Author：Liu Jian

Time：2020-06-07

机器学习8b-Gradient Boosting 和 XGBoost
1 Function estimation/approximation
1.1 概率分布已知
1.2 样本数据已知
2 Gradient Boosting
2.1 取不同的损失函数
2.2 基学习器为决策树 - Gradient Boost Decision Tree
2.2.1 回归树
2.2.2 分类树
3 XGBoost
3.1 理论
3.1.1 基本公式
3.1.2 划分算法
3.1.2.1 精确算法
3.1.2.2 近似算法
3.1.2.3 稀疏算法
3.1.3 其他
3.2 实现

机器学习8b-Gradient Boosting 和 XGBoost

> 参考文献
>
> 1. Greedy function approximation: a gradient boosting machine - Friedman - 1999
> 2. XGBoost: a scalable tree boosting system - Tianqi Chen - 2016
> 3. The Elements of Statistical Learning - 第 10 章
> 4. 个人博客-GBDT算法原理深入解析 可作参考

1 Function estimation/approximation

1.1 概率分布已知

期望风险最小化就是在求映射 $H^*:\mathbb{X}\to\mathbb{Y}$，即如下的**泛函问题**：

$$\begin{aligned}H^*&=\arg\min_H\mathbb{E}_{\boldsymbol{x},y}\{l(H(\boldsymbol{x}),y)\}\\&=\arg\min_H\mathbb{E}_{\boldsymbol{x}}\{\mathbb{E}_{y|\boldsymbol{x}}\{l(H(\boldsymbol{x}),y)\}\}\end{aligned}$$

其中，$l(\cdot)$ 为损失函数。

若**概率分布已知**，我们有两种方法求解上述问题：

1. **参数空间 (parameter space) 中寻优。**我们限制映射 H 的形式，**将函数优化问题转化为参数优化问题**：$H(\boldsymbol{x})\to H(\boldsymbol{x};\theta)$。而在 Boosting 中，我们假设 H 为如下的加性模型：

$$H(\boldsymbol{x};\theta)=H\left(\boldsymbol{x};\{\alpha_i,\beta_i\}_{i=1}^T\right)=\sum_{i=1}^T\alpha_ih_i(\boldsymbol{x};\beta_i)$$

其中，$h_i(\cdot)$ 为第 i 个基学习器，β_i 为其待定参数，各基学习器的类型可以不同，α_i 为基学习器的系数，因此 $H(\cdot)$ 是各基学习器的线性组合。

可以看到，**将函数优化问题转化为参数优化问题实际上就是限制了我们寻优的函数空间的大小，改变了原问题的定义域，因此得到的是一个次优解**：

$$H^* = \arg\min_H \mathbb{E}_{\boldsymbol{x},y}\{l(H(\boldsymbol{x}),y)\}$$

$$\downarrow$$

$$\theta^* = \arg\min_\theta \mathbb{E}_{\boldsymbol{x},y}\{l(H(\boldsymbol{x};\theta),y)\} \triangleq \arg\min_\theta \Phi(\theta)$$

接下来，我们就要求解这个参数优化问题。因为概率分布已知，因此上述参数优化问题的未知数只有参数 θ，我们可以：

(1) 解析法。求导，令导数为 0：

$$\frac{\partial\Phi(\theta)}{\partial\theta}\Big|_{\theta^*} = 0$$

(2) 数值解法：梯度下降法，$\theta^* = \theta_0 + \sum_{j=1}\Delta\theta_j$，$\theta_0$ 为设定的初始值。已知 $\theta_t = \theta_0 + \sum_{j=1}^{j=t}\Delta\theta_j$，我们要得到 θ_{t+1}：

$$\text{计算梯度：}\quad \frac{\partial\Phi(\theta)}{\partial\theta}\Big|_{\theta_t} \triangleq g_t$$

$$\text{沿负梯度方向进行线性搜索：}\quad \rho_t = \arg\min_\rho \Phi(\theta_t - \rho g_t)$$

$$\theta_{t+1} = \theta_t - \rho_t g_t$$

因为上式第二行中我们只是对 ρ (一个变量) 而不是参数向量 θ (多个变量) 进行优化，因此优化的难度降低了。

2. **函数空间 (function space) 中寻优。**我们记：$\Psi(H) = \mathbb{E}_{\boldsymbol{x},y}\{l(H(\boldsymbol{x}),y)\}$，其中因变量是一个映射，求这个映射 $H:\mathbb{X}\to\mathbb{Y}$，实际上就是要得到它在每种输入下的输出 $H(\boldsymbol{x}),\ \boldsymbol{x}\in\mathbb{X}$。比如，输入空间是离散的 $\mathbb{X} = \{\boldsymbol{q}_1,\boldsymbol{q}_2,\cdots\}$，对应的输入 $\{H(\boldsymbol{q}_1),H(\boldsymbol{q}_2),\cdots\} \triangleq \{o_1,o_2,\cdots\}$，我们实际上就在对“参数” $\{o_1,o_2,\cdots\}$ 进行优化，只不过，一般情况下输入是连续的，我们要估计的“参数”有无穷个。通过上述分析，我们记 $\psi(H(\boldsymbol{x})) = \mathbb{E}_{y|\boldsymbol{x}}\{l(H(\boldsymbol{x}),y)\}$，其中自变量 $H(\boldsymbol{x})$ 可以看做是很多个“参数” $\{o_1,o_2,\cdots\}$ 的统一形式。为了便于理解和区分，我们记自变量 $H(\boldsymbol{x}) \triangleq o$，可以看到，映射 H 的解与 $\boldsymbol{x}$ 的概率分布无关：

$$H^* = \arg\min_H \mathbb{E}_{\boldsymbol{x},y}\{l(H(\boldsymbol{x}),y)\} = \arg\min_H \mathbb{E}_{\boldsymbol{x}}\{\mathbb{E}_{y|\boldsymbol{x}}\{l(H(\boldsymbol{x}),y)\}\}$$

$$\Downarrow$$

$$H^*(\boldsymbol{x}) = \arg\min_{H(\boldsymbol{x})} \mathbb{E}_{y|\boldsymbol{x}}\{l(H(\boldsymbol{x}),y)\}$$

即，

$$H^* = \arg\min_H \Psi(H)$$

$$\Downarrow$$

$$H^*(\boldsymbol{x}) = \arg\min_{H(\boldsymbol{x})} \psi(H(\boldsymbol{x})) = \arg\min_o \mathbb{E}_{y|\boldsymbol{x}}\{l(o,y)\}$$

其中，概率分布 $p(y|\boldsymbol{x})$ 已知，$\mathbb{E}_{y|\boldsymbol{x}}\{l(o,y)\}$ 中的变量为 o 和 $\boldsymbol{x}$，而通过以 o 为变量，$\boldsymbol{x}$ 为固定量，最小化 $\mathbb{E}_{y|\boldsymbol{x}}\{l(o,y)\}$，我们就得到了变量 $o,\boldsymbol{x}$ 之间的关系，即函数 $o = H^*(\boldsymbol{x})$。若**解析求解存在难度，那么我们可以采用函数空间中的梯度下降法，即以函数为变量，进行优化 (比如对函数求偏导，实际上就是一个泛函问题)**，$H^*(\boldsymbol{x}) = H_0(\boldsymbol{x}) + \sum_{j=1} h_j(\boldsymbol{x})$，$H_0(\boldsymbol{x})$ 为给定的初始函数，已知 $H_t(\boldsymbol{x}) = H_0(\boldsymbol{x}) + \sum_{j=1}^{t} h_j(\boldsymbol{x})$，我们要得到 $H_{t+1}(\boldsymbol{x})$：

(1) 计算对函数 $H_t(\boldsymbol{x})$ 的梯度 (**我们以函数而不是某个参数为变量求导，因此是在函数空间中寻优**)：

$$\frac{\partial \psi(H(\boldsymbol{x}))}{\partial H(\boldsymbol{x})}\Big|_{H(\boldsymbol{x})=H_t(\boldsymbol{x})}=\frac{\partial \mathbb{E}_{y|\boldsymbol{x}}\{l(o,y)\}}{\partial o}\Big|_{o=H_t(\boldsymbol{x})}\triangleq g_t(\boldsymbol{x})$$

(2) 在函数空间中沿负梯度方向进行线性搜索：

$$\rho_t=\arg\min_{\rho}\mathbb{E}_{\boldsymbol{x},y}\{l(H_t(\boldsymbol{x})-\rho g_t(\boldsymbol{x}),y)\}$$

注意，我们选择的优化目标是 $\mathbb{E}_{\boldsymbol{x},y}\{l(H_t(\boldsymbol{x})-\rho g_t(\boldsymbol{x}),y)\}$，其中的变量只有 ρ， $\boldsymbol{x},y$ 都被积分积掉了，因此返回的是一个常数；当然，理论上我们也可以选择优化 $\mathbb{E}_{y|\boldsymbol{x}}\{l(H_t(\boldsymbol{x})-\rho g_t(\boldsymbol{x}),y)\}$，其中的变量有 $\rho,\boldsymbol{x}$，我们只优化 ρ，而视 $\boldsymbol{x}$ 为固定的，因此返回的是一个函数：

$$\rho_t(\boldsymbol{x})=\arg\min_{\rho}\mathbb{E}_{y|\boldsymbol{x}}\{l(H_t(\boldsymbol{x})-\rho g_t(\boldsymbol{x}),y)\}$$

但这一难度和直接进行： $H^*(\boldsymbol{x})=\arg\min_o\mathbb{E}_{y|\boldsymbol{x}}\{l(o,y)\}$ 相当，也就直接给出了最终结果 $H^*(\boldsymbol{x})$，不符合我们采用梯度下降法的前提假设。此外，对 $\mathbb{E}_{\boldsymbol{x},y}\{l(H_t(\boldsymbol{x})-\rho g_t(\boldsymbol{x}),y)\}$ 进行优化也并没有改变原始问题，我们仍然是针对原始问题在求解，而不是像参数空间寻优中那样，对原始问题作了限制(限定了 $H(\boldsymbol{x})$ 的函数形式)，求解的是一个次最优。

(3) 得到 $H_{t+1}(\boldsymbol{x})$：

$$h_{t+1}(\boldsymbol{x})=-\rho_t g_t(\boldsymbol{x})$$
$$H_{t+1}(\boldsymbol{x})=H_t(\boldsymbol{x})+h_{t+1}(\boldsymbol{x})$$

总地来说，无论是参数寻优还是函数寻优，若无法解析地求解，我们都会采用梯度下降算法：参数寻优时，函数的形式 $H(\boldsymbol{x};\theta)$ 已知，我们以参数为自变量，求目标函数关于参数的负梯度，每一步在前一步所得参数 θ_t 的基础上进行修正 $\theta_{t+1}=\theta_t+\Delta\theta_t$；函数寻优时，我们以函数为自变量，求目标函数关于函数的负梯度 (因此是一个泛函问题)，每一步在前一步所得函数 $H_t(\boldsymbol{x})$ 的基础上通过叠加一个增量函数 $h_{t+1}(\boldsymbol{x})$ 实现进一步的优化。可以看到，参数寻优和函数寻优的区别在于我们以参数还是函数为变量，进行求负梯度，更新等操作。此外，我们需要说明的是，函数寻优与参数寻优的判断与函数是否是参数形式无关，对参数形式的函数进行优化时也可以在函数空间中进行寻优，比如下文中的 Gradient Boosting。

1.2 样本数据已知

此时，我们手头有的是采样得到的数据 $\mathcal{D}=\{\boldsymbol{x}_i,y_i\}_{i=1}^N$，而概率分布 $p(\boldsymbol{x},y)$ 是未知的。问题变为经验风险最小化：

$$H^*=\arg\min_H\frac{1}{N}\sum_{i=1}^N l(H(\boldsymbol{x}_i),y_i)$$

首先，对于上述优化问题，若不假设 $H(\boldsymbol{x})$ 的为某种参数形式的函数 $H(\boldsymbol{x},\theta)$ (the parameterized class of functions)，我们是无法求解的。因为若没有模型类型的限制，求解上述优化问题时直接令 $H^*(\boldsymbol{x}_i)=y_i$ 即可，我们只能得到 $H^*(\boldsymbol{x})$ 在采样点 $\boldsymbol{x}_i$ 的值为 y_i，无法得到一个具有泛化能力的函数。因此，问题实际上为：

$$\theta^*=\arg\min_\theta\frac{1}{N}\sum_{i=1}^N l(H(\boldsymbol{x}_i;\theta),y_i)$$

现在，**我们的目标就是优化参数 θ，同样地，若问题比较简单，则可以直接通过求导得到解析解；否则，可以采用梯度下降算法进行迭代优化。但是，也可能存在梯度下降算法求解难度也很大的情况，这时，我们就需要考虑其他的优化方法了。**

接下来，我们考察模型为加性模型，且基学习器的类型都相同的情况 (**加性模型的出发点和傅里叶展开一样，都是通过增加被加项来不断地逼近真实函数**)：

$$H(\boldsymbol{x};\theta)=H\left(\boldsymbol{x};\{\alpha_i,\beta_i\}_{i=1}^T\right)=\sum_{i=1}^T\alpha_i h(\boldsymbol{x};\beta_i)$$

$$\{\alpha_i^*,\beta_i^*\}_{i=1}^T=\arg\min_{\{\alpha_i,\beta_i\}_{i=1}^T}\sum_{i=1}^N l\left(\sum_{i=1}^T\alpha_i h(\boldsymbol{x};\beta_i),y_i\right)$$

若梯度下降算法的优化难度也很大，那么我们可以采用分段 (stagewise) 优化策略，即每一步在前一步的基础上只优化一个基学习器，且不改变已有的优化结果 (分步 stepwise 策略还会对之前已优化的项进行再调整)。我们记 $\hat{y}_i^{(t)}=H_t(\boldsymbol{x}_i)=\sum_{j=1}^t\alpha_j^*h(\boldsymbol{x}_i;\beta_j^*)$ 为已优化的模型在 $\boldsymbol{x}_i$ 上的输出，是一个已知量，则第 $t+1$ 轮优化：

$$(\alpha_{t+1}^*,\beta_{t+1}^*)=\arg\min_{\alpha_{t+1},\beta_{t+1}}\sum_{i=1}^N l(\hat{y}_i^{(t)}+\alpha_{t+1}h(\boldsymbol{x}_i;\beta_{t+1}),y_i)$$

上述算法就是加性模型 (additive model) 的前向分段算法 (forward stagewise algorithm)，我们有：

(1) 对于二分类问题，当损失函数取指数损失时，根据上述优化策略就可以推得 AdaBoost 算法。注意 AdaBoost 中并没有讨论如何求解 $(\alpha_{t+1}^*,\beta_{t+1}^*)$ 。

(2) 对于某些特殊的损失函数和基学习器，
$(\alpha_{t+1}^*,\beta_{t+1}^*)=\arg\min_{\alpha_{t+1},\beta_{t+1}}\sum_{i=1}^N l(\hat{y}_i^{(t)}+\alpha_{t+1}h(\boldsymbol{x}_i;\beta_{t+1}),y_i)$ 的求解可能比较困难，无法直接得到解析解，那么我们可以采用如下被称为 **Gradient Boosting 的在函数空间中进行梯度下降的近似求解策略**：

(2.1) 计算损失函数关于函数 $H(\boldsymbol{x})$ 在每个样本点 $(\boldsymbol{x}_i,y_i)$ 的负梯度值 (也称 pseudo-responses)：

$$-g_t(\boldsymbol{x}_i)=-\frac{\partial l(H(\boldsymbol{x}_i),y_i)}{\partial H(\boldsymbol{x}_i)}\Big|_{H(\boldsymbol{x})=H_t(\boldsymbol{x})}=-\frac{\partial l(o,y_i)}{\partial o}\Big|_{o=\hat{y}_i^{(t)}},\quad i=1,\cdots,N$$

得到负梯度数据：$-\boldsymbol{g}_t=\{-g_t(\boldsymbol{x}_i)\}_1^N$ 。

(2.2) 梯度拟合确定 β_{t+1}^* 。使用基学习器 $h(\boldsymbol{x};\beta_{t+1})$ 拟合负梯度数据 $-\boldsymbol{g}_t$ ，我们使用平方损失 (理论上也可以使用其他损失，只不过平方损失的求解比较方便)：

$$\beta_{t+1}^*=\arg\min_{\beta_{t+1}}\sum_{i=1}^N(-g_t(\boldsymbol{x}_i)-h(\boldsymbol{x}_i;\beta_{t+1}))^2$$

我们拟合梯度数据实际上是对方向进行拟合，绝对大小没有影响，因此文献 [1] 中给出的计算公式中含有参数 ρ ，但显然二者等价：

$$\beta_{t+1}^*=\arg\min_{\beta_{t+1}}\sum_{i=1}^N(-g_t(\boldsymbol{x}_i)-\rho h(\boldsymbol{x}_i;\beta_{t+1}))^2$$

(2.3) 线性搜索确定 α_{t+1}^*：

$$\alpha_{t+1}^*=\arg\min_{\alpha_{t+1}}\sum_{i=1}^N l(\hat{y}_i^{(t)}+\alpha_{t+1}h(\boldsymbol{x}_i;\beta_{t+1}^*),y_i)$$

可以看到，**我们将复杂的优化问题**
$(\alpha_{t+1}^*,\beta_{t+1}^*)=\arg\min_{\alpha_{t+1},\beta_{t+1}}\sum_{i=1}^N l(\hat{y}_i^{(t)}+\alpha_{t+1}h(\boldsymbol{x}_i;\beta_{t+1}),y_i)$ **分解为一个最小二乘问题 (least-squares function minimization, 步骤 (2.2)) 和一个基于原损失函数的单参数优化问题 (a single parameter optimization based on the original criterion, 步骤 (2.3))。**

最后，我们补充如下两点：

(a) **关于步骤 (2.2) 的解释，即为什么我们要用基学习器 $h(\boldsymbol{x};\beta_{t+1})$ 拟合负梯度数据 $-\boldsymbol{g}_t$ 。**

总的损失函数为：

$$L=\sum_{i=1}^{N}l(H(\boldsymbol{x}_i),y_i)$$

我们在函数空间中进行梯度下降，即将函数视为优化变量，进行求导操作，沿负梯度函数进行线性搜索。函数 $H(\boldsymbol{x})$ 在每个样本输入 $\boldsymbol{x}_i$ 上的输出可视为一个待优化的"参数"，对这些"参数"求负梯度：

$$\begin{pmatrix}\dfrac{\partial L}{\partial H(\boldsymbol{x}_1)}\\ \vdots \\ \dfrac{\partial L}{\partial H(\boldsymbol{x}_N)}\end{pmatrix}=\begin{pmatrix}\dfrac{\partial l(H(\boldsymbol{x}_1),y_1)}{\partial H(\boldsymbol{x}_1)}\\ \vdots \\ \dfrac{\partial l(H(\boldsymbol{x}_N),y_N)}{\partial H(\boldsymbol{x}_N)}\end{pmatrix}$$

我们在函数空间中进行梯度下降，本轮函数 $H(\boldsymbol{x})$ 初始为 $H_t(\boldsymbol{x})$ ，优化形式为 $H_t(\boldsymbol{x})+\alpha h(\boldsymbol{x})$ ，搜索方向 $h(\boldsymbol{x})$ 使得总损失函数 L 下降得最快，即满足：

$$\left.\begin{pmatrix}\dfrac{\partial l(H(\boldsymbol{x}_1),y_1)}{\partial H(\boldsymbol{x}_1)}\\ \vdots \\ \dfrac{\partial l(H(\boldsymbol{x}_N),y_N)}{\partial H(\boldsymbol{x}_N)}\end{pmatrix}\right|_{H(\boldsymbol{x})=H_t(\boldsymbol{x})}=-\boldsymbol{g}_t=\begin{pmatrix}h(\boldsymbol{x}_1)\\ \vdots \\ h(\boldsymbol{x}_N)\end{pmatrix}$$

注意，前文提到，此时我们优化的函数都需要指定类型，即为参数形式的函数。对于参数形式的函数 $h(\boldsymbol{x};\beta_{t+1})$ ，我们无法保证上述等式恒成立，因此只能尽可能地去拟合负梯度数据 $-\boldsymbol{g}_t$ 。我们取拟合时的损失函数为均方误差，即得：

$$\beta_{t+1}^{*}=\arg\min_{\beta_{t+1}}\sum_{i=1}^{N}(-g_t(\boldsymbol{x}_i)-h(\boldsymbol{x}_i;\beta_{t+1}))^2$$

(b) 最后我们有： $H_{t+1}(\boldsymbol{x})=H_t(\boldsymbol{x})+\alpha_{t+1}^{*}h(\boldsymbol{x};\beta_{t+1}^{*})$ 。一般地，为了防止过拟合和避免陷入局部极小，我们还会引进一个学习率 (learning rate) $0<\upsilon<1$ ，令 $H_{t+1}(\boldsymbol{x})=H_t(\boldsymbol{x})+\upsilon\alpha_{t+1}^{*}h(\boldsymbol{x};\beta_{t+1}^{*})$ ，这种方法也被称为 shrinkage。此外，我们也可以不进行步骤 (2.3) 的线性搜索，直接取一个较小的 $0<\rho<1$ ，令 $H_{t+1}(\boldsymbol{x})=H_t(\boldsymbol{x})+\rho h(\boldsymbol{x};\beta_{t+1}^{*})$ 。

2 Gradient Boosting

我们总结 Gradient Boosting 算法如下：

--	**Algorithm: Gradient_Boost**
1	$H_1(\boldsymbol{x}) = \arg\min_{\rho} \sum_{i=1}^{N} l(y_i, \rho)$
2	For $t = 1, \cdots, T-1$ do:
3	$\tilde{y}_i = -\left[\dfrac{\partial l(H(\boldsymbol{x}_i), y_i)}{\partial H(\boldsymbol{x}_i)}\right]_{H(\boldsymbol{x})=H_t(\boldsymbol{x})}, i = 1, \cdots, N$
4	$\beta_{t+1}^* = \arg\min_{\beta_{t+1}} \sum_{i=1}^{N} (\tilde{y}_i - h(\boldsymbol{x}_i; \beta_{t+1}))^2$
5	$\alpha_{t+1}^* = \arg\min_{\alpha_{t+1}} \sum_{i=1}^{N} l(H_t(\boldsymbol{x}_i) + \alpha_{t+1} h(\boldsymbol{x}_i; \beta_{t+1}^*), y_i)$
6	$H_{t+1}(\boldsymbol{x}) = H_t(\boldsymbol{x}) + \alpha_{t+1}^* h(\boldsymbol{x}; \beta_{t+1}^*)$
7	endFor
--	end Algorithm

有了上述基本算法，选择不同的**损失函数**和**基学习器**就可以衍生出各种各样的 Gradient Boosting 算法。

2.1 取不同的损失函数

1. 损失函数为均方损失：$l(H(\boldsymbol{x}), y) = (H(\boldsymbol{x}) - y)^2/2$，即 Least-squares regression。此时 GB 算法第 3 行的 pseudo-response 为：

$$\tilde{y}_i = -\left[\frac{\partial l(H(\boldsymbol{x}_i), y_i)}{\partial H(\boldsymbol{x}_i)}\right]_{H(\boldsymbol{x})=H_t(\boldsymbol{x})} = y_i - H_t(\boldsymbol{x}_i)$$

算法 4、5 行可以合并为：

$$(\alpha_{t+1}^*, \beta_{t+1}^*) = \arg\min_{\alpha_{t+1}, \beta_{t+1}} (\tilde{y}_i - \alpha_{t+1} h(\boldsymbol{x}_i; \beta_{t+1}))^2$$

即 $\alpha_{t+1} h(\boldsymbol{x}; \beta_{t+1})$ 在对当前的残差 $\tilde{y}_i = y_i - H_t(\boldsymbol{x}_i),\ i = 1, \cdots, N$ 进行拟合。

2. 损失函数为绝对损失：$l(H(\boldsymbol{x}), y) = |H(\boldsymbol{x}) - y|$，即 Least-absolute-deviation (LAD) regression。此时 GB 算法第 3 行的 pseudo-response 为：

$$\tilde{y}_i = -\left[\frac{\partial l(H(\boldsymbol{x}_i), y_i)}{\partial H(\boldsymbol{x}_i)}\right]_{H(\boldsymbol{x})=H_t(\boldsymbol{x})} = sign(y_i - H_t(\boldsymbol{x}_i))$$

算法第 5 行的求解：

$$\begin{aligned}
\alpha_{t+1}^* &= \arg\min_{\alpha_{t+1}} \sum_{i=1}^{N} |y_i - H_t(\boldsymbol{x}_i) - \alpha_{t+1} h(\boldsymbol{x}_i; \beta_{t+1}^*)| \\
&= \arg\min_{\alpha_{t+1}} \sum_{i=1}^{N} |h(\boldsymbol{x}_i; \beta_{t+1}^*)| \left| \frac{y_i - H_t(\boldsymbol{x}_i)}{h(\boldsymbol{x}_i; \beta_{t+1}^*)} - \alpha_{t+1} \right| \\
&= \underset{w}{median} \left\{ \frac{y_i - H_t(\boldsymbol{x}_i)}{h(\boldsymbol{x}_i; \beta_{t+1}^*)} \right\}_1^N, \quad w_i = |h(\boldsymbol{x}_i; \beta_{t+1}^*)|
\end{aligned}$$

where $median_w\{\cdot\}$ is the weighted median (加权中位数) with weights w_i。

3. 此外，还有：

(1) 用于回归的 Huber Loss function：

$$l(H(\boldsymbol{x}),y)=\begin{cases}\dfrac{1}{2}(H(\boldsymbol{x})-y)^2, & |H(\boldsymbol{x})-y|\leqslant\delta\\ \delta(|H(\boldsymbol{x})-y|-\delta/2), & |H(\boldsymbol{x})-y|>\delta\end{cases}$$

相比均方损失，Huber 损失函数能较好地处理 long-tailed error distributions (长尾分布) 和 outliers (异常点；可以看到，当 $|H(\boldsymbol{x})-y|>\delta$，即二者相差较大时，很有可能是异常点，而此时损失只是 $|H(\boldsymbol{x})-y|$ 的一阶增量，因此相比均方损失的二阶增量，异常点产生的影响较小)。

(2) 用于二分类问题 $y\in\{-1,+1\}$ 的 negative binomial log-likelihood:
$l(H(\boldsymbol{x}),y)=\log\left(1+\exp\left(-2H(\boldsymbol{x})y\right)\right)$。

2.2 基学习器为决策树 - Gradient Boost Decision Tree

采用决策树作为基学习器的 Gradient Boosting 算法被称为 GBDT。

2.2.1 回归树

基学习器为 J 结点的回归树 (J -terminal node regression tree)：

$$h(\boldsymbol{x};\beta)=h(\boldsymbol{x};\{b_j,R_j\}_1^J)=\sum_{j=1}^{J}b_j\mathbb{I}(\boldsymbol{x}\in R_j)$$

其中 R_j 表示第 j 个叶节点对应的区域， b_j 表示第 j 个叶节点对应的输出。

已知 $H_t(\boldsymbol{x})$，下一步待求函数的形式：

$$H_t(\boldsymbol{x})+\alpha_{t+1}h(\boldsymbol{x};\beta_{t+1})=H_t(\boldsymbol{x})+\alpha_{t+1}\sum_{j=1}^{J}b_{j(t+1)}\mathbb{I}(\boldsymbol{x}\in R_{j(t+1)})$$

此时，GB 算法第 4、5 行如下：

$$\{b^*_{j(t+1)},R^*_{j(t+1)}\}_1^J=\underset{\{b_{j(t+1)},R_{j(t+1)}\}_1^J}{\arg\min}\sum_{i=1}^{N}\left(\tilde{y}_i-\sum_{j=1}^{J}b_{j(t+1)}\mathbb{I}(\boldsymbol{x}_i\in R_{j(t+1)})\right)^2$$

$$\begin{aligned}\alpha^*_{t+1}&=\arg\min_{\alpha_{t+1}}\sum_{i=1}^{N}l\left(H_t(\boldsymbol{x}_i)+\alpha_{t+1}\sum_{j=1}^{J}b^*_{j(t+1)}\mathbb{I}(\boldsymbol{x}_i\in R^*_{j(t+1)}),y_i\right)\\&=\arg\min_{\alpha_{t+1}}\sum_{i=1}^{N}l\left(H_t(\boldsymbol{x}_i)+\sum_{j=1}^{J}\alpha_{t+1}b^*_{j(t+1)}\mathbb{I}(\boldsymbol{x}_i\in R^*_{j(t+1)}),y_i\right)\end{aligned}$$

可以看到，**算法第 4 行通过拟合算法第 3 行得到的 pseudo responses 得到一颗回归树 $\{b^*_{j(t+1)},R^*_{j(t+1)}\}_1^J$，算法第 5 行则对这棵回归树进行整体缩放来拟合原始数据。注意，是整体缩放，而事实上，我们可以作更精细的拟合，即修改算法第 5 行，我们只使用算法第 4 行得到的结点区域 $\{R^*_{j(t+1)}\}_1^J$，并直接对各结点对应的输出 (不妨记为 $\{\gamma_{j(t+1)}\}_1^J$) 进行优化 (即将式中的 $\alpha_{t+1}b^*_{j(t+1)}$ 换为 $\gamma_{j(t+1)}$)**：

$$\{\gamma^*_{j(t+1)}\}_1^J=\arg\min_{\{\gamma_{j(t+1)}\}_1^J}\sum_{i=1}^{N}l\left(H_t(\boldsymbol{x}_i)+\sum_{j=1}^{J}\gamma_{j(t+1)}\mathbb{I}(\boldsymbol{x}_i\in R^*_{j(t+1)}),y_i\right)$$

由此得到的下一轮的模型：

$$H_{t+1}(\boldsymbol{x})=H_t(\boldsymbol{x})+\sum_{j=1}^{J}\gamma^*_{j(t+1)}\mathbb{I}(\boldsymbol{x}\in R^*_{j(t+1)})$$

注意，上述讨论中我们没有指定损失函数 l ，接下来，我们给定不同的损失函数 l 也就对应了不同的具体算法。

可以看到，基学习器的结点个数 J 是一个超参数。从方差分析 (ANOVA) 的角度，我们指出，A tree with J terminal nodes produces a function with interaction **at most** $\min(J-1,d)$ ，其中， d 表示输入 $\boldsymbol{x}$ 的维度。

2.2.2 分类树

二分类和多分类，可参见文献 [1]，不再赘述。

3 XGBoost

XGBoost 包含两个方面：(1) 理论上的创新；(2) XGBoost 在实现上的设计与优化。

网上很多人把 XGBoost 和 GBDT 搞混，或者认为 XGBoost 属于 GB 体系，**个人认为这是不对的。** GBDT 和 XGBoost 均是在函数空间中进行寻优，GBDT 通过拟合梯度数据，再代入原目标函数中进行线性搜索；而 XGBoost 则对原目标函数进行二阶泰勒展开，优化的是近似目标函数。虽然 GBDT 利用了一阶导数信息，但并不是像 XGBoost 那样进行的泰勒展开，因此，它们并不属于同一体系。

3.1 理论

3.1.1 基本公式

样本数据 $\{\boldsymbol{x}_i, y_i\}_i^N$ ，模型：

$$H(\boldsymbol{x}) = \sum_{i=1}^{T} h_i(\boldsymbol{x}), \ \ h_i \in \mathcal{H}$$

其中， $\mathcal{H}$ 为决策树空间：

$$\mathcal{H} = \{h(\boldsymbol{x}) = w_{q(\boldsymbol{x})} | q : \mathbb{R}^d \to J, \ \boldsymbol{w} = (w_1, \cdots, w_J)^T \in \mathbb{R}^J\}$$

Here q represents the structure of each tree that maps an example to the corresponding leaf index， w_j 给出了第 j 个叶子结点的权值。

采用前向分段算法进行学习，已知 $t-1$ 轮的学习器对 $\boldsymbol{x}_i$ 输出的结果 $\hat{y}_i^{(t-1)}$ ，我们要学习 h_t ，目标函数：

$$L^{(t)} = \sum_{i=1}^{N} l(\hat{y}_i^{(t-1)} + h_t(\boldsymbol{x}_i), y_i) + \Omega(h_t)$$

$$\text{where } \ \Omega(h_t) = \frac{1}{2}\lambda \sum_{j=1}^{J} w_j^2 + \mu J$$

其中，损失函数 l 为可微的凸函数， $\Omega(h_t)$ 为正则项。**接下来，我们将目标函数 $L^{(t)}$ 在 $\hat{y}_i^{(t-1)}$ 处进行二阶泰勒展开，我们的优化目标从 $L^{(t)}$ 变为其近似函数 $\tilde{L}^{(t)}$ ：**

$$L^{(t)} \approx \sum_{i=1}^{N}\left[l(\hat{y}_i^{(t-1)}, y_i) + g_i h_t(\boldsymbol{x}_i) + \frac{1}{2}s_i h_t^2(\boldsymbol{x}_i)\right] + \Omega(h_t)$$

$$\tilde{L}^{(t)} \triangleq \sum_{i=1}^{N}\left[g_i h_t(\boldsymbol{x}_i) + \frac{1}{2}s_i h_t^2(\boldsymbol{x}_i)\right] + \Omega(h_t)$$

$$\text{where}$$

$$g_i = \frac{\partial l(o, y_i)}{\partial o}\Big|_{o=\hat{y}_i^{(t-1)}}$$

$$s_i = \frac{\partial^2 l(o, y_i)}{\partial o^2}\Big|_{o=\hat{y}_i^{(t-1)}}$$

上述操作体现了近似的思想 (数值渐近法也作了二阶泰勒展开，但 ANM 处理的是非线性方程的求解，若不是二次型，求解误差会不断积累；而这里是一个优化问题，没有误差积累一说。实际上，这里就是针对优化问题的牛顿法的二阶泰勒展开近似步，只不过牛顿法接下来还会进行迭代，而这里并没有进行迭代了。注意这里说的迭代仍然是在学习第 t 个学习器，不要与第 $t+1$ 个学习器的训练搞混了)；此外，求偏导时 o 代表的是函数 $H(\boldsymbol{x})$，求导对象是函数，因此，我们是在函数空间中寻找最优解。

对于 h_t 我们要求解两个量：结构 q 和叶子结点权重 $\boldsymbol{w}$ (这就类似于 BSPCE 中的基函数集合和基函数系数)。我们记 $I_j = \{i|q(\boldsymbol{x}_i) = j,\ i = 1, \cdots, N\}$ 为属于结点 j 的样本点的指标集合，则：

$$\begin{aligned}\tilde{L}^{(t)} &= \sum_{i=1}^{N}\left[g_i h_t(\boldsymbol{x}_i) + \frac{1}{2}s_i h_t^2(\boldsymbol{x}_i)\right] + \frac{1}{2}\lambda\sum_{j=1}^{J}w_j^2 + \mu J \\ &= \sum_{j=1}^{J}\left[(\sum_{i\in I_j}g_i)w_j + \frac{1}{2}(\lambda + \sum_{i\in I_j}s_i)w_j^2\right] + \mu J\end{aligned}$$

当结构 q 给定时，可得 w_j 的最优解析解 w_j^*：

$$w_j^* = -\frac{\sum_{i\in I_j}g_i}{\lambda + \sum_{i\in I_j}s_i}$$

此时， $\tilde{L}^{(t)}$ 的值为：

$$\tilde{L}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{J}\frac{(\sum_{i\in I_j}g_i)^2}{\lambda + \sum_{i\in I_j}s_i} + \mu J$$

可以看到，**类似于 BSPCE 的 KIC，上式可用于结构 q 的评估，我们称之为 scoring function (打分函数)。**This score is like the impurity score for evaluating decision trees，except that it is derived for a wider range of objective functions (打分函数就类似于用于评估决策树的纯度指标，只不过它是针对更广泛的目标函数得出的).

Normally it is impossible to enumerate all the possible tree structures q. A greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used instead (**通常，不可能列举并评估所有可能的树结构 q，取而代之的是一种贪心算法，该算法从单个叶子开始，反复向树中添加分支**).假设我们当前要分裂的结点所含样本指标为 I，分裂后的左右结点分别为 I_L, I_R，$I = I_L \bigcup I_R$，则**结点分裂前后损失函数的减少量为**：

$$L_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L}g_i)^2}{\lambda + \sum_{i\in I_L}s_i} + \frac{(\sum_{i\in I_R}g_i)^2}{\lambda + \sum_{i\in I_R}s_i} - \frac{(\sum_{i\in I}g_i)^2}{\lambda + \sum_{i\in I}s_i}\right] - \mu$$

我们就可以基于上式选择划分点，对一个结点进行分裂，最好的划分点应使上式最大。

3.1.2 划分算法

基于上一小节得到的公式，本小节我们给出**针对结点 I 的分裂算法**，包括：精确算法 (exact greedy algorithm)，近似算法 (approximate algorithm) 和处理稀疏数据的稀疏算法 (sparsity-aware split finding)。其中，**稀疏算法 (包括稀疏精确算法和稀疏近似算法) 是 XGBoost 结点分裂算法的最终形态。**

3.1.2.1 精确算法

对于结点 I 的分裂，我们需要知道划分的**属性**和**大小**。为此，精确算法依次考察每个属性 (输入 $\boldsymbol{x}$ 共有 d 个属性)，而为了得到第 i 个属性的最佳划分点，我们按照第 i 个属性值的大小对结点 I 中的样本点进行排序 $sorted(I,i)$ ，再依次考察所有可能的划分点，考察完所有属性的所有可能的划分点 (这就是"精确"的含义) 后，得分最高的情况就是我们想要的。可以看到，对于整颗树而言，算法是贪心的，我们每次都只考虑使下一步最优的情况，而不是整体最优。

--	**Exact Greedy Algorithm for Split Finding**
Input	I instance set of current node
initialization	$G \leftarrow \sum_{i\in I} g_i,\ \ S \leftarrow \sum_{i\in I} s_i,\ \ score \leftarrow 0$
1	For $i = 1, \cdots, d$ do:
2	$\quad G_L \leftarrow 0,\ \ S_L \leftarrow 0$
3	$\quad$ For j in $sorted(I,i)$ do:
4	$\quad\quad G_L \leftarrow G_L + g_j,\ \ S_L \leftarrow S_L + s_j$
5	$\quad\quad G_R \leftarrow G - G_L,\ \ S_R \leftarrow S - S_L$
6	$\quad\quad score \leftarrow \max\left(score, \frac{G_L^2}{\lambda + S_L} + \frac{G_R^2}{\lambda + S_R} - \frac{G^2}{\lambda + S}\right)$
7	$\quad$ endFor
8	endFor
Output	Split with max score

3.1.2.2 近似算法

近似算法与精确算法的区别在于对于某个属性，精确算法会考察所有可能的划分点，而近似算法只选择具有代表性的点进行考察。因此，近似算法的一个关键问题在于如何生成划分点。

划分点的生成方式有两种，一种是**全局性的**，即基于所有样本点生成划分点，供接下来的每次结点分裂使用；另一种是**局部性的**，即基于当前待分裂结点所含的样本点生成划分点，再进行结点分裂。

不管是全局的还是局部的，**划分点的生成均是基于加权分位数 (weighted quantile sketch) 进行的，权重由样本点处的二阶偏导给出。**我们以全局生成为例，对于第 i 个属性，我们要生成 q 个划分点 $\{c_{i1}, c_{i2}, \cdots, c_{iq}\}$ 。我们首先对样本 $\{\boldsymbol{x}_j\}_1^N$ 按照其第 i 个属性的大小进行排序得到 $\{\boldsymbol{x}_{ij}\}_1^N$ ，则划分点应该满足 $c_{i1} = \boldsymbol{x}_{ij}, c_{iq} = \boldsymbol{x}_{iN}$ 且落在划分点间的样本点的二阶偏导之和相等或者说尽可能相等：

$$\frac{\sum_{c_{ik} < \boldsymbol{x}_j < c_{i(k+1)}} s_j}{\sum_{j=1}^N s_j} \approx \frac{1}{q-1},\ \ k = 1, \cdots, q-1$$

对于加权分位数的生成，XGBoost 也提出了一种大数据也适用的实现算法，文献 [2] 中没有介绍但给出了链接。

至于**为什么权重是二阶偏导**，这是因为损失函数：

$$\tilde{L}^{(t)}=\sum_{i=1}^{N}\frac{1}{2}s_i\left(h_t(\boldsymbol{x}_i)-\frac{g_i}{s_i}\right)^2+\Omega(h_t)+constant$$

可视为标记是 $\frac{g_i}{s_i}$ 权重为 s_i 的均方损失。

3.1.2.3 稀疏算法

现实中，常常会遇到输入 $\boldsymbol{x}$ 是稀疏的情况，稀疏数据可能由如下原因产生：(1) 数据中存在缺失值；(2) 数据中 0 值大量出现；(3) 特征工程，比如 one-hot encoding。**因此，我们还需对前文所提出的精确算法和近似算法作进一步改进以能够处理稀疏数据。**为此，XGBoost 对每个树结点都赋予一个默认方向，当输入 $\boldsymbol{x}$ 相应的属性值缺失时，该输入就被划分进默认的方向 (进入左子树或右子树)。**划分属性和大小，最优的默认方向同时从数据中学得，下面我们给出精确稀疏算法 (稍经修改可得近似稀疏算法，不再赘述)。**

思路：对于结点 I 的分裂，我们依次考察每个属性 $k,\ (k=1,\cdots,d)$，寻找每个属性对应的最优划分点和默认方向，再在所有属性中寻找最优的情况。而对于某个属性 k，划分点在第 k 个属性值没有缺失的样本中产生，即样本集合 $I^k=\{i\in I|\boldsymbol{x}_i^{(k)}\neq missing\}$；而为了得到最优的默认方向，我们依次考察 k 属性值缺失样本全部被划到右边 (算法 2-7 行) 和左边 (算法 8-13 行) 的情况。注意，k 属性值缺失样本虽然在第 k 个属性上没有值，但其一阶导 g_i 和二阶导 s_i 是存在的，因此会对默认方向产生影响：

1. 算法 line 5 中，G_L,S_L 中包含的样本点全部是 k 属性值**存在**的样本点，因此 $G_R=G-G_L,S_R=S-S_L$ 中包含所有的 k 属性值**缺失**的样本点，即考察的默认方向是**右边**；
2. 算法 line 11 中，G_R,S_R 中包含的样本点全部是 k 属性值**存在**的样本点，因此 $G_L=G-G_R,S_L=S-S_R$ 中包含所有的 k 属性值**缺失**的样本点，即考察的默认方向是**左边**；

--	**Sparsity-aware Split Finding**
Input	I instance set of current node
Input	$I^k = \{i \in I \vert \boldsymbol{x}_i^{(k)} \neq missing\}$，即结点 I 中第 k 个分量的值存在而没有丢失的样本
initialization	$G \leftarrow \sum_{i \in I} g_i,\ \ S \leftarrow \sum_{i \in I} s_i,\ \ score \leftarrow 0$
1	For $k = 1, \cdots, d$ do:
2	$G_L \leftarrow 0,\ \ S_L \leftarrow 0$
3	For i in $sorted(I^k, k, ascent)$ do:
4	$G_L \leftarrow G_L + g_i,\ \ S_L \leftarrow S_L + s_i$
5	$G_R \leftarrow G - G_L,\ \ S_R \leftarrow S - S_L$
6	$score \leftarrow \max\left(score, \frac{G_L^2}{\lambda + S_L} + \frac{G_R^2}{\lambda + S_R} - \frac{G^2}{\lambda + S}\right)$
7	endFor
8	$G_R \leftarrow 0,\ \ S_R \leftarrow 0$
9	For i in $sorted(I^k, k, descent)$ do:
10	$G_R \leftarrow G_R + g_i,\ \ S_R \leftarrow S_R + s_i$
11	$G_L \leftarrow G - G_R,\ \ S_L \leftarrow S - S_R$
12	$score \leftarrow \max\left(score, \frac{G_L^2}{\lambda + S_L} + \frac{G_R^2}{\lambda + S_R} - \frac{G^2}{\lambda + S}\right)$
13	endFor
14	endFor
Output	Split and default directions with max score

3.1.3 其他

为防止过拟合，我们还可以采用 shrinkage (引进学习率 υ)、column (feature) subsampling (借鉴于随机森林)、row subsampling。根据经验，column subsampling 比 row subsampling 比防止过拟合的效果更好。此外，column subsampling 也有助于提高并行算法的计算效率。

3.2 实现

除了理论上的创新，XGBoost 的优良性能还源于其具体实现上，二者结合使得 XGBoost 非常强大。

XGBoost 在实现上的特点包括：column block 数据结构的引入、缓存机制 (cache access patterns)、block compression and sharding (有助于核外计算 out-of-core computation)。这使得 XGBoost is able to solve realworld scale problems using a minimal amount of resources，具体细节可见文献 [2] 第 4 部分。此外，文献第 6 章算例部分还给出了一系列实验结果，显示出 XGBoost 在相关方面 (处理稀疏数据、并行、核外计算、分布式计算等) 的优越性能。