_m=\sum\limits_{i=1}^{N}w_{m-1,i}I(y_i\neq G_m^*(\boldsymbol{x}_i;\boldsymbol{w}_m))$
7. $\alpha_m^*=\frac{1}{2}\log\frac{1-e_m}{e_m}$
8. $\boldsymbol{w}_m=(w_{m,1},w_{m,2},\ldots,w_{m,N})$ where:
9. $$w_{m,i}=\frac{w_{m-1,i}\exp(-y_i\alpha_m G_m^*(\boldsymbol{x}_i;\boldsymbol{w}_m))}{\sum\limits_{j=1}^{N}w_{m-1,j}\exp(-y_j\alpha_m G_m^*(\boldsymbol{x}_j;\boldsymbol{w}_m))}$$
10. $f_m(\boldsymbol{x})=f_{m-1}(\boldsymbol{x})+\alpha_m G_m^*\left(\boldsymbol{x};\boldsymbol{w}_m\right)$

最后，我们来直观感受一下权重 $\boldsymbol{w}_m$ 的指数级缩放效果。在初始时，$w_{0,i}=\frac{1}{N}$，$i=1,2,\ldots,N$。在第 1 轮，假设第 i 个实例样本点被正确分类，第 j 个实例样本点被误分类，那么它们的权值将分别调整为：

$$\begin{cases}w_{1,i}=\dfrac{1}{N}e^{-\alpha_1^*}/Z_m, & y_i=G_m^*(\boldsymbol{x}_i;\boldsymbol{w}_1)\\ w_{1,j}=\dfrac{1}{N}e^{\alpha_1^*}/Z_m, & y_j\neq G_m^*(\boldsymbol{x}_j;\boldsymbol{w}_1)\end{cases}\quad\Rightarrow\quad\frac{w_{1,j}}{w_{1,i}}=e^{2\alpha_1^*}=\frac{1-e_m}{e_m}\tag{22}$$

正常情况下，对于仅比随机猜测略好的弱分类器，$e_m\leqslant\frac{1}{2}$ 成立，得到 $\frac{1-e_m}{e_m}\geqslant 1$ 和

No.	1	2	3	4	5	6	7	8	9	10
x	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1

图 2-2: 例2.1的训练数据

$\alpha_1^* \geqslant 0$。因此，$w_{1,i}$ 因实例样本点被正确分类而缩小，$w_{1,j}$ 因实例样本点被误分类而放大，后者是前者的 $e^{2\alpha_1^*}$ 倍，而原先两者相等。因此，误分类实例样本点因权重变大，将是下一轮的重点分类目标。

2.4 AdaBoost 算法示例

例 2.1 给定如图*2-2*所示的训练数据。假设分类决策由简单的弱分类器 $x < v$ 或 $x > v$ 产生，其阈值 v 使得该分类器在训练数据集上的分类误差率最低。试用 *AdaBoost* 算法学习一个强分类器。

解：

1. $m = 0$，初始化：

$$\begin{aligned} \boldsymbol{w}_0 &= (w_{0,1}, w_{0,2}, \ldots, w_{0,10}) \\ w_{0,i} &= \frac{1}{10} = 0.1 \end{aligned} \tag{23}$$

2. $m = 1$：在权值分布为 $\boldsymbol{w}_0$ 的数据集上，阈值取 2.5，分类误差率最小，弱分类器为：

$$G_1(x) = \begin{cases} 1 & x < 2.5 \\ -1 & x > 2.5 \end{cases} \tag{24}$$

$G_1(x)$ 的 (带权值) 误差率为：$e_1 = 0.3$，权重 $\alpha_1 = \frac{1}{2}\log\frac{1-e_1}{e_1} = 0.4236$。更新训练样本点的权值分布：

$$\boldsymbol{w}_1 = (0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.1666, 0.1666, 0.1666, 0.0715) \tag{25}$$

得到加法模型 $f_1(x) = 0.4236G_1(x)$，分类器 $\operatorname{sign}(f_1(x))$ 在训练数据集上有 3 个误分类点。

3. $m=2$：在权值分布为 $\boldsymbol{w}_1$ 的数据集上，阈值取 8.5，分类误差率最小，弱分类器为：

$$G_2(x)=\begin{cases}1 & x<8.5\\ -1 & x>8.5\end{cases} \tag{26}$$

$G_2(x)$ 的 (带权值) 误差率为：$e_2=0.2143$，权重 $\alpha_2=0.6496$。更新训练样本点的权值分布：

$$\boldsymbol{w}_2=(0.0455,0.0455,0.0455,0.1667,0.1667,0.1667,0.1060,0.1060,0.1060,0.0455) \tag{27}$$

得到加法模型 $f_2(x)=0.4236G_1(x)+0.6496G_2(x)$，分类器 $\operatorname{sign}(f_2(x))$ 在训练数据集上有 3 个误分类点。

4. $m=3$：在权值分布为 $\boldsymbol{w}_2$ 的数据集上，阈值取 5.5，分类误差率最小，弱分类器为：

$$G_3(x)=\begin{cases}1 & x>5.5\\ -1 & x<5.5\end{cases} \tag{28}$$

$G_3(x)$ 的 (带权值) 误差率为：$e_3=0.1820$，权重 $\alpha_3=0.7514$。更新训练样本点的权值分布：

$$\boldsymbol{w}_3=(0.125,0.125,0.125,0.102,0.102,0.102,0.065,0.065,0.065,0.125) \tag{29}$$

得到加法模型 $f_3(x)=0.4236G_1(x)+0.6496G_2(x)+0.7514G_3(x)$，分类器 $\operatorname{sign}(f_3(x))$ 在训练数据集上有 0 个误分类点。

于是，得到最终的分类器为：

$$G(x)=\operatorname{sign}(f_3(x))=\operatorname{sign}(0.4236G_1(x)+0.6496G_2(x)+0.7514G_3(x)) \tag{30}$$

□

2.5 AdaBoost 算法的训练误差分析

前面，我们根据加法模型、前向分步算法以及指数损失函数，推导出了 AdaBoost 算法。下面将证明，在一定 (较弱) 条件下，AdaBoost 学习算法的训练误差以指数级下降。

定理 2.1（*AdaBoost* 算法的训练误差上界）*AdaBoost* 算法得到的最终分类器 $G(\boldsymbol{x})$ 的训练误差上界为：

$$\frac{1}{N}\sum_{i=1}^{N} I\left(y_i \neq G(\boldsymbol{x}_i)\right) \leqslant \frac{1}{N}\sum_{i=1}^{N} \exp\left(-y_i f\left(\boldsymbol{x}_i\right)\right) = \prod_{m=1}^{M} Z_m \tag{31}$$

其中：

$$\begin{aligned} G(\boldsymbol{x}) &= \operatorname{sign}(f(\boldsymbol{x})) \\ f(\boldsymbol{x}) &= \sum_{m=1}^{M} \alpha_m G_m^*(\boldsymbol{x}; \boldsymbol{w}_m) \\ Z_m &= \sum_{j=1}^{N} w_{m-1,j} \exp\left(-y_j \alpha_m G_m^*\left(\boldsymbol{x}_j; \boldsymbol{w}_m\right)\right) \end{aligned} \tag{32}$$

证明：对于 $\forall i (i = 1, 2, \ldots, N)$，根据指数函数性质，$\exp\left(-y_i f\left(\boldsymbol{x}_i\right)\right) > 0$ 成立；且当 $y_i \neq G(\boldsymbol{x}_i)$ 时，$y_i f\left(\boldsymbol{x}_i\right) \leqslant 0$[6]，则 $I\left(y_i \neq G(\boldsymbol{x}_i)\right) = 1 \leqslant \exp\left(-y_i f\left(\boldsymbol{x}_i\right)\right)$，从而得到：

$$\frac{1}{N}\sum_{i=1}^{N} I\left(y_i \neq G(\boldsymbol{x}_i)\right) \leqslant \frac{1}{N}\sum_{i=1}^{N} \exp\left(-y_i f\left(\boldsymbol{x}_i\right)\right) \tag{33}$$

定理公式的前半部分得证。下面，再证定理公式的后半部分。利用公式 (19)，得到：

$$w_{m-1,i} \exp\left(-y_i \alpha_m G_m^*\left(\boldsymbol{x}_i; \boldsymbol{w}_m\right)\right) = Z_m w_{m,i} \tag{34}$$

于是，利用 $w_{0,i} = \frac{1}{N}$：

$$\begin{aligned} \frac{1}{N}\sum_{i=1}^{N} \exp\left(-y_i f\left(\boldsymbol{x}_i\right)\right) &= \sum_{i=1}^{N} w_{0,i} \exp\left(-y_i \sum_{m=1}^{M} \alpha_m G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)\right) \\ &= \sum_{i=1}^{N} w_{0,i} \prod_{m=1}^{M} \exp\left(-y_i \alpha_m G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)\right) \\ &= \sum_{i=1}^{N} Z_1 w_{1,i} \prod_{m=2}^{M} \exp\left(-y_i \alpha_m G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)\right) \\ &= \sum_{i=1}^{N} Z_1 Z_2 w_{2,i} \prod_{m=3}^{M} \exp\left(-y_i \alpha_m G_m^*(\boldsymbol{x}_i; \boldsymbol{w}_m)\right) \\ &= \vdots \\ &= \sum_{i=1}^{N} Z_1 Z_2 \ldots Z_M w_{M,i} = \prod_{m=1}^{M} Z_m \sum_{i=1}^{N} w_{M,i} = \prod_{m=1}^{M} Z_m \end{aligned} \tag{35}$$

[6]根据定义，$G(\boldsymbol{x})$ 是 $f(\boldsymbol{x})$ 的符号二值化函数 (取值 $\{+1, -1\}$)；而 $f(\boldsymbol{x})$ 是加法模型，在误分类时，$yf(\boldsymbol{x}) < 0$ 成立。一般，也可以将 0 情形考虑在内，从而变为 $yf(\boldsymbol{x}) \leqslant 0$。

□

上述定理表明，AdaBoost 算法的训练误差上界是由各个弱分类器的规范化因子 Z_m 共同确定的。其意义在于，可以在每一轮关于 $G_m^*(\boldsymbol{x};\boldsymbol{w}_m)$ 最小化 Z_m，使得训练误差下降最快。下面的定理，是上述定理在二分类问题上的拓展。

定理 **2.2**（二分类问题中 *AdaBoost* 算法的训练误差上界）

$$\prod_{m=1}^{M} Z_m = \prod_{m=1}^{M} \left(2\sqrt{e_m\left(1-e_m\right)}\right) = \prod_{m=1}^{M} \sqrt{1-4\gamma_m^2} \leqslant \exp\left(-2\sum_{m=1}^{M}\gamma_m^2\right) \tag{36}$$

其中，$\gamma_m = \frac{1}{2} - e_m$。

证明：根据 Z_m 的计算公式 (32)、公式 $e_m = \sum\limits_{j=1}^{N} w_{m-1,j} I(y_j \neq G_m^*(\boldsymbol{x}_j;\boldsymbol{w}_m))$(公式 (21)) 以及公式 $\alpha_m = \frac{1}{2}\log\frac{1-e_m}{e_m}$(公式 (16))，有：

$$\begin{aligned} Z_m &= \sum_{j=1}^{N} w_{m-1,j} \exp\left(-y_j \alpha_m G_m^*\left(\boldsymbol{x}_j;\boldsymbol{w}_m\right)\right) \\ &= \sum_{y_j = G_m^*(\boldsymbol{x}_j;\boldsymbol{w}_m)} w_{m-1,j} \mathrm{e}^{-\alpha_m} + \sum_{y_j \neq G_m^*(\boldsymbol{x}_j;\boldsymbol{w}_m)} w_{m-1,j} \mathrm{e}^{\alpha_m} \\ &= \mathrm{e}^{-\alpha_m}\left(1-e_m\right) + \mathrm{e}^{\alpha_m} e_m = 2\sqrt{e_m\left(1-e_m\right)} = \sqrt{1-4\gamma_m^2} \end{aligned} \tag{37}$$

其中，最后一步的推导：令 $\gamma_m = \frac{1}{2} - e_m$，并将 $e_m = \frac{1}{2} - \gamma_m$ 代入即可。

进一步地，使用不等式 $1 + x \leqslant \mathrm{e}^x$，可得[7]：

$$\sqrt{1-4\gamma_m^2} \leqslant \left(\mathrm{e}^{-4\gamma_m^2}\right)^{\frac{1}{2}} = \mathrm{e}^{-2\gamma_m^2} \tag{38}$$

于是，得到：

$$\prod_{m=1}^{M} \sqrt{1-4\gamma_m^2} \leqslant \prod_{m=1}^{M} \mathrm{e}^{-2\gamma_m^2} = \exp\left(-2\sum_{m=1}^{M}\gamma_m^2\right) \tag{39}$$

□

推论 **2.1** 如果存在 $\gamma > 0$，且对 $\forall m (m = 1, 2, \ldots, M)$，$\gamma_m \geqslant \gamma$ 成立，那么二分类 *AdaBoost* 算法的训练误差上界为：

$$\frac{1}{N}\sum_{i=1}^{N} I\left(y_i \neq G(\boldsymbol{x}_i)\right) \leqslant \exp(-2M\gamma^2) \tag{40}$$

[7]将 e^x 在 $x = 0$ 处展开为泰勒级数：$\mathrm{e}^x = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots$，$-\infty < x < +\infty$，则 $\mathrm{e}^x \geqslant 1 + x$。

该推论表明，只要任一弱分类器 $G_m(\boldsymbol{x};\boldsymbol{w}_m)$ 的性能仅比随机分类器略好，即 $e_m < \frac{1}{2}$，$\gamma_m = \frac{1}{2} - e_m > 0$ 成立，则可取 $\gamma = \min\{\gamma_1, \gamma_2, \ldots, \gamma_M\}$，那么上述推论就成立。

该推论的意义在于，在正常情况下，AdaBoost 算法的训练误差将随着弱分类器个数的增加呈指数级下降。

3 回归提升树

3.1 回归提升树算法

提升树 (Boosting Tree) 指的是以分类树或回归树作为弱学习器的提升方法。

在例2.1中，使用了一个简单的二分类器。例如，当 $m = 1$ 时，如果 $x < 2.5$，则 $G_1(x) = 1$；否则，$G_1(x) = -1$。我们可以将该分类器看作一棵由 1 个根节点直接连接 2 个叶子节点的简单分类决策树或决策树桩 (Decision Stump)。

本小节，将讨论回归提升树。它可以表示为如下形式的加法模型[8]：

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} T_m(\boldsymbol{x}) \tag{41}$$

其中，M 表示树的棵树，$T_m(\boldsymbol{x})$ 表示回归树。

对于给定的输入空间 $\mathcal{X}$，回归树 $T_m(\boldsymbol{x})$ 将它划分为 J 个互不相交的区域 $\{R_{m,1}, R_{m,2}, \ldots, R_{m,J}\}$，且在每个区域上输出的常量为 $c_{m,j}$，那么回归树 $T_m(\boldsymbol{x})$ 可表示为：

$$T_m(\boldsymbol{x}) = \sum_{j=1}^{J} c_{m,j} I\left(\boldsymbol{x} \in R_{m,j}\right) \tag{42}$$

其中，J 表示回归树中叶节点的个数。

与 AdaBoost 模型一样，回归提升树也采用前向分步算法进行最优化：

$$\begin{aligned} T_m^* &= \arg\min_{T_m} \sum_{i=1}^{N} L\left(y_i, f_m\left(\boldsymbol{x}\right)\right) \\ &= \arg\min_{T_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}\left(\boldsymbol{x}\right) + T_m\left(\boldsymbol{x}\right)\right) \\ f_m\left(\boldsymbol{x}\right) &= f_{m-1}\left(\boldsymbol{x}\right) + T_m^*\left(\boldsymbol{x}\right) \quad m = 1, 2, \ldots, M \end{aligned} \tag{43}$$

[8]对提升树模型而言，$T(\boldsymbol{x})$ 的最优化过程是一个黑箱。因此，没有将 $T(\boldsymbol{x})$ 自身的参数列出。

其中，初始提升树 $f_0(\boldsymbol{x}) = 0$。

在回归提升树中，一般可选取平方误差损失函数：

$$L\left(y, f\left(\boldsymbol{x}\right)\right) = \left(y - f\left(\boldsymbol{x}\right)\right)^2 \tag{44}$$

因此，在加法模型下，可将损失函数进一步变形为：

$$\begin{aligned} L\left(y, f_m\left(\boldsymbol{x}\right)\right) &= \left(y - f_m\left(\boldsymbol{x}\right)\right)^2 \\ &= \left(y - f_{m-1}\left(\boldsymbol{x}\right) - T_m\left(\boldsymbol{x}\right)\right)^2 \\ &= \left(r - T_m\left(\boldsymbol{x}\right)\right)^2 \end{aligned} \tag{45}$$

其中，$r = y - f_{m-1}\left(\boldsymbol{x}\right)$ 表示模型 $f_{m-1}\left(\boldsymbol{x}\right)$ 拟合 y 的残差 (Residual)。当 $m = 1$ 时，模型 $f_0(\boldsymbol{x})$ 拟合原始期望输出 y 的残差为 $r_1 = y$，因为模型 $f_0(\boldsymbol{x}) = 0$，初始提升树，没有拟合效果；当 $m = 2$ 时，模型 $f_1(\boldsymbol{x})$ 拟合 y 的残差为 $r_2 = y - f_1(\boldsymbol{x})$。上述过程，一直持续下去，残差将逐渐变小，直至为 0 或满足要求为止；最后，得到回归提升树模型：

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} T_m^*(\boldsymbol{x}) \tag{46}$$

综上所述，回归提升树的算法相当简洁——每次迭代步，只需简单地拟合残差。

下面，给出回归提升树算法。

算法 **3.1**（回归提升树算法）

No.	1	2	3	4	5	6	7	8	9	10
x	1	2	3	4	5	6	7	8	9	10
y	5.56	5.70	5.91	6.40	6.80	7.05	8.90	8.70	9.00	9.05

图 3-3: 例3.1的训练数据

Input:

Dataset: $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Numbers of Regression Tree: M

Regression Tree: $T_m(\boldsymbol{x})$

Output:

Regression Boosting Tree: $f(\boldsymbol{x}) = \sum_{m=1}^{M} T_m^*(\boldsymbol{x})$

Algorithm:

1. Initialize $f_0(\boldsymbol{x}) = 0$
2. for $m = 1, 2, \ldots, M$:
3. $r_{m,i} = y_i - f_{m-1}(\boldsymbol{x}_i),\ i = 1, 2, \ldots, N$
4. Black Box Optimization for:
5. $$T_m^*(\boldsymbol{x}) = \arg\min_{T_m} \sum_{i=1}^{N} (r_{m,i} - T_m(\boldsymbol{x}))^2$$
6. $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + T_m^*(\boldsymbol{x})$

3.2 回归提升树算法示例

例 **3.1** 如图*3-3*所示，列出了训练数据，试使用回归树桩学习回归提升树模型。

解：

1. $m = 0$，初始化 $f_0(\boldsymbol{x}) = 0$；
2. $m = 1$，$r_{1,i} = y_i$，$i = 1, 2, \ldots, N$；

3. 黑箱优化回归树 $T_1(\boldsymbol{x})$：

$$\begin{cases} T_1^*(\boldsymbol{x}) = \arg\min\limits_{T_1(\boldsymbol{x};s)} \left[\sum\limits_{x_i \in R_{1,1}} (y_i - c_{1,1})^2 + \sum\limits_{x_i \in R_{1,2}} (y_i - c_{1,2})^2 \right] \\ R_{1,1} = \{x | x \leqslant s\}, R_{1,2} = \{x | x > s\} \\ c_{1,1} = \dfrac{1}{|R_{1,1}|} \sum\limits_{x_i \in R_{1,1}} y_i, c_{1,2} = \dfrac{1}{|R_{1,2}|} \sum\limits_{x_i \in R_{1,2}} y_i \end{cases} \tag{47}$$

其中，s 表示训练数据的切分点，$R_{1,1}$ 和 $R_{1,2}$ 分别表示切分之后的训练数据子集，$|R_{1,1}|$ 和 $|R_{1,2}|$ 分别表示 $R_{1,1}$ 和 $R_{1,2}$ 的样本个数。

对于本例，可以手工计算，以模拟回归树 $T_1(\boldsymbol{x})$ 的最优化。算法的编程实现，请参见附录。

根据所给的训练数据，可考虑的切分点 s，列出如下：

$$1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5$$

当 $s = 1.5$ 时，计算得到：$R_{1,1} = \{1\}$，$R_{1,2} = \{2, 3, \ldots, 10\}$，$c_{1,1} = 5.56$，$c_{1,2} = 7.50$，

$$t(s) = \sum_{x_i \in R_{1,1}} (y_i - c_{1,1})^2 + \sum_{x_i \in R_{1,2}} (y_i - c_{1,2})^2 = 0 + 15.72 = 15.72$$

当 s 为其它数值时，分别得到相应的 $R_{1,1}$、$R_{1,2}$、$c_{1,1}$、$c_{1,2}$ 和 $t(s)$ 值，计算方法类似。将 s 与相应的 $t(s)$ 列出如下：

s	1.5	2.5	3.5	4.5	5.5	6.5	7.5	8.5	9.5
$t(s)$	15.72	12.07	8.36	5.78	3.91	1.93	8.01	11.73	15.74

可以看出，当 $s = 6.5$ 时，$t(s)$ 具有最小值 1.93。此时，$R_{1,1} = \{1, 2, 3, 4, 5, 6\}$，$R_{1,2} = \{7, 8, 9, 10\}$，$c_{1,1} = 6.24$，$c_{1,2} = 8.91$。因此，得到最优回归树：

$$T_1^*(\boldsymbol{x}) = \begin{cases} 6.24 & x < 6.5 \\ 8.91 & x \geqslant 6.5 \end{cases}$$

4. 得到回归提升树：$f_1(\boldsymbol{x}) = T_1^*(\boldsymbol{x})$，其拟合训练数据的平方误差损失为：

$$L\left(y, f_1(\boldsymbol{x})\right) = \sum_{i=1}^{10} \left(y_i - f_1(\boldsymbol{x}_i)\right)^2 = 1.93$$

5. $m=2$，使用 $f_1(\boldsymbol{x})$ 拟合训练数据的残差为 $r_{2,i}=y_i-f_1(\boldsymbol{x}_i)(i=1,2,\ldots,10)$，列出如下：

$\boldsymbol{x}_i$	1	2	3	4	5	6	7	8	9	10
$r_{2,i}$	-0.68	-0.54	-0.33	0.16	0.56	0.81	-0.01	-0.21	0.09	0.14

使用上述数据作为训练数据，黑箱优化回归树 $T_2^*(\boldsymbol{x})$(计算过程略)，得到：

$$T_2^*(\boldsymbol{x})=\begin{cases}-0.52 & x<3.5\\ 0.22 & x\geqslant 3.5\end{cases}$$

进一步地，得到回归提升树：

$$f_2(\boldsymbol{x})=f_1(\boldsymbol{x})+T_2^*(\boldsymbol{x})=\begin{cases}5.72 & x<3.5\\ 6.46 & 3.5\leqslant x<6.5\\ 9.13 & x\geqslant 6.5\end{cases}$$

它的平方误差损失为：

$$L\left(y,f_2(\boldsymbol{x})\right)=\sum_{i=1}^{10}\left(y_i-f_2(\boldsymbol{x}_i)\right)^2=0.79$$

6. $m=3$，得到 $T_3^*(\boldsymbol{x})$ 与平方损失：

$$T_3^*(\boldsymbol{x})=\begin{cases}0.15 & x<6.5\\ -0.22 & x\geqslant 6.5\end{cases}\qquad L\left(y,f_3(\boldsymbol{x})\right)=0.47$$

7. $m=4$，得到 $T_4^*(\boldsymbol{x})$ 与平方损失：

$$T_4^*(\boldsymbol{x})=\begin{cases}-0.16 & x<4.5\\ 0.11 & x\geqslant 4.5\end{cases}\qquad L\left(y,f_4(\boldsymbol{x})\right)=0.30$$

8. $m=5$，得到 $T_5^*(\boldsymbol{x})$ 与平方损失：

$$T_5^*(\boldsymbol{x})=\begin{cases}0.07 & x<6.5\\ -0.11 & x\geqslant 6.5\end{cases}\qquad L\left(y,f_5(\boldsymbol{x})\right)=0.23$$

9. $m=6$，得到 $T_6^*(\boldsymbol{x})$ 与平方损失：

$$T_6^*(\boldsymbol{x})=\begin{cases}-0.15 & x<2.5\\ 0.04 & x\geqslant 2.5\end{cases}\qquad L\left(y,f_6(\boldsymbol{x})\right)=0.17$$

此时，如果停止训练，则可以得到最终的回归提升树：

$$f_6(\boldsymbol{x}) = f_5(\boldsymbol{x}) + T_6^*(\boldsymbol{x}) = T_1^*(\boldsymbol{x}) + T_2^*(\boldsymbol{x}) + T_3^*(\boldsymbol{x}) + T_4^*(\boldsymbol{x}) + T_5^*(\boldsymbol{x}) + T_6^*(\boldsymbol{x})$$

$$= \begin{cases} 5.63 & x < 2.5 \\ 5.82 & 2.5 \leqslant x < 3.5 \\ 6.56 & 3.5 \leqslant x < 4.5 \\ 6.83 & 4.5 \leqslant x < 6.5 \\ 8.95 & x \geqslant 6.5 \end{cases}$$

□

4 梯度提升树

前面使用的损失函数，例如平方误差损失函数与指数损失函数等，都有比较好的性质，且提升模型的优化过程也简洁明了。但是，对于一般的损失函数而言，问题可能会变得要复杂些。

针对这一问题，Freidman 提出了一种被称为“梯度提升”(Gradient Boosting) 的算法。这是一种最速下降法的近似方法。它的基本思想是，利用损失函数的负梯度值，作为回归提升树中的残差近似值或目标值：

$$r_{m,i} = -\frac{\partial L\left(y_i, f\left(\boldsymbol{x}_i\right)\right)}{\partial f(\boldsymbol{x}_i)} = -\frac{\partial L\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right)\right)}{\partial f_{m-1}(\boldsymbol{x}_i)} \tag{48}$$

其中，$f(\boldsymbol{x})$ 表示当前的回归提升树，即第 $m-1$ 轮结束后得到的回归提升树 $f_{m-1}(\boldsymbol{x})$。

实际上，梯度提升算法将回归提升树的残差 $r_{m,i} = y_i - f_{m-1}(\boldsymbol{x}_i)$ 进行了泛化，即将其扩展到了一般的损失函数情形。例如，如果 $L\left(y_i, f\left(\boldsymbol{x}_i\right)\right) = \frac{1}{2}\left(y_i - f_{m-1}\left(\boldsymbol{x}_i\right)\right)^2$，那么[9]：

$$r_{m,i} = -\frac{\partial L\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right)\right)}{\partial f_{m-1}(\boldsymbol{x}_i)} = y_i - f_{m-1}(\boldsymbol{x}_i) \tag{49}$$

因此，回归提升树中的残差 $r_{m,i}$ 也是一种负梯度。

下面，给出梯度提升树算法[10]。

[9]即平方误差损失函数，与前述的平方误差损失函数只相差常数因子，不影响优化结果。

[10]从本质上讲，算法3.1是本算法的特例。此处列出的梯度提升树算法，要更详细些，且模糊了回归提升树中回归树的黑箱优化策略。

算法 **4.1**（梯度提升树算法）

Input:

Dataset: $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Numbers of Regression Tree: M

Regression Tree: $T_m(\boldsymbol{x})$

Loss Function: $L\left(y, f\left(\boldsymbol{x}\right)\right)$

Output:

Gradient Boosting Regression Tree:

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} T_m^*(\boldsymbol{x}) = \sum_{m=1}^{M} \sum_{j=1}^{J} c_{m,j} I(\boldsymbol{x}_i \in R_{m,j})$$

Algorithm:

1. Initialize $f_0(\boldsymbol{x}) = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$
2. for $m = 1, 2, \ldots, M$:
3. $r_{m,i} == -\frac{\partial L(y_i, f_{m-1}(\boldsymbol{x}_i))}{\partial f_{m-1}(\boldsymbol{x}_i)}$, $i = 1, 2, \ldots, N$
4. Black Box Optimization for:
5. $T_m^*\left(\boldsymbol{x}\right) = \arg\min_{T_m} \sum_{i=1}^{N} \left(r_{m,i} - T_m\left(\boldsymbol{x}\right)\right)^2$
6. And:
7. Get Leaf Regions: $R_{m,j}$, $j = 1, 2 \ldots, J$
8. $c_{m,j} = \arg\min_c \sum_{\boldsymbol{x}_i \in R_{m,j}} L\left(y_i, f_{m-1}\left(\boldsymbol{x}_i\right) + c\right)$, $j = 1, 2, \ldots, J$
9. $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + T_m^*\left(\boldsymbol{x}\right) = f_{m-1}(\boldsymbol{x}) + \sum_{j=1}^{J} c_{m,j} I(\boldsymbol{x}_i \in R_{m,j})$

算法的第1步，初始化梯度提升树，估计使得初始损失函数极小化的常数值。此时，梯度提升树是一棵只有根节点的树。算法的第8步，利用一维线搜索，估计叶子节点 (区域) 的最优拟合值，使得损失函数极小化。

5 提升方法实验

AdaBoost 算法的实现

```
In [1]: import numpy as np
        from collections import Counter
        import copy
```

```
In [2]: class AdaBoost:
            def __init__(self, base):
                self.base = base # 弱分类器

            def fit(self, X, Y, max_step):
                N = len(X)
                self.W = np.ones(N) / N # 每个样本的权值初始化
                self.alpha = [] # 存放弱分类器权重
                self.weaker = [] # 存放弱分类器
                for step in range(max_step): # 训练弱分类器
                    weaker = copy.deepcopy(self.base) # 生成一个新弱分类器
                    # 使用弱分类器带样本权值进行训练
                    weaker.fit(X, Y, sample_weight = self.W)
                    results = weaker.predict(X)
                    # 输出准确度（Accuracy）
                    scores = (results == Y)
                    print('Weaker {:} accuracy = {:3.2f}%'.format(step,
                     Counter(scores)[True]/len(Y) * 100))
                    # 计算当前弱分类器的分类误差率：带权值误差
                    error = np.dot(self.W, [0 if score else 1 for score in scores])
                    alpha = 0.5 * np.log((1 - error)/error) # 计算当前弱分类器的权重
                    self.alpha.append(alpha)
                    # 更新每个样本的权值：分类正确-yG(x)=-1，否则 1
                    self.W = self.W * np.exp(alpha * np.array([-1 if score else 1
                     for score in scores]))
                    self.W = self.W / sum(self.W) # 归一化
                    self.weaker.append(weaker)
                # 计算最终分类器的结果
                f = np.dot(np.c_[[weaker.predict(X) for weaker in self.weaker]].T,
                 np.array(self.alpha).reshape(-1, 1))
                results = [1 if y >= 0 else -1 for y in f]
                scores = (results == Y)
                print('\nAdaBoost accuracy = {:3.2f}%, training completed!'.format(
                 Counter(scores)[True]/len(Y) * 100))

            def predict(self, X):
                f = np.dot(np.c_[[weaker.predict(X) for weaker in self.weaker]].T,
                 np.array(self.alpha).reshape(-1, 1))
                return [1 if y >= 0 else -1 for y in f]
```

例 8.1 的程序验证

构建例 8.1 的数据集

```
In [3]: X = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
        Y = [1, 1, 1, -1, -1, -1, 1, 1, 1, -1]
```

构造一个简易的弱分类器

```
In [4]: import operator
```

```
In [5]: class Simple: # 最简决策树桩, 仅用于验证例 8.1
            def fit(self, X, Y, sample_weight):
                minloss = float('inf')
                for compare in [operator.le, operator.ge]:
                    for threshold in X:
                        results = compare(np.array(X), threshold)
                        results = [1 if result else -1 for result in results]
                        results = results == Y
                        results = [0 if result else 1 for result in results]
                        loss = np.dot(results, sample_weight)
                        if minloss > loss:
                            minloss = loss
                            self.threshold = threshold
                            self.compare = compare
            def predict(self, X):
                return [1 if self.compare(x, self.threshold) else -1 for x in X]
```

```
In [6]: ada = AdaBoost(Simple())
        ada.fit(np.array(X), np.array(Y), 3)
        print(ada.alpha)
```

```
Weaker 0 accuracy = 70.00%
Weaker 1 accuracy = 70.00%
Weaker 2 accuracy = 60.00%

AdaBoost accuracy = 100.00%, training completed!
[0.42364893019360172, 0.64964149206513044, 0.75203869838813697]
```

使用 sklearn 的决策树作为决策树桩

```
In [7]: from sklearn.tree import DecisionTreeClassifier
        ada = AdaBoost(DecisionTreeClassifier(max_depth = 1))
        ada.fit(np.array(X).reshape(-1, 1), np.array(Y), 3)
        print(ada.alpha)
```

```
Weaker 0 accuracy = 70.00%
Weaker 1 accuracy = 70.00%
Weaker 2 accuracy = 60.00%

AdaBoost accuracy = 100.00%, training completed!
[0.42364893019360172, 0.64964149206513044, 0.75203869838813697]
```

使用鸢尾花数据集进行测试，载入预存的鸢尾花数据集

```
In [8]: from sklearn.model_selection import train_test_split
        import matplotlib.pyplot as plt
        %matplotlib inline

In [9]: iris_npz = np.load('iris.npz')
        data = iris_npz['data']
        X = iris_npz['X']
        Y = iris_npz['Y']
        # 转换成适合 AdaBoost 处理的标签
        Y[:50] = 1
        Y[50:] = -1

In [10]: XTRAIN, XTEST, YTRAIN, YTEST = train_test_split(X, Y, test_size = 0.25)

In [11]: import copy

         # 使用决策树桩作为弱分类器
         ada = AdaBoost(DecisionTreeClassifier(max_depth = 1))
         ada.fit(XTRAIN, YTRAIN, 10)
```

```
Weaker 0 accuracy = 90.67%
Weaker 1 accuracy = 84.00%
Weaker 2 accuracy = 77.33%
Weaker 3 accuracy = 64.00%
Weaker 4 accuracy = 74.67%
Weaker 5 accuracy = 70.67%
Weaker 6 accuracy = 86.67%
Weaker 7 accuracy = 80.00%
Weaker 8 accuracy = 77.33%
Weaker 9 accuracy = 74.67%

AdaBoost accuracy = 100.00%, training completed!
```

使用鸢尾花测试数据集进行测试

```
In [12]: results = ada.predict(XTEST)
         scores = (results == YTEST)
         print('Accuracy = {:3.2f}%'.format(Counter(scores)[True]/len(YTEST) * 100))
```

```
Accuracy = 100.00%
```

使用 sklearn.ensemble.AdaBoostClassifier

- algorithm：该参数只对 AdaBoostClassifier 有效——scikit-learn 实现了两种 Adaboost 分类算法，即 SAMME 和 SAMME.R。两者的主要区别是弱学习器权重的度量，前者使用分类效

果作为弱学习器的权重；后者使用分类预测概率作为弱学习器的权重，迭代一般比前者快。AdaBoostClassifier 的默认算法是 SAMME.R。需要注意的是，如果使用 SAMME.R，则弱分类学习器的参数 base_estimator 必须限制为支持概率预测的分类器，而 SAMME 算法没有此限制；

- n_estimators：该参数在 AdaBoostClassifier 和 AdaBoostRegressor 中都有效，它是弱学习器的最大迭代次数或弱学习器的最大数目。一般来说，n_estimators 太小，容易欠拟合；n_estimators 太大，又容易过拟合。一般选择一个适中的数值，默认是 50。在实际调参的过程中，常常将 n_estimators 和下面介绍的参数 learning_rate 一起考虑；
- learning_rate: 该参数在 AdaBoostClassifier 和 AdaBoostRegressor 中都有效，表示每个弱学习器的权重衰减系数；
- base_estimator：该参数在 AdaBoostClassifier 和 AdaBoostRegressor 中都有效，表示弱分类学习器或者弱回归学习器。理论上，可以选择任何一个分类或者回归学习器，不过需要支持样本权重。常用学习器可以是 CART 决策树或者神经网络 MLP；

```
In [13]: from sklearn.ensemble import AdaBoostClassifier
         clf = AdaBoostClassifier(n_estimators = 10)
         clf.fit(XTRAIN, YTRAIN)
```

```
Out[13]: AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
                    learning_rate=1.0, n_estimators=10, random_state=None)
```

```
In [14]: clf.score(XTEST, YTEST)
```

```
Out[14]: 1.0
```

回归提升树的实现

```
In [15]: class BoostTree:
             def __init__(self, base):
                 self.base = base # 弱回归器

             def fit(self, X, Y, max_step):
                 self.weaker = [] # 存放弱回归器
                 R = copy.deepcopy(Y) # 残差
                 for step in range(max_step): # 训练弱回归器
                     weaker = copy.deepcopy(self.base) # 生成一个新弱回归器
                     weaker.fit(X, R) # 训练弱回归器
                     results = weaker.predict(X)
                     R = R - results # 计算残差
                     self.weaker.append(weaker)
                     # 计算误差
                     f = np.sum(np.c_[[weaker.predict(X) for weaker in self.weaker]].T,
                      axis = 1)
                     print('Step {}, square loss {}'.format(step,
                      np.linalg.norm(f - Y)**2))
                 print('\nBoosting tree training completed!')

             def predict(self, X):
                 return np.sum(np.c_[[weaker.predict(X) for weaker in self.weaker]].T,
                  axis = 1)
```

验证例 8.2
构建训练数据集

```
In [16]: X = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
         Y = [5.56, 5.70, 5.91, 6.40, 6.80, 7.05, 8.90, 8.70, 9.00, 9.05]
         XTEST = [1.2, 2.3, 3.4, 4.5, 5.6, 6.7, 7.8, 8.9, 9.5, 10.8]

In [17]: from sklearn.tree import DecisionTreeRegressor
         # 使用回归决策树桩作为弱回归器
         bt = BoostTree(DecisionTreeRegressor(max_depth = 1))
         bt.fit(np.array(X).reshape(-1, 1), Y, 6)

Step 0, square loss 1.9300083333333338
Step 1, square loss 0.800675
Step 2, square loss 0.4780083333333336
Step 3, square loss 0.3055592592592599
Step 4, square loss 0.22891522633744946
Step 5, square loss 0.1721780649862837

Boosting tree training completed!


In [18]: bt.predict(np.array(XTEST).reshape(-1, 1))

Out[18]: array([ 5.63      ,  5.63      ,  5.81831019,  6.55164352,  6.81969907,
                 8.95016204,  8.95016204,  8.95016204,  8.95016204,  8.95016204])
```

使用 sklearn.ensemble.AdaBoostRegressor

```
In [19]: from sklearn.ensemble import AdaBoostRegressor
         rsr = AdaBoostRegressor(loss = 'square', n_estimators = 6)
         rsr.fit(np.array(X).reshape(-1, 1), Y)

Out[19]: AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='square',
                  n_estimators=6, random_state=None)

In [20]: print('Final square loss {}'.format(np.linalg.norm(rsr.predict(
 np.array(X).reshape(-1, 1)) - Y)**2))

Final square loss 0.04490000000000047


In [21]: rsr.predict(np.array(XTEST).reshape(-1, 1))

Out[21]: array([ 5.63,  5.7 ,  5.91,  6.4 ,  7.05,  8.9 ,  8.9 ,  9.  ,  9.  ,
                 9.05])
```

使用 sklearn.ensemble.GradientBoostingRegressor

```
In [22]: from sklearn.ensemble import GradientBoostingRegressor
         rsr = GradientBoostingRegressor(loss = 'ls', n_estimators = 6)
         rsr.fit(np.array(X).reshape(-1, 1), Y)
```

```
Out[22]: GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                     learning_rate=0.1, loss='ls', max_depth=3, max_features=None,
                     max_leaf_nodes=None, min_impurity_decrease=0.0,
                     min_impurity_split=None, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=6, presort='auto', random_state=None,
                     subsample=1.0, verbose=0, warm_start=False)

In [23]: print('Final square loss {}'.format(np.linalg.norm(rsr.predict(
 np.array(X).reshape(-1, 1)) - Y)**2))

Final square loss 5.424059630005812


In [24]: rsr.predict(np.array(XTEST).reshape(-1, 1))

Out[24]: array([ 6.50762479,  6.54599735,  6.64125405,  6.89189235,  7.14105909,
                 8.05341449,  7.95970269,  8.10027039,  8.10027039,  8.12369834])
```

6 参考文献

1. 李航。《统计学习方法》，清华大学出版社，2019 年 5 月第 2 版。

2. 周志华。《机器学习》，清华大学出版社，2016 年 1 月第 1 版。

3. Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

4. Kevin P. Murphy. Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.

5. https://zhuanlan.zhihu.com/p/59121403.