集成学习

IN : 机器学习 (HTTPS://LUMINGDONG.CN/CATEGORY/TECH/ML) 2018-11-28

3 评论 (HTTPS://LUMINGDONG.CN/ENSEMBLE-LEARNING.HTML#COMMENTS) 3 22 千字

56 分钟

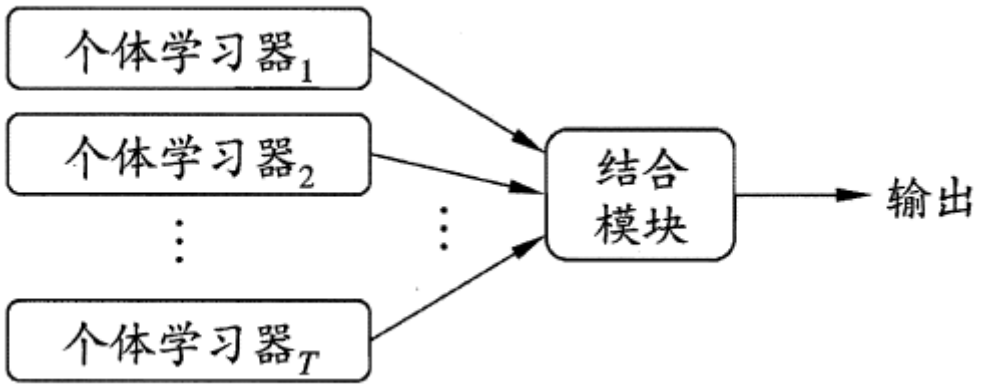
目录

1. 集成学习介绍

集成学习（Ensemble Learning）可以说是机器学习兵器谱上排名第一的 “屠龙刀”，是一个非常万能且有效的强大工具。这把 “屠龙刀” 在各大机器学习竞赛中被广泛使用，曾多次斩下桂冠。集成学习是用多个弱学习器构成一个强学习器，其哲学思想是 “三个臭皮匠赛过诸葛亮”，有时也被称为多分类器系统（multi-classifier system）、基于委员会的学习（committee-based learning）、元算法（meta-algorithm）等。

集成方法的研究点集中在使用什么模型以及这些模型怎么被组合起来。

一般的弱学习器可以由 Logistic 回归，决策树，SVM，神经网络，贝叶斯分类器，K-近邻等构成。集成中的个体学习器可以是相同的算法，可称为同质集成（Homogeneous Ensemble），其中的个体学习器称为 “基学习器”（base learner），相应的学习算法称为 “基学习算法”（base learning algorithm）；集成也可以包含不同类型的算法，可称为异质集成（heterogenous），个体学习器一般称为 “组件学习器” 或直接称为个体学习器。

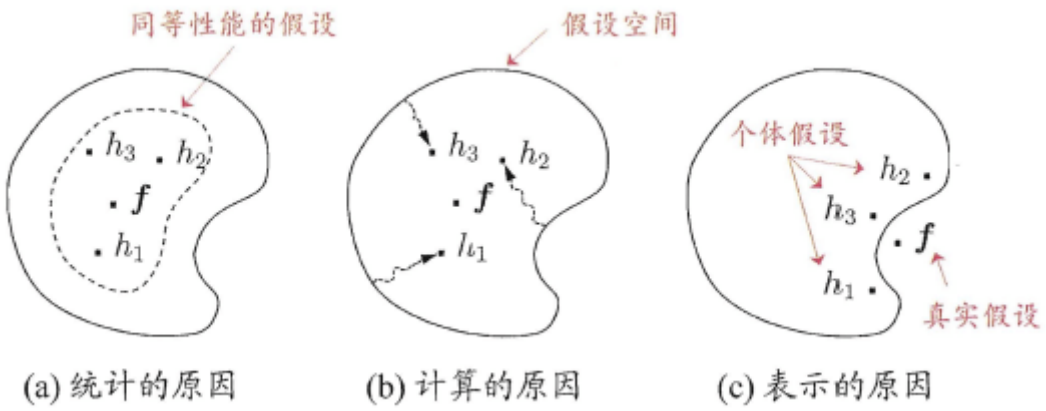


(https://lumingdong.cn/wp-content/uploads/2018/12/1541991000579.png)

> 弱学习器常指泛化性能略优于随机猜测的学习器，例如在二分类问题上精度略高于 50% 的分类器。虽然从理论上来说使用弱学习器集成足以获得好的性能，但在实践中出于种种考虑，例如希望使用较少的个体学习器，或是重用关于常见学习的一些经验等，人们往往会使用比较强的学习器。

$ 集成学习为什么可以提高学习器的性能?

集成学习可能会从三个方面带来好处 [Dietterich， 2000] : 首先从统计的方面来看，由于学习任务的假设空间往往很大，可能有多个假设在训练集上达到同等性能， 此时若使用单学习器可能因误选而导致泛化性能不佳，结合多个学习器则会减小这一风险；第二，从计算的方面来看，学习算法往往会陷入局部极小? 有的局部极小点所对应的泛化性能可能很糟糕， 而通过多次运行之后进行结合， 可降低陷入糟糕局部极小点的风险；第三， 从表示的方面来看，某些学习任务的真实假设可能不在当前学习算法所考虑的假设空间中，此时若使用单学习器则肯定无效，而通过结合多个学习器， 由于相应的假设空间有所扩大，有可能学得更好的近似，下图给出了一个直观示意图。



(https://lumingdong.cn/wp-content/uploads/2018/12/1541990221245.png)

个体学习器之间的整合方式，一般有平均法，简单投票，贝叶斯投票，基于 D-S 证据理论的整合，基于不同的特征子集的整合。

在一般经验中，如果把好坏不等的东西掺到一起，那么通常结果会是比最坏的要好一些，比最好的要坏一些，集成学习把多个学习器结合起来，**要想获得比最好的单一学习器更好的性能，个体学习器应“好而不同”**，即个体学习器要有一定的“准确性”，即学习器不能太坏，并且要与“多样性”，即学习器间具有差异。然而，我们在集成过程中都会有一个关键假设：基学习器的误差相互独立。但是在现实任务中，个体学习器是为解决同一个问题训练出来的，它们显然不可能相互独立！事实上，个体学习器的“准确性”和“多样性”本身就存在冲突，一般的，准确性很高的时候，要增加多样性就需牺牲准确性，事实上，如何产生并结合“好而不同”的个体学习器，恰是集成学习研究的核心。[1]

根据个体学习器的生成方式，目前的集成学习方法大致可分为两大类，即个体学习器间不存在强依赖关系、可同时生成的并行化方法，代表是 Bagging；以及个体学习器问存在强依赖关系、必须串行生成的序列化方法，代表是 Boosting 。

2. Bagging

Bagging 是 Bootstrap Aggregation 的缩写，直译为自助汇聚法，其中 Bootstrap 是指一种有放回式的自助采样法（bootstrap sampling）。Bagging 是并行集成学习方法的典型代表，著名的集成学习算法随机森林其实是 Bagging 的一个扩展变体。

2.1. Bagging

Bagging［Breiman, 1996a］ 是为了得到泛化能力强的集成，因而就需要让各个子学习器之间尽可能独立，但是如果将样本分为了不同的不重合子集，那么每个基学习器学习的样本就会不足。所以它采用一种自助采样的方法（boostrap sampling），生成互相有交叠的采样子集。然后对每个采样子集分别进行训练，产生多个具有比较大差异的基学习器，最后使用投票法或平均法获得最终的预测结果。

2.1.1. Bagging算法过程

1）自助采样

给定包含 m 个样本的数据集，我们先随机取出一个样本放入采样集中，再把该样本放回初始数据集，使得下次采样时该样本仍有可能被选中，这样，经过 m 次随机采样操作，我们得到含 m 个样本的采样集，初始训练集中有的样本在采样集里多次出现，有的则从未出现，初始训练集中约有 63.2% 的样本出现在来样集中。

袋外数据（Out-Of-Bag，OOB）

对于自助采样法，我们可以做一个简单的估计，样本再 m 次采样中始终不被采到的概率是 $(1-\frac{1}{m})^m$，取极限得到

$$\lim_{m\to\infty}\left(1-\frac{1}{m}\right)^m=\frac{1}{e}\approx 0.368$$

因此，每次约有 36.8% 的样本未出现在采样数据集中，将未参与模型训练的数据称为袋外数据（西瓜书中是 “包外样本”），它可以用于取代验证测试集用于误差估计，称为袋外估计（out-of-bag，estimate）。Breiman 以经验性实例的形式证明袋外估计与同训练集一样大小的测试集精度相同，因此得到的模型参数是无偏估计。另外，如果基学习器是决策树时，可使用袋外数据来辅助剪枝，或用于估计决策树中各结点的后验概率以辅助对零训练样本结点的处理；当基学习器是神经网络时，可使用包外样本来辅助早期停止以减小过拟合风险。

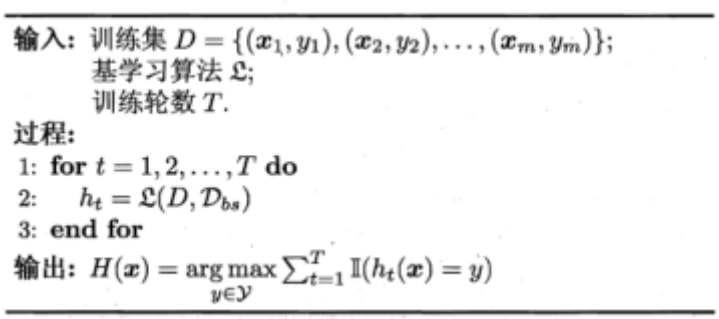
2）训练基学习器

照这样，我们可采样出 T 个含 m 个训练样本的采样集，然后基于每个采样集训练出一个基学习器，再将这些基学习器进行结合。这就是 Bagging 的基本流程。

3）集成

在对预测输出进行结合时，Bagging 通常对分类任务使用简单投票法，对回归任务使用简单平均法。若分类预测时出现两个类收到同样票数的情形，则最简单的做法是随机选择一个，也可进一步考察学习器投票的置信度来确定最终胜者。

整个过程可以总结为：



(https://lumingdong.cn/wp-content/uploads/2018/12/1542003994582.png)

其中，D_{bs} 是自助采样产生的样本分布。

2.1.2. Bagging的特点

1）从偏差-方差分解角度来看，Bagging 主要关注降低方差，因此它在不剪枝决策树、神经网络等易受样本扰动的学习器上效用更为明显；

2）Bagging 是一个非常高效的集成学习算法，算法复杂度大致为 $T(O(m)+O(s))$，其中 $O(m)$ 为基学习器的计算复杂度，$O(s)$ 是采样与投票/平均过程的复杂度，T 是训练轮数。一般情况下，$O(s)$ 的复杂度较小，T 也是一个不太大的常数，因此，训练一个 Bagging 集成与直接使用基学习算法训练一个学习器的复杂度同阶。

3）Bagging 能够不经修改地用于多分类、回归等任务，而 Adaboost 若不经修改的话，只适用于二分类任务。

2.2. 随机森林

随机森林（Random Forest， RF） ［Breiman, 2001a］ 是 Bagging 的一个扩展变体，RF 在以决策树为基学习器构建 Bagging 集成的基础上，进一步在决策树的训练过程中引入了随机属性选择。也就是说，随机森林不仅具有 Bagging 中**样本扰动**（初始训练集采样）的过程，还增加了一个**属性扰动**的过程，使得基学习器的 “多样性” 和 “差异度” 进一步提升，从而提高了最终集成的泛化性能。

2.2.1. 随机森林算法过程

1）自助采样

从样本集中用 Bootstrap 采样选出 n 个样本。

2）选择随机属性子集

传统决策树在选择划分属性时是在当前结点的属性集合（假定有 d 个属性）中选择一个最优属性，而在 RF 中，对基决策树的每个结点，先从该结点的属性集合中随机选择一个包含 k 个属性的子集，然后再从这个子集中选择一个最优属性用于划分。这里的参数 k 控制了随机性的引入程度：若令 $k=d$，则基决策树的构建与传统决策树相同；若令 $k=1$，则是随机选择一个属性用于划分； 一般情况下，推荐值 $k=\log_2 d$ ［Breiman, 2001a］。

3）建立 CART 决策树

选择最优属性后，用采样样本建立 CART 决策树。

4）重复

重复以上三步 m 次，即建立了 m 棵 CART 决策树。

5）集成

对预测输出进行结合，如分类任务使用多数投票法，对回归任务使用简单平均法。

$ 集成策略

1）平均法

a）简单平均法（simple averaging）

$$H(x) = \frac{1}{T}\sum_{i=1}^{T} h_i(x)$$

b）加权平均法（weighted averging）

$$H(x) = \sum_{i=1}^{T} w_i h_i(x)$$

其中，$w_i \geq 0,\ \sum_{i=1}^{T} w_i = 1$。

2）投票法

a）绝对多数投票法（majority voting）

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

即若某标记得票过半数，则预测为该标记，否则拒绝预测。

b）相对多数投票法（plurality voting）

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

即预测为得票最多的标记，若同时有多个标记获得最高票，则从中随机选取一个。

c）加权投票法（weighted voting）

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

与加权平均类法类似，其中，$w_i \geq 0$, $\sum_{i=1}^{T} w_i = 1$。

注：分类问题中，不同类型的个体学习器可能产生不同类型的 $h_i^j(x)$ 值，常见的有：

- 类标记：$h_i^j(x) \in \{0,1\}$，若 h_i 将样本 x 预测为类别 c_j 则取值为 1，否则为 0。使用类标记的投票亦称 “硬投票”（hard voting)。
- 类概率：$h_i^j \in [0,1]$，相当于对后验概率 $P(c_j|x)$ 的一个估计，使用类概率的投票亦称 “软投票”（soft voting)。

经验发现，虽然分类器估计出的类概率值一般都不太准确，但基于类概率进行集成却往往比直接基于类标记进行集成性能更好。不同类型的 h_i^j 值不能混用，若基学习器的类型不同（异质集成），则其概率值不能直接进行比较，这种情况下，通常可将类概率输出转化为类标记输出（例如将类概率输出最大的时 $h_i^j(x)$ 设为 1，其他设为 0）然后再投票。

3）学习法

当训练数据很多时，一种更为强大的结合策略是使用 “学习法”，即通过另一个学习器来进行结合。Stacking [Wolpert, 1992; Breiman, 1996b] 是学习法的典型代表，它本身又是一种著名的集成学习方法，与 Bagging 和 Boosting 最大的区别是它可以进行异质集成，即个体学习器可以是多种不同类型的算法。关于 Stacking，暂时不在本文介绍，Stacking 作为学习法实现集成策略，可参考周志华老师的西瓜书。

2.2.2. 随机森林的特点

1）随机森林简单易实现，计算开销小；

2）随机森林在实践应用中表现出强大的性能，被誉为 “代表集成学习技术水平的方法”；

3）随机森林的收敛性与 Bagging 相似，起始性能较差，随着个体学习器数目的增加，随机森林通常会收敛到更低的泛化误差。

4）随机森林训练效率常优于 Bagging，因为在个体决策树的构建过程中，Bagging 使用的是 “确定型” 决策树，在选择划分属性时要对结点的所有属性进行考察，而随机森林使用的 “随机型” 决策树则只需考察一个属性子集。

5）除了使用决策树作为个体学习器，随机森林还可以使用 SVM、Logistic 回归等其他学习器，习惯上，这些学习器组成的 “总学习器”，仍然叫作随机森林。

3. Boosting

Boosting 就是著名的提升方法，是一族可将弱学习器提升为强学习器的算法，可用于回归和分类问题。严格来讲提升并不是一种独立算法，而是在已有算法上进行优化的机器学习技术。**从偏差-方差分解的角度看， Boosting 主要关住降低偏差，因此 Boosting 能基于泛化性能相当弱的学习器构建出很强的集成，从理论上讲，如果一个问题存在弱学习器，则可以通过提升的方法得到强学习器。**

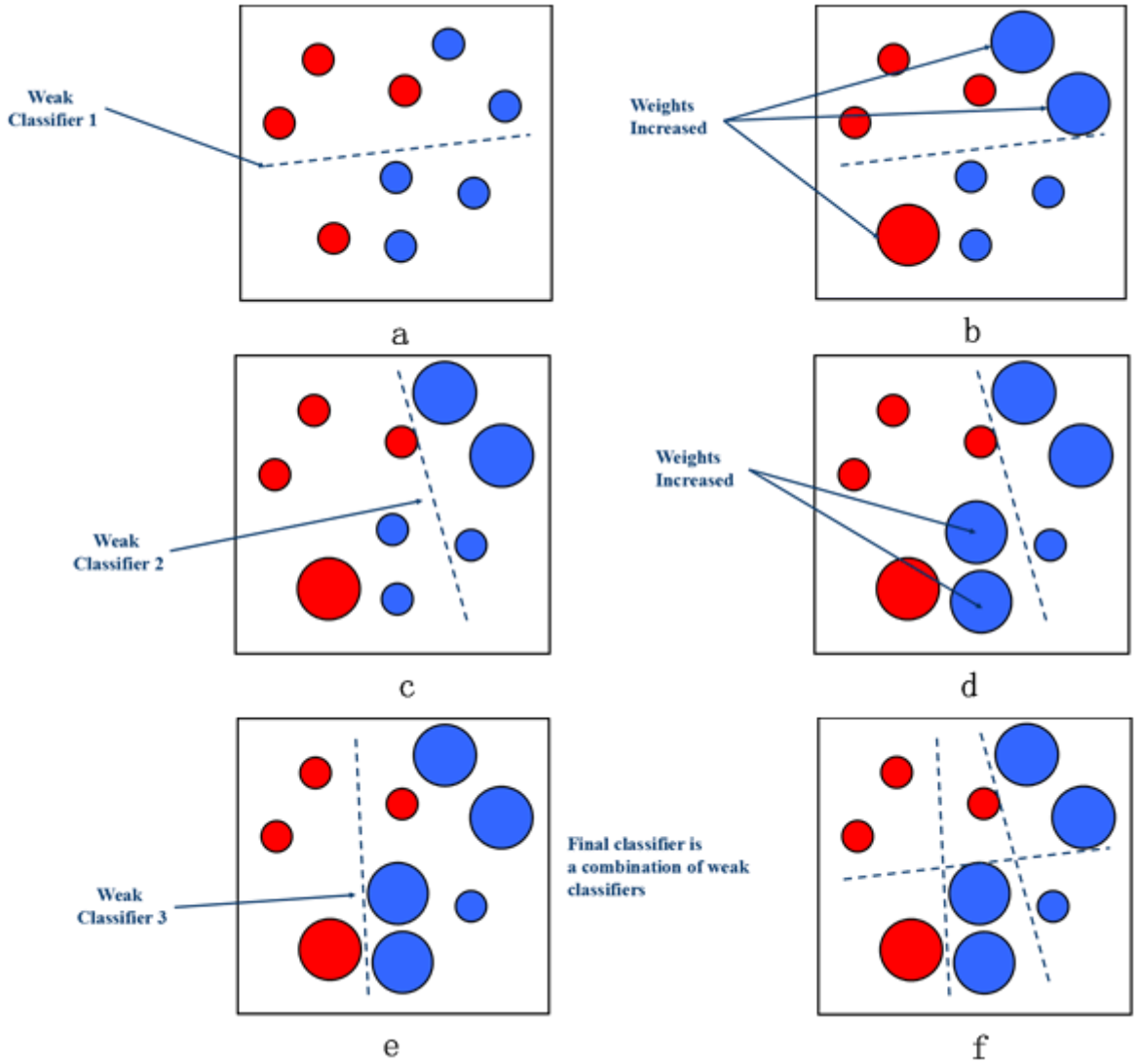
提升算法族的工作机制类似：先从初始训练集训练出一个基学习器，再根据基学习器的表现对训练样本分布进行调整，使得先前基学习器做错的训练样本在后续受到更多关注，然后基于调整后的样本分布来训练下一个基学习器；如此重复进行，直至基学习器数目达到事先指定的值 T ，最终将这 T 个基学习器进行加权结合（通常为线性加权）。

Boosting 算法族著名的代表有 AdaBoost [Freund and Schapire, 1997]、GBDT[Friedman, 1999] 以及 XGBoost[Tianqi Chen, 2016] 等。

3.1. Adaboost

Adaboost（Adaptive Boost）即自适应提升算法。对提升方法来说，有两个问题需要回答：一是在每一轮如何改变训练数据的权值或概率分布；二是如何将弱分类器组合成一个强分类器。关于第一个问题，AdaBoost 的做法是，提高那些被前一轮弱分类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的弱分类器的更大关注。于是，分类问题被一系列的弱分类器 “分而治之”。至于第二个问题，即弱分类器的组合，AdaBoost 采取加权多数表决的方法。具体地，加大分类误差率小的弱分类器的权值，使其在表决中起较大的作用，减小分类误差率大的弱分类器的权值，使其在表决中起较小的作用。AdaBoost 的巧妙之处就在于它将这些想法自然且有效地实现在一种算法里。

从下图（图片源自百度百科 Boosting 词条）可以看出，从弱分类器出发，每一次分类错误的样本都会增加权重，使得在下一次分类的时候更加关注这些错分样本，如此重复，最后将这些弱分类器组合，构成一个强分类器。这里需要注意的一点是，**我们最后需要的强分类器不是通过提升得到的最后一个分类器，而是结合了所有中间的弱分类器，最终构成的强分类器。**



(https://lumingdong.cn/wp-content/uploads/2018/12/caef76094b36acaf4872a57d76d98d1000e99ca4.png)

3.1.1. Adaboost算法过程

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，其中 $x_i \in X \subseteq \mathbf{R}^n$，$Y=\{-1,+1\}$；弱学习器算法；

输出：最终分类器 $G(x)$。[2]

1）初始化训练数据的权值分布 (直接用均值)

$$D_1=(w_{11},w_{12},\cdots,w_{1i},\cdots,w_{1N}),\ w_{1i}=\frac{1}{N},\ i=1,2,\cdots,N$$

2）对于 $m=1,2,\cdots,M$，M 为训练轮数。

a）使用具有权值分布 D_m 的训练数据集学习，得到基本分类器

$$D_m(x) : X \to \{-1, +1\}$$

b）计算 $G_m(x)$ 在训练数据集上的分类误差率

$$e_m = P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i)$$

c）计算 $G_m(x)$ 的系数

$$\alpha_m = \frac{1}{2} \ln \frac{1 - e_m}{e_m}$$

计算基分类器 $G_m(x)$ 的系数 α_m，当 $e_m \leq \frac{1}{2}$ 时，$\alpha_m \geq 0$，并且 α_m 随着 e_m 的减小而增大，所以分类误差率越小的基本分类器在最终分类器中的作用越大。

有时候，为了避免 Adaboost 过拟合，可以在基本分类器的系数算好后，再乘以一个学习率，如 0.1，保证让模型不要学得太快，在此基础上再去计算下一次基分类器系数。

d）更新训练数据集的权值分布

$$D_{m+1} = (w_{m+1,1}, \cdots, w_{m+1,i}, \cdots, w_{m+1,N})$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), i = 1, 2, \cdots, N$$

其中，Z_m 是规范化因子：

$$Z_m = \sum_{i=1}^{N} w_{mi} \exp(-\alpha_m y_i G_m(x_i))$$

它使 D_{m+1} 成为一个概率分布，作用相当于归一化，它的值等于所有分子的加和。

上式中：

y_i 是实际值，$G_m(x_i)$ 是预测值，对于二分类（+1/-1），如果预测错误，则二者必然有一个是负的，所以 $y_i G_m(x_i) < 0$；因为 α_m 是基本分类器的权值，正常来讲，α_m 是个正数，所以 $-\alpha_m y_i G_m(x_i) > 0$；又因为 exp(正数) > 1，$w_{mi}$ 乘以一个大于 1 的数，那更新后的值会变大，也就是说，如果预测错误，更新后的 $w_{m+1,i}$ 会变大；反之，预测正确权值降低。

3）构成基本分类器的线性组合

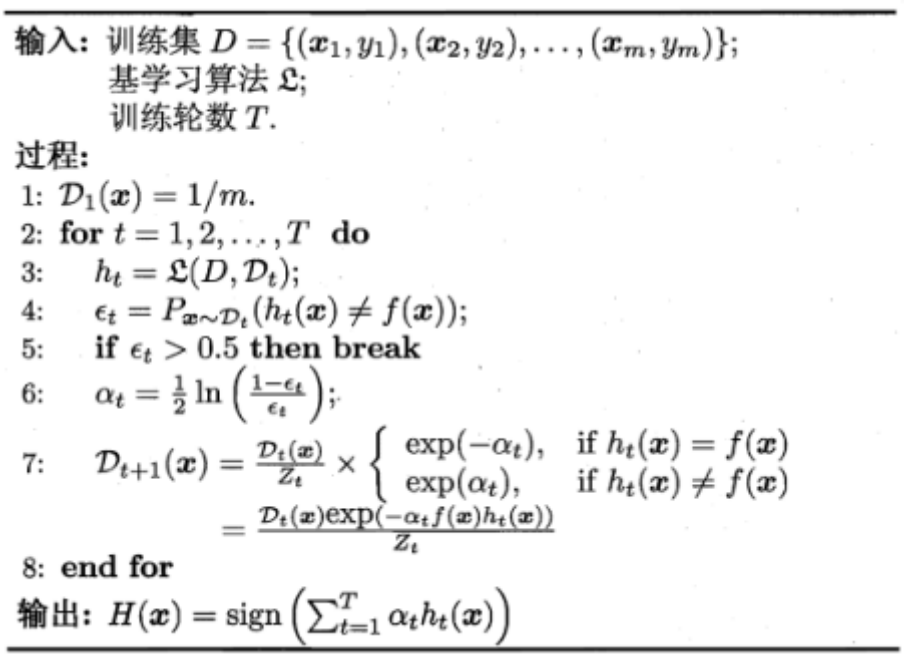
$$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$$

得到最终分类器：

$$G(x) = \text{sign}(f(x)) = \text{sign}(\sum_{m=1}^{M} \alpha_m G_m(x))$$

其中，**sign 是一个符号函数**，和之间介绍的 sigmod 函数作用类似，有些地方也写作 sgn，在这里的映射空间是 {-1，+1}，即 $\text{sign}(x)\begin{cases}1, & x>0\\ -1, & x\le 0\end{cases}$。

整个过程可以总结简写为：



(https://lumingdong.cn/wp-content/uploads/2018/12/1542357378125.png)

3.1.2. Adaboost 算法推导

Adaboost 算法过程步骤其实不难理解，关键在于训练更新过程中，有几处公式出现的莫名其妙，甚至还有指数和对数出现，这就难免会产生以下疑惑：

1）Adaboost 的损失函数是什么，为什么后来出现很多带有指数或对数形式的公式？

2）基分类器的系数更新公式 α_m 虽然符合随着误差率的减小而系数越大的基本要求，但为什么要写成这种对数形式的？

3）为什么样本权值的更新公式写那么复杂，还有很多指数形式的公式?

其实上面的问题最关键的原因是 **Adaboost 使用的是指数损失函数**，后面的基分类器系数和样本分布更新公式都是按照这个指数损失函数推导而来的，所以会出现自然指数和自然对数。

实际上二分类问题最常用的应该是 0-1 损失函数，即当预测错误时，损失函数值为 1，预测正确时，损失函数值为 0。而 Adaboost 则使用指数损失函数作为 0-1 损失函数的替代损失函数，通过 “加性模型” 即基学习器的线性组合来最小化指数损失函数。之所以选择指数损失函数，是因为这个替代函数具有更好的数学性质，比如它是连续可微函数，便于后续的优化推导。替代损失函数是否可以替代，需要证明这个函数与原损失函数的 “一致性”，这在周志华老师的西瓜书中有非常详细的证明过程，最终的结果能够达到贝叶斯最优错误率，能够证明指数损失函数最小化，则分类错误率也能最小化，满足经验风险最小化的一致性充要条件，指数损失函数可以作为 0-1 函数的替代函数。

因此，后面的基学习器和样本分布更新公式都是为了最小化指数损失函数推导得到的，并不是 “莫名其妙” 而来，在西瓜书中有非常详细的证明推导过程，这里不再复述。

Adaboost 算法之重采样法

Boosting 要求基学习器能对特定的数据分布进行学习，则可通过上面提到**重赋权法（re-weighting）**实施，即在训练过程的每一轮中，根据样本分布为每个训练样本重新赋予一个权重。然而**有些基学习器无法接受带权样本**，则可通过**重采样法（re-sampling）**来处理，即在每一轮学习中，根据样本分布对训练集重新进行采样，再用重采样而得的样本集对基学习器进行训练。一般而言，这两种做法没有显著的优劣差别，不过在一些特殊情况中，重采样法还有一些额外的好处。Boosting 算法在训练的每一轮都要检查当前生成的基学习器是否满足基本条件（例如检查当前基分类器是否是比随机猜测好），一旦条件不满足，则当前基学习器即被抛弃，且学习过程停止。在此种情形下，初始设置的学习轮数 T 也许遥远未达到，可能导致最终集成中只包含很少的基学习器而性能不佳。**若采用重采样法，则可获得 “重启动” 机会以避免训练过程过早停止 [Kohavi and Wolpert, 1996]**，即在抛弃不满足条件的当前基学习器之后，可根据当前分布重新对训练样本进行采样，再基于新的采样结果重新训练出基学习器，从而使得学习过程可以持续到预设的 T 轮完成。[1]

3.1.3. Adaboost算法特点

1）Adaboost 算法可以认为是模型是加法模型、损失函数为指数损失函数、学习算法为前向分步算法的二类分类学习方法；

前向分步算法的优化思路是：因为学习的是加法模型，如果能够从前向后，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，那么就可以简化优化的复杂度。比如在 Adaboost 中，前向分步算法将同时求解从 $m = 1$ 到 M 所有基本分类器 $G_m(x)$ 及其系数 α_m 的优化

问题简化为逐一学习基本分类器及其系数的过程，Adaboost 算法是前向分步加法算法的特例。[2]

2）Adaboost 的训练误差是以指数速率下降的；

3）Adaboost 算法的样本权重和基分类器的权重是交叉更新的；

4）Adaboost 算法不需要事先知道每个分类器误差下降的是多少，具有自适应性（Adaptive），它能自适应分类器的训练误差率。

5）Adaboost 算法可以使用 Logistic 回归，决策树，SVM，神经网络，贝叶斯分类器等作为基分类器；

6）Adaboost 最初的算法过程仅适用于二分类的分类任务，对于处理多分类或回归任务的情况，需要对 Adaboost 算法过程进行修改，目前已经有适用的变体算法。[1]

3.2. GBDT

GBDT（Gradient Boosting Decision Tree）称为梯度提升树，也是 Boosting 族算法之一，在传统机器学习算法里面是对真实分布拟合的最好的几种算法之一。GBDT 有很多简称，有 GBT（Gradient Boosting Tree）、GTB（Gradient Tree Boosting ）、 GBRT（Gradient Boosting Regression Tree）、 MART（Multiple Additive Regression Tree），通常是指的同一种算法，本文统一简称 GBDT。

GBDT 是从提升树而来，可以说是提升树的升级版本，我们先从提升树说起。

3.2.1. 提升树与梯度提升树

提升树（Boosting Tree）是以决策树为基学习器（基函数）的提升方法，个体决策树通常都会使用 CART 决策树生成算法，对分类问题是二叉决策树，对回归问题是二叉回归树。提升树模型可以表示为决策树的加法模型：

$$f(x) = \sum_{m=1}^{M} \alpha_m T(x;\theta_m)$$

其中，$T(x;\theta_m)$ 表示个体决策树，α_m 是每棵树的权重，θ_m 为决策树的参数，M 为树的个数。注意，这里的 α_m 应该只有在二分类问题中退化成 Adaboost 才会用到，回归提升树以及 GBDT 中都是通过残差或者伪残差来 “度量” 与真实样本之间的差距，个人觉得残差和伪残差已经具有了基学习器权重类似的作用，可通过拟合残差和伪残差来实现对基本决策树的关注度的影响和调节。

对于二类分类问题，提升树算法只需将 AdaBoost 基本分类器限制为二类分类树即可，可以说这时的提升树算法是 AdaBoost 算法的特殊情况。但是回归问题就不能沿用 Adaboost 的步骤了，我们需要进行修改。

已知一个训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，$x_i\in X\subseteq \mathbf{R}^n$，$X$ 为输入空间，$y_i\in Y\subseteq \mathbf{R}$，$Y$ 为输出空间。如果将输入空间 X 划分为 J 个互不相交的区域 $R_1,R_2,\cdots,R_J$，并且在每个区域上确定输出的常量 c_j，那么树可以表示为：

$$T(x;\theta)=\sum_{j=1}^{J}c_jI(x\in R_j)$$

其中，参数 $\theta=\{(R_1,c_1),(R_2,c_2),\cdots,(R_3,c_J)\}$ 表示树的区域划分和各区域上的常数，J 是回归树的复杂度即叶结点个数。

按照 Boosting 的前向分步算法，第 m 步基学习器是对第 m-1 步基学习器的进一步优化，前面已经学习到了一些基学习器，但是线性加和后与期望值之间依然有些误差，Boosting 的做法就是通过继续更新学习新的基学习器，用新的基学习器来逐步 “弥补” 误差，直至这个误差消失或者非常小。

所以，它们之间存在下面的关系：

$$F_m(x)=F_{m-1}(x)+T(x;\theta_m),\ \ m=1,2,\cdots,M$$

其中，M 为训练轮数，有些地方，这个树也会直接说成是函数 ，意思一样。

模型的初始化值可以用一个离所有样本观测值都比较近的常数，即输出空间所有训练样本的均值:

$$F_0(x)=\frac{\sum_{i=1}^{N}y_i}{N}$$

最终的模型可表示为：

$$F_M(x)=\sum_{m=1}^{M}T(x;\theta_m)$$

其中，α_m 为基学习器第 m 棵树的权重。

假设在前向分步算法的第 m 步，给定当前模型 $F_{m-1}(x)$，可求解

$$\theta_m^* = \arg\min_{\theta_m} \sum_{i=1}^{N} L(y_i, F_{m-1} + T(x_i; \theta_m))$$

得到 $\theta_m^* = \{(R_1, c_1), (R_2, c_2), \cdots, (R_3, c_J)\}$，即第 m 棵树的参数。

假定最优函数是 $F^*(x)$，代价函数是所有损失期望最小的函数：

$$F^*(x) = \arg\min_{F} E_{(x,y))}[L(y, F(x)]$$

当采用平方误差损失函数时，即 $L(y, F(x)) = (y - F(x))^2$，其损失变为：

$$\begin{aligned} L(y, f_{m-1}(x) + T(x; \theta_m)) &= [y - F_{m-1}(x) - T(x; \theta_m)]^2 \\ &= [r - T(x; \theta_m)]^2 \end{aligned}$$

其中，$r = y - F_{m-1}(x)$ 是当前模型拟合数据的**残差（residual）**。所以，对回归问题的提升树算法来说，只需简单地拟合当前模型的残差，不断更新决策树，直到获得最优模型。

Boosting 每一次更新目的就是让当前模型的预测值更加接近期望值，也就是使得当前 $F(x_i)$ 的预测结果与期望值 y_i 之间的差距越来越小，这个差距如果使用平方误差损失函数的话，其实就是残差。因此，我们可以将减小残差作为优化方向，每一次更新学到的新决策树都是为了减小残差，这其实就是 GBDT 的本质。

关于回归提升树对于残差的拟合，我们借助一个简单的例子来理解：

> 用提升树来预测年龄，假设我们的预测一共迭代 3 轮，真实年龄为 [5, 6, 7]。
>
> 第 1 轮预测：[6, 6, 6] (平均值)
>
> 第 1 轮残差：[-1, 0, 1]
>
> 第 2 轮预测：[6, 6, 6] (平均值) + [-0.9, 0, 0.9] (第 1 颗回归树) = [5.1, 6, 6.9]
>
> 第 2 轮残差：[-0.1, 0, 0.1]
>
> 第 3 轮预测：[6, 6, 6] (平均值) + [-0.9, 0, 0.9] (第 1 颗回归树) + [-0.08, 0, 0.07] (第 2 颗回归树) = [5.02, 6, 6.97]
>
> 第 3 轮残差：[-0.08, 0, 0.03]

可以看出残差不断减小，其实上面的过程就是 GBDT，只不过它是一个特例。

提升树利用加法模型与前向分步算法实现学习的优化过程，如果损失函数是平方损失和指数损失函数的话，那么每一步优化都是很简单的，但在实践应用中，在不同的场景下可能会用到不同的损失函数，对于一般损失函数而言，往往每一步优化并不那么容易，因此，通过最小化损失函数求解最优模型成为了一个 NP 难问题，这个时候，我们便可通过贪心法，借助梯度下降法，迭代求局部最优解。

利用一般损失函数的负梯度在当前模型的值仅仅是回归提升树算法中残差的近似值，我们称之为**伪残差（pseudo-residual）**，有时也称为**响应（response）**或负梯度误差，公式如下：

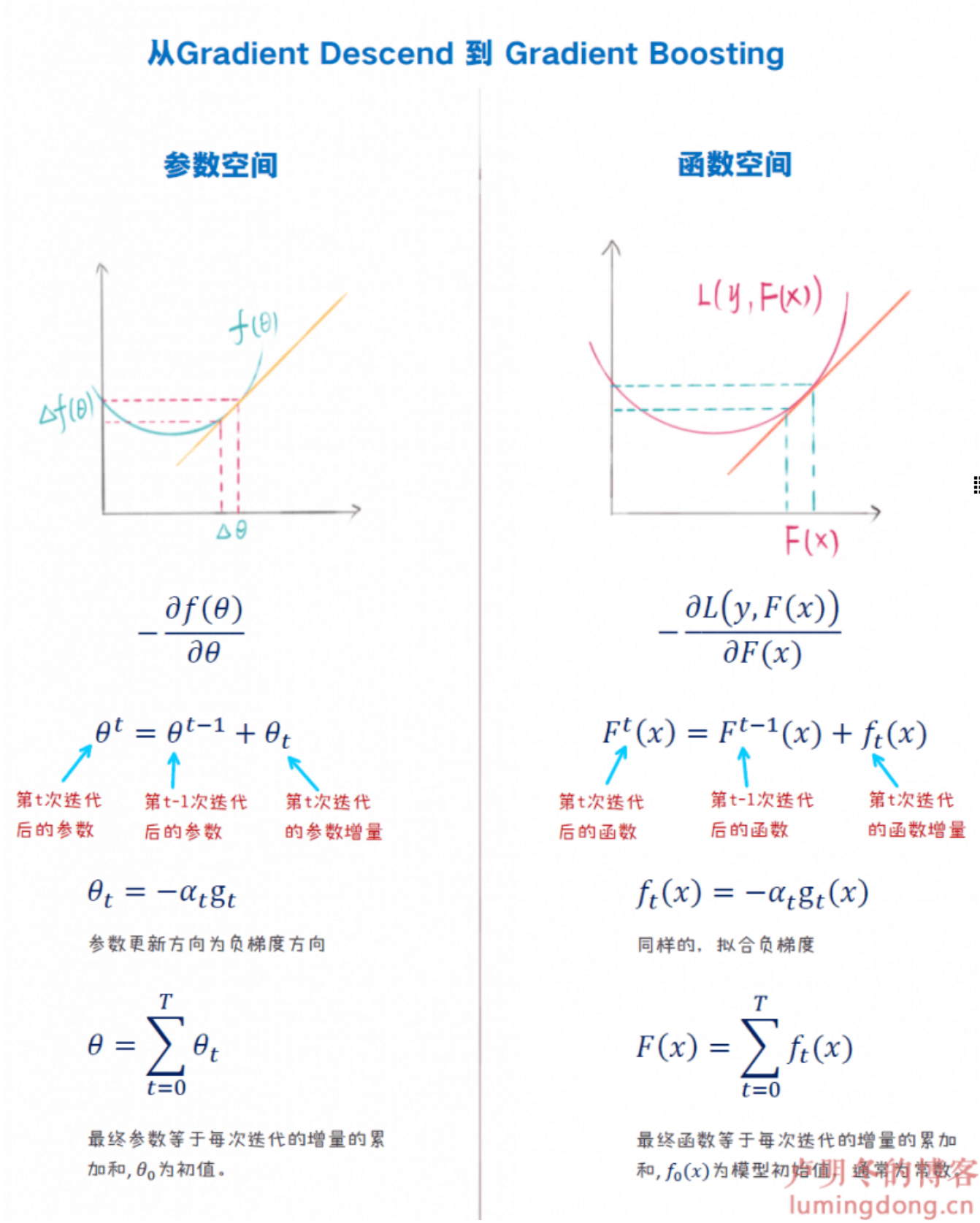
$$r_{mi} = -[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)}$$

如果损失函数为平方误差时，即 $L = \frac{1}{2}(y - F(x))^2$，对 F 求偏导可得：

$$r_i = -g_i = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

可以发现此处的响应结果正是回归提升树中所提到的残差，此时响应与残差相等，可以说残差是 GBDT 的一个特例情况。使用了梯度下降法的提升树就是梯度提升树。

GBDT 就是在函数空间的梯度下降，与梯度下降类似，我们不断减去 $\frac{\partial f(x)}{\partial x}$，可以得到 $\min_x f(x)$；同理不断减去 $\frac{\partial L}{\partial F}$，就能得到 $\min_F L(F)$。可参考下图：



(https://lumingdong.cn/wp-content/uploads/2018/11/函数空间.png)

再往深层一点，其实在《梯度下降算法总结》中，我们已经提到了**梯度下降法的原理。对于一些损失函数，我们无法直接求得其最优解，这个求解最优化参数的过程是一个 NP 难问题，既然无法直接求解，那我们可以利用贪心算法的思想，通过不断逼近最优解，来求得其近似值。**

而对于近似计算，泰勒公式是一个非常好的工具，因此，梯度下降法和牛顿法便被发明了出来，梯度下降法是泰勒公式的一阶展开，牛顿法是泰勒公式的二阶展开。同样的，提升树也可以用这两种优化方法，**GBDT 在函数空间中利用梯度下降法进行优化，而 XGBoost 在函数空间中用牛顿法进行优化** [3]，XGBoost 是陈天奇开发出的大赛大杀器，算是 GBDT 的一个变种，除了使用的是二阶展开的牛顿法进行优化，XGBoost 还使用了很多技巧，打算在之后与实践相结合，专门写一篇关于 XGboost 的文章。

3.2.2. GBDT算法过程

综上，我们可以整理一下 GBDT 的算法过程。

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，$x_i \in X \subseteq \mathbf{R}^n$，$y_i \in Y \subseteq \mathbf{R}$；损失函数 $L(y,F(x))$;

输出：回归树 $F^*(x)$。

1）初始化 F_0；

$ 几种常见的初始化方法

a）随机初始化；

b）用训练样本中的充分统计量初始化，如习惯上有：

- 如果损失函数是平方误差损失（服从正态分布），用样本的均值初始化
- 如果损失函数是绝对损失（服从拉普拉斯分布），用样本的中位数初始化

c）用其他函数的预测值初始化

GBDT 很健壮，对初始值并不敏感，但是更好的初始值能够获得更快的收敛速度和质量。

2）对 $m=1,2,\ldots,M$

a）对 $i=1,2,\ldots,N$，计算伪残差：

$$r_{mi}=-[\frac{\partial L(y_i,F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)}$$

b）利用 $(x_i,r_{mi})\ \ (i=1,2,..N)$， 拟合一棵 CART 回归树，得到第 m 棵回归树，其对应的叶子节点区域为 R_{mj}，$j=1,2,\ldots,J$。其中 J 为第 m 棵回归树的叶子节点的个数。

c）对叶子区域 $j=1,2,\cdots,J$，利用线性搜索（Line Search）计算最佳拟合值：

$$c_{mj} = \arg\min_{c} \sum_{x_i \in R_{mj}} L(y_i, F_{m-1}(x_i) + c)$$

d）更新模型：

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$$

3）得到最优模型:

$$F^*(x) = F_M(x) = F_0(x) + \sum_{m=1}^{M} \sum_{j=1}^{J} c_{mj} I(x \in R_m j)$$

3.2.3. GBDT分类算法

这里我们再看看 GBDT 分类算法，GBDT 的分类算法从思想上和 GBDT 的回归算法没有区别，但是由于样本输出不是连续的值，而是离散的类别，导致我们无法直接从输出类别去拟合类别输出的误差。

为了解决这个问题，主要有两个方法，**一个是用指数损失函数，此时 GBDT 退化为 Adaboost 算法。另一种方法是用类似于 Logistic 回归的对数似然损失函数的方法。也就是说，我们用的是类别的预测概率值和真实概率值的差来拟合损失。**本文仅讨论用对数似然损失函数的 GBDT 分类。而对于对数似然损失函数，我们又有二元分类和多元分类的区别。[4]

值得注意的是，GBDT 虽然使用 CART 决策树，但是已经不是用 gini 系数来做选择特征分割点了，gini 系数只适用分类问题，而 GBDT 实质是个回归问题。即便是 GBDT 来解决分类问题，实质上也是把它转化为回归问题（转化为概率分布），分类问题选择分裂点和 GBDT 做回归是一样的，比如用分裂后方差最小之类的指标。

1）二元 GBDT 分类算法

对于二元 GBDT，我们使用 negative binomial log-likehood 作为我们的损失函数，有点类似于 Logistic 回归的对数似然损失函数：

$$L(y, F(x)) = log(1 + exp(-2yF(x))), \quad y \in \{-1, +1\}$$

其中，

$$F(x)=\frac{1}{2}\log\left[\frac{Pr(y=1|x)}{Pr(y=-1|x)}\right]$$

上述的对数损失是 logit 函数（log odds），是在 Freidman 的论文中使用的公式，我认为使用在 Logistic 回归中常见的 $L(y,F)=ylogF+(1-y)log(1-F)$，其中 $F(z)=\frac{1}{1+exp(-z)}$ 也是可以的。

则此时的响应为

$$r_{mi}=-\left[\frac{\partial L(y,F(x_i)))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}=\frac{2y_i}{(1+exp(2y_iF(x_i)))}$$

对于生成的决策树，我们各个叶子节点的最佳残差拟合值为

$$c_{mj}=\arg\min_{c}\sum_{x_i\in R_{mj}}log(1+exp(-2y_i(F_{t-1}(x_i)+c)))$$

由于上式比较难优化，我们一般使用近似值（Newton-Raphson）代替

$$c_{mj}=\sum_{x_i\in R_{mj}}r_{mi}\Big/\sum_{x_i\in R_{mj}}|r_{mi}|(2-|r_{mi}|)$$

最终得到的 $F_M(x)$ 与对数几率 log-odds 相关，我们可以用来进行概率估计

$$F(x)=\frac{1}{2}log\left(\frac{p}{1-p}\right)$$

$$e^{2F(x)}=\frac{p}{(1-p)}$$

$$P_+(x)=p=\frac{e^{2F(x)}}{1+e^{2F(x)}}=\frac{1}{1+e^{-2F(x)}}$$

$$P_-(x)=1-p=\frac{1}{1+e^{2F(x)}}$$

有了概率之后，我们接下来就可以利用概率进行分类，除了负梯度计算和叶子节点的最佳残差拟合的线性搜索，二元 GBDT 分类和 GBDT 回归算法过程相同。

2）多元 GBDT 分类算法

多元 GBDT 要比二元 GBDT 复杂一些，对应的是多元 Logistic 回归和二元 Logistic 回归的复杂度差别。我们使用 multi-class log-loss 作为损失函数，假设类别数为 K，则此时我们的对数似然损失函数为：

$$L(y, F(x)) = -\sum_{k=1}^{K} y_k log\ p_k(x)$$

其中如果样本输出类别为 k，则 $y_k = 1$。我们可使用 softmax 来计算概率，第 k 类的概率 $p_k(x)$ 的表达式为：

$$p_k(x) = \frac{exp(F_k(x))}{\sum_{l=1}^{K} exp(F_l(x))}$$

可以看出，对于**多分类问题，我们需要为每个类别创建一棵树**$F_l(x),\ \ l = 1, 2, \cdots, k$。

结合上两式，我们可以计算出第 m 轮的第 i 个样本对应类别 l 的响应为：

$$r_{mil} = -\left[\frac{\partial L(y_i, F(x_i)))}{\partial F(x_i)}\right]_{F_k(x)=F_{l,m-1}\quad(x)} = y_{il} - p_{l,m-1}(x_i)$$

观察上式可以看出，其实这里的误差就是样本 i 对应类别 l 的真实概率和 $m-1$ 轮预测概率的差值。

对于生成的决策树，我们各个叶子节点的最佳残差拟合值为：

$$c_{tjl} = \arg\min_{c_{jl}} \sum_{i=0}^{m} \sum_{k=1}^{K} L(y_k, F_{m-1,l}(x) + \sum_{j=0}^{J} c_{jl} I(x_i \in R_{mj}))$$

由于上式比较难优化，我们一般使用近似值（Newton-Raphson）代替

$$c_{mjl} = \frac{K-1}{K} \frac{\sum\limits_{x_i \in R_{mjl}} r_{mil}}{\sum\limits_{x_i \in R_{mil}} |r_{mil}|(1-|r_{mil}|)}$$

除了负梯度计算和叶子节点的最佳残差拟合的线性搜索，多元 GBDT 分类和二元 GBDT 分类以及 GBDT 回归算法过程相同。

3.2.4. GBDT常用损失函数

这里我们再对常用的 GBDT 损失函数做一个总结。

先说两个最为常见的，也是我们上面提到的损失函数。

1）平方损失，主要应用于回归问题，使用平方损失的 GBDT，可称为 LSBoost。

损失函数：

$$L = \frac{1}{2}(y - F(x))^2$$

对 F 求导得到响应（此时响应=残差）：

$$r_i = -g_i = -\frac{\partial L(y_i, F(x_i)}{\partial F(x_i)} = y - F(x)$$

二阶导是：

$$h_i = 1$$

2）对数损失，主要应用于分类问题， 使用对数损失的 GBDT，可称为 LogitBoost。

损失函数：

$$L = \log(1 + exp(-2yF(x)))$$

对 F 求导得到响应：

$$r_i = -g_i = \frac{2y_i}{1 + \exp(2y_i F(x_i))}$$

二阶导是：

$$h_i = \frac{4\exp(2y_u F(x_i))}{(1 + \exp(2y_i F(x_i)))^2} = |g_i|(2 - |g_i|)$$

多分类可参见多元 GBDT 分类算法。

GBDT 的其它损失函数，用于分类：

1) 如果是指数损失函数，则损失函数表达式为

$$L(y, f(x)) = exp(-yF(x))$$

此时退化为 Adaboost 算法，严格来讲不属于 GBDT 范畴，详见 Adaboost。

用于回归：

2）绝对损失，这个损失函数也很常见

$$L(y, f(x)) = |y - F(x)|$$

对应负梯度误差为：

$$sign(y_i - F(x_i))$$

3）Huber 损失，它是均方差和绝对损失的折衷产物，对于远离中心的异常点，采用绝对损失，而中心附近的点采用平方误差损失。这个界限一般用分位数点度量。损失函数如下：

$$L(y, F(x)) = \begin{cases} \frac{1}{2}(y - F(x))^2 & |y - F(x)| \leq \delta \\ \delta(|y - F(x)| - \frac{\delta}{2}) & |y - F(x)| > \delta \end{cases}$$

对应的响应为：

$$r(y_i, F(x_i)) = \begin{cases} y_i - F(x_i) & |y_i - F(x_i)| \leq \delta \\ \delta sign(y_i - F(x_i)) & |y_i - F(x_i)| > \delta \end{cases}$$

4）分位数损失。它对应的是分位数回归的损失函数，表达式为

$$L(y, F(x)) = \sum_{y \geq F(x)} \theta |y - F(x)| + \sum_{y < F(x)} (1 - \theta) |y - F(x)|$$

其中 θ 为分位数，需要我们在回归前指定。

对应的响应为：

$$r(y_i, F(x_i)) = \begin{cases} \theta & y_i \geq F(x_i) \\ \theta - 1 & y_i < F(x_i) \end{cases}$$

对于 Huber 损失和分位数损失，主要用于健壮回归，也就是减少异常点对损失函数的影响。[4]

3.2.5. GBDT常用正则化方法

GBDT 有非常快的降低 Loss 的能力，这也会造成一个问题：Loss 迅速下降，容易使模型低 bias，高 variance，造成过拟合。

下面一一介绍 GBDT 中抵抗过拟合的技巧[5]，其中很多是在 XGBoost 中采用了的。

1）在损失后面增加正则惩罚项

$$\mathbf{J} = \sum_{i=1}^{N} L(y_i, F(x_i)) + \sum_{m=1}^{M} \Omega(f_m)$$

其中，$\mathbf{J}$ 是目标函数，f_m 是训练学习到的第 m 棵树，$f_m = T(\theta_m) = \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$，$\Omega$ 是正则化函数，它惩罚 f_m 的复杂度，树结构越复杂它的值越大，一般这个函数可以自定义，只需满足二次可微即可。

下面是陈天奇在 XGBoost 中使用的正则化函数：

$$\Omega(f_m) = \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} c_{mj}^2$$

其中，J 为第 m 棵树的叶子节点的个数，c_{mj} 是第 m 棵树第 j 个叶子结点的预测值，这里对 c_{mj} 使用 L2 正则化。γ 和 λ 是超参数，这两个超参数越大，表示越希望获得结构简单的树。当两个超参数为零时，正则化作用消失。正则化项可以平滑每个叶节点的学习权重来避免过拟合。

2）限制树的复杂度（树深度）

Ω 函数对树的结点数以及结点上预测值 c_{mj} 的平方和均有惩罚。除此之外，我们通常在终止条件上还会增加一条：树的深度。

3）行采样

训练每棵树的时候，只使用一部分样本，参考 Bagging。

4）列采样

训练每棵树的时候，只使用一部分特征，这是 XGBoost 的创新，它将随机森林中的思想引入了 GBDT。

5）衰减因子 Shrinkage

在更新 $F_m(x)$ 的时候，进一步惩罚 c_{mj}，给它们乘一个大于 0 小于 1 的系数 α（通常 $\alpha \in [0.001, 0.01]$），也可以理解为设置了一个较低的学习率，这样每次走一小步逐渐逼近结果的效果，要比每次迈一大步很快逼近结果的方式更容易避免过拟合。即它不完全信任每一个棵残差树，它认为每棵树只学到了真理的一小部分，累加的时候只累加一小部分，通过多学几棵树弥补不足。。

$$F_m(x) = F_{m-1}(x) + \alpha \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$$

6）早停止 Early Stop

因为 GBDT 的可叠加性，我们使用的模型不一定是最终的最优集合模型，因此可根据测试集的测试情况，选择使用前若干棵树。

4. 算法对比

Bagging 与 Boosting 算法区别

项目	Bagging	Boosting
样本选择	有放回式的自助采样，产生多个相互独立的训练集	使用原始训练集
样本权重	训练期间各样本权重相等	按错误率调整样本权重，提升对错误样本的关注以便下一轮优化
个体学习器权重	个体学习器权重相等	个体学习器权重不同，误差小的权重更大。
并行计算	个体学习器可并行生成	个体学习器前后依赖，无法并行生成
偏差方差分解	关注降低方差	关注降低偏差

$ 偏差-方差分解（重要）

为什么说 bagging 是减少 variance，而 boosting 是减少 bias? [6]

xgboost/gbdt 在调参时为什么树的深度很少就能达到很高的精度，而 DecisionTree/RandomForest 需要把树的深度调到很高? [7]

（个人）直观简单的理解：

Bagging 通过样本采样，一定几率减少了异常噪声（Outlier）出现在个体学习器的训练数据集中，在训练基学习器的过程中也就减少了噪声的干扰，最后再经过多个模型的平均或投票，进一步了降低异常噪声的影响，因此，Bagging 更加关注减小方差；

Boosting 是通过最优化损失来获得最优模型的，无论是 GBDT 中的拟合伪残差还是 Adaboost 中依据指数损失更新的权重值，Boosting 的每一次更新都是以减小损失为目的进行的优化，反应在最后获得模型上，就是最小化偏差。因此，Boosting 更加关注减小偏差。

（觉得还是用关注比较好，因为两种算法并不是绝对地只是减少方差或偏差，他们也可以靠个体学习器来弥补缺陷。）

严谨数学化的推导 [6]：

Bagging 对样本重采样，对每一重采样得到的子样本集训练一个模型，最后取平均。由于子样本集的相似性以及使用的是同种模型，因此各模型有近似相等的 bias 和 variance（事实上，各模型的分布也近似相同，但不独立）。由于 $E[\frac{\sum X_i}{n}] = E[X_i]$，所以 Bagging 后的 bias 和单个子模型的接近，一般来说不能显著降低 bias。另一方面，若各子模型独立，则有 $Var(\frac{\sum X_i}{n}) = \frac{Var(X_i)}{n}$，此时可以显著降低 variance。若各子模型完全相同，则 $Var(\frac{\sum X_i}{n}) = Var(X_i)$，此时不会降低 variance。Bagging 方法得到的各子模型是有一定相关性的，属于上面两个极端状况的中间态，因此可以一定程度降低 variance。为了进一步降低 variance，Random Forest 通过随机选取变量子集做拟合的方式 de-correlated 了各子模型（树），使得 variance 进一步降低。

（用公式可以一目了然：设有 i.d.「同分布但不一定独立，i.i.d. 是独立同分布」的 n 个随机变量，方差记为 σ^2，两两变量之间的相关性为 ρ，则 $\frac{\sum X_i}{n}$ 的方差为 $\rho * \sigma^2 + (1-\rho) * \sigma^2/n$，bagging 降低的是第二项，random forest 是同时降低两项。详见 ESL p588 公式 15.1 (https://lumingdong.cn/go/0fjapr)）

Boosting 从优化角度来看，是用 forward-stagewise 这种贪心法去最小化损失函数 $L(y, \sum_i \alpha_i f_i(x))$。例如，常见的 AdaBoost 即等价于用这种方法最小化 exponential loss：$L(y, f(x)) = \exp(-yf(x))$。所谓 forward-stagewise，就是在迭代的第 n 步，求解新的子模型 $f(x)$ 及步长 α（或者叫组合系数），来最小化 $L(y, f_{n-1}(x) + \alpha f(x))$，这里 $f_{n-1}(x)$ 是前 n-1 步得到的子模型的和。因此 Boosting 是在 sequential 地最小化损失函数，其 bias 自然逐步下降。但由于是采取这种 sequential、adaptive 的策略，各子模型之间是强相关的，于是子模型之和并不能显著降低 variance。所以说 Boosting 主要还是靠降低 bias 来提升预测精度。

注重偏差和注重方差带来的调参区别：

Bagging 更加关注降低方差，对样本重采样和增加个体学习器的数量，其主要目的就是降低方差，因此它在不剪枝的决策树、神经网络等学习器上效用更为明显。但要想使模型达到一定的精度，集成方法对降低偏差的作用就不明显了，所以只能通过提升个体学习器的性能来弥补，因此 Bagging 对个体学习器的性能要求也稍高一些，即便存在一些性能稍弱的，也会在投票和权重更新中被淘汰或降权。这也是为什么 RandomForest 在调参的时候，个体决策树的深度往往要调的比较高，就是为了极力降低偏差且不用考虑因树深导致方差高带来的影响。

Boosting 更加关注降低偏差，通过前向分步这种贪心算法，每一步我们都会在上一轮的基础上更加拟合原数据，因此 Boosting 对个体学习器的性能要求并不高，它能基于泛化性能相当弱的学习器构建出很强的集成，因为即便仍有偏差，也会不断用新的个体学习器进行拟合叠加来弥补上的，而且为了避免过拟合，往往选择方差更小的简单学习器，所以在 GBDT 和 XGBoost 的调参中，个体决策树的深度往往不高，

综上，我们可以发现 Bagging 和 Boosting 在调参的时候有不同的关注点，Bagging 更加注重降低方差，因此我们在模型优化的时候要更加注重如何降低偏差，甚至为了降低偏差还要适当的提高个体学习器的复杂度，却不用担心过拟合情况会发生，因为 Bagging 自带降低方差的效果，类似剪枝这种决策树的正则化方法就更不需要用了。而 Boosting 却恰恰相反，因为 Boosting 更加关注降低偏差，然而有时候因为降低偏差太快，往往有过拟合的风险，因此，Boosting 是需要正则方法的，比如可使用：类似 Bagging 中的自采样法，限制树的深度，降低偏差减小的速度，还有甚至是以牺牲一定偏差为代价的早停止方法。

5. 优缺点

两种集成算法的主要代表是随机森林和 GBDT，我们以这两个算法为例进行讨论。[8] [9]

5.1. 随机森林的优缺点

优点：

1. 表现性能好，与其他算法相比有着很大优势。
2. 随机森林能处理很高维度的数据（也就是很多特征的数据），并且不用做特征选择。
3. 在训练完之后，随机森林能给出哪些特征比较重要。
4. 训练速度快，容易做成并行化方法 (训练时，树与树之间是相互独立的)。
5. 在训练过程中，能够检测到 feature 之间的影响。
6. 对于不平衡数据集来说，随机森林可以平衡误差。当存在分类不平衡的情况时，随机森林能提供平衡数据集误差的有效方法。
7. 如果有很大一部分的特征遗失，用 RF 算法仍然可以维持准确度。

8. 随机森林算法有很强的抗干扰能力（具体体现在 6,7 点）。所以当数据存在大量的数据缺失，用 RF 也是不错的。
9. **随机森林抗过拟合能力比较强**（虽然理论上说随机森林不会产生过拟合现象，但是在现实中噪声是不能忽略的，增加树虽然能够减小过拟合，但没有办法完全消除过拟合，无论怎么增加树都不行，再说树的数目也不可能无限增加的。）
10. 随机森林能够解决分类与回归两种类型的问题，并在这两方面都有相当好的估计表现。（虽然 RF 能做回归问题，但通常都用 RF 来解决分类问题）。
11. 在创建随机森林时候，对 generlization error(泛化误差) 使用的是无偏估计模型，泛化能力强。

缺点：

1. 随机森林在解决回归问题时，并没有像它在分类中表现的那么好，这是因为它并不能给出一个连续的输出。当进行回归时，随机森林不能够做出超越训练集数据范围的预测，这可能导致在某些特定噪声的数据进行建模时出现过度拟合。（PS: 随机森林已经被证明在某些噪音较大的分类或者回归问题上会过拟合）。
2. 对于许多统计建模者来说，随机森林给人的感觉就像一个黑盒子，你无法控制模型内部的运行。只能在不同的参数和随机种子之间进行尝试。
3. 可能有很多相似的决策树，掩盖了真实的结果。
4. 对于小数据或者低维数据（特征较少的数据），可能不能产生很好的分类。（处理高维数据，处理特征遗失数据，处理不平衡数据是随机森林的长处）。
5. 执行数据虽然比 boosting 等快（随机森林属于 bagging），但比单只决策树慢多了。

5.2. GBDT的优缺点

优点：

1. 可以灵活处理各种类型的数据，包括连续值和离散值。
2. 在相对少的调参时间情况下，预测的准确率也可以比较高。这个是相对 SVM 来说的。
3. 使用一些健壮的损失函数，对异常值的鲁棒性非常强。比如 Huber 损失函数和 Quantile 损失函数。
4. 预测阶段树与树之间可并行化计算，计算速度快，最终的预测值就是所有树的预测值之和。
5. 在分布稠密的数据集上，泛化能力和表达能力都很好，这使得 GBDT 在 Kaggle 的众多竞赛中，经常名列榜首。
6. 采用决策树作为弱分类器使得 GBDT 模型具有较好的解释性和鲁棒性，能够自动发现特征间的高阶关系，并且也不需要对数据进行特殊的预处理如归一化等。

7. （特点）从决策边界来说，线性回归的决策边界是一条直线，Logistic 回归的决策边界根据是否使用核函数可以是一条直线或者曲线，而 GBDT 的决策边界可能是很多条线。

缺点：

1. 由于个体学习器之间存在依赖关系，训练过程需要串行训练，难以并行训练数据。只能在决策树内部采用一些局部并行的手段提高训练速度。如通过自采样的 SGBT 来达到部分并行，XGBoost 则实现了特征粒度上的并行计算。『采用子采样的 GBDT 有时也称为随机梯度提升树（SGBT）』
2. GBDT 在高维稀疏的数据集上，表现不如支持向量机或者神经网络。
3. GBDT 在处理文本分类特征问题上，相对其他模型的优势不如它在处理数值特征时明显。

6. 应用场景

集成学习的代表算法 RF 和 GBDT 在各个领域都有着不俗的表现，可以说是 “万能钥匙” 了。对于随机森林和 GBDT 的应用场景，仅总结一些不容易想到的地方，后面也会不断补充。

1. GBDT 可用于搜索引擎和推荐系统排序过程
2. GBDT 与 LR 融合提升广告点击率预估模型

7. 参考资料

[1] 周志华. 机器学习. 清华大学出版社↩↩↩

[2] 李航. 统计学习方法. 清华大学出版社↩↩

[3] wepon. GBDT 算法原理与系统设计简介 (https://lumingdong.cn/go/b7gmtk)↩

[4] 刘建平. 梯度提升树 (GBDT) 原理小结. cnblogs (https://lumingdong.cn/go/bedgl2)↩↩

[5] Yafei Zhang. GBDT 详解. 火光摇曳 (https://lumingdong.cn/go/wcc3ri)↩

[6] 过拟合. 为什么 Bagging 减小方差 Boosting 减小偏差. 知乎 (https://lumingdong.cn/go/vkd85l)↩↩

[7] 于菲. 为什么 gbdt 在调参时为什么树的深度很少就能达到很高的精度. 知乎 (https://lumingdong.cn/go/v1klsx)↩

[8] boyan_RF. 随机森林的生成方法以及优缺点. CSDN (https://lumingdong.cn/go/rnxwp1)↩

[9] 诸葛越 / 葫芦娃. 百面机器学习. 人民邮电出版社↩