An Introduction to Tree Models

张春光
WISER Club 2018, XMU

October 3, 2018

Contents

1 Decision Tree

1.1 基本概念

决策树模型可以分为决策树生成和决策树剪枝两个过程，决策树生成就是结点递归分裂产生叶结点的过程，决策树剪枝就是将某些叶结点移除的过程，所以决策树学习过程主要关注三点：

- 结点如何分裂
- 生成叶结点条件
- 如何剪枝

1.2 决策树分裂

结点分裂的目的是为了提升结点的纯度，使得分裂后子结点中所属的样本尽可能属于同一类别，减少数据集的不确定性。目前不确定性衡量的方法有基于信息增益的信息论和统计检验方法。基于信息论的方法主要有信息增益、信息增益率、基尼指数以及 TwoValuing 等，而基于统计检验的方法主要为卡方检验方法。

1.2.1 离散属性

假定决策树中一结点 $\mathbb{N}$ 中包含 k 个样本 $\{(\mathbf{x}_i, y_i) \,|\, i = 1, 2, \ldots, n;\, y_i = 1, 2, \ldots, k\}$，样本包含 k 类，每一类样本对应的比例为 p_k，该结点可分裂的属性集合为 $\mathcal{A} = \{\mathbf{a}_i \,|\, i = 1, 2, \ldots, m\}$，令 $\mathbb{N}^v$ 为结点中属性 $\mathbf{a}$ 上取值为 a^v 的样本集合，定义 $|\mathbb{N}|$ 为结点包含的样本容量。

(1) 信息增益

定义结点的纯度指标信息熵 (information entropy) 为：

$$\mathrm{Ent}(\mathbb{N}) = -\sum_{i=1}^{k} p_i \log_2 p_i$$

结点的信息熵 $\mathrm{Ent}(\mathbb{N})$ 值越高，则结点纯度越高，则结点 $\mathbb{N}$ 根据属性 $\mathbf{a}$ 进行分裂的信息增益为：

$$\mathrm{EntGain}(\mathbb{N}, \mathbf{a}) = \mathrm{Ent}(\mathbb{N}) - \sum_{i=1}^{v} \frac{|\mathbb{N}^i|}{|\mathbb{N}|} \mathrm{Ent}(\mathbb{N}^i)$$

则最优属性分裂为：

$$\mathbf{a}_* = \arg\max_{\mathbf{a}} \mathrm{EntGain}(\mathbb{N}, \mathbf{a})$$

即信息增益越大，样本纯度越高，分裂效果越好，考虑以下不等式：

$$\begin{aligned} -p_i \log_2 p_i - p_j \log_2 p_j + (p_i + p_j) \log_2 (p_i + p_j) &= -p_i \log_2 \frac{p_i}{p_i + p_j} - p_j \log_2 \frac{p_j}{p_i + p_j} \\ &\geq 0 + 0 = 0 \\ -p_i \log_2 p_i - p_j \log_2 p_j &\geq -(p_i + p_j) \log_2 (p_i + p_j) \end{aligned}$$

因此若一个属性 $\mathbf{a}$ 的取值数目越多，则该属性的信息增益会越高，在结点进行分裂时决策树会偏好取值较多的属性进行分裂，不过在样本量较大时可以不考虑该有偏性。

(2) 信息增益率

由于信息增益在结点分裂时偏好取值数目多的属性进行划分，因此需要对属性的取值数目加一个惩罚项进行平衡，定义属性 $\mathbf{a}$ 的属性分离信息为：

$$\text{SplitInfo}(\mathbb{N},\, \mathbf{a}) = -\sum_{i=1}^{v} \frac{|\mathbb{N}^i|}{|\mathbb{N}|} \log_2 \frac{|\mathbb{N}^i|}{|\mathbb{N}|}$$

则结点 $\mathbb{N}$ 根据属性 $\mathbf{a}$ 进行分裂的信息增益率：

$$\text{EntGainRatio}(\mathbb{N},\, \mathbf{a}) = \frac{\text{EntGain}(\mathbb{N},\, \mathbf{a})}{\text{SplitInfo}(\mathbb{N},\, \mathbf{a})} = \frac{\text{Ent}(\mathbb{N}) - \sum_{i=1}^{v} \frac{|\mathbb{N}^i|}{|\mathbb{N}|} \text{Ent}(\mathbb{N}^i)}{-\sum_{i=1}^{v} \frac{|\mathbb{N}^i|}{|\mathbb{N}|} \log_2 \frac{|\mathbb{N}^i|}{|\mathbb{N}|}}$$

SplitInfo($\mathbb{N}$, $\mathbf{a}$) 称为属性 $\mathbf{a}$ 的分离信息，在 0 和 1 两处接近于 0 且随着属性取值个数减少而减少，因此仅考虑属性的信息增益率进行分裂，则结点在分裂时会偏好取值较不平衡或取值个数较少的属性，因此一般使用信息增益率来筛选属性集合再用信息增益选取最优属性即最优属性分裂为：

$$\mathbf{a}_* = argmax_{\mathbf{a} \in \{\mathbf{a}_i \,|\, \text{EntGainRatio}(\mathbb{N},\, \mathbf{a}_i) \geq \text{mean}(\text{EntGainRatio}(\mathbb{N},\, \mathbf{a}_i))\}} EntGain(\mathbb{N},\, \mathbf{a})$$

即先筛选出属性信息增益率高于平均信息增益率的属性集合，再计算信息增益选取最优属性。

(3) 基尼指数

基尼指数一般用于二叉树模型中，表示结点分裂前后的基尼值的提升，基尼值直观理解为从结点中随机抽取两个样本不属于同一类的概率，因此基尼指数和信息熵不同，基尼指数越小表示结点纯度越高，即

$$\text{Gini}(\mathbb{N}) = 1 - \sum_{i=1}^{k} p_i^2$$

$$\text{GiniIndex}(\mathbb{N},\, \mathbf{a}) = \sum_{j=1}^{v} \frac{|\mathbb{N}^j|}{|\mathbb{N}|} \text{Ent}(\mathbb{N}^j)$$

$GiniIndex$ 从函数形式上也会偏好多值属性，但由于 $GiniIndex$ 常用于二叉树，所以基本不考虑其属性有偏影响。

(4) TwoingValue 等

$$\text{TwoingValue} = \frac{|\mathbb{N}_L| * |\mathbb{N}_R|}{|\mathbb{N}|^2} \left(\sum_{i=1}^{m} \left| \frac{L_i}{|\mathbb{N}_L|} - \frac{R_i}{|\mathbb{N}_R|} \right| \right)^2$$

其中 L_i, R_i 表示属性 $\mathbf{a}_i$ 在左右结点 $\mathbb{N}_L$, $\mathbb{N}_R$ 中的样本个数。

(5) 卡方检验：因基于信息论的属性选择有偏，QUEST 及 CHAID 决策树算法基于卡方检验选择最优属性结点属性分裂的检验统计量为：

$$G^2(\mathbb{N},\, \mathbf{a}) = 2 \ln 2 * |\mathbb{N}| \text{EntGain}(\mathbb{N},\, \mathbf{a}) \sim \chi^2_{U(\mathbf{a})-1,|\mathbb{N}|-1}$$

选取 p–value 值最低的属性作为最优分裂属性。

1.2.2 连续属性

决策树从本质上来讲属于分类方法，因此对于连续属性特征的分裂需要先根绝属性的取值计算不同数值分裂的增益指标选取最优划分点，再与其他属性比较，具体计算如下：

- 假定连续属性 $\mathbf{a}$ 有 v 个不同的取值 $\{a^1, a^2, \ldots, a^v\}$, 先属性特征进行排序得到 $\{a^{(1)}, a^{(2)}, \ldots, a^{(v)}\}$;

- 对于排序后的数列得到属性 $\mathbf{a}$ 的中值数列 $T_a = \{\frac{a^{(1)}+a^{(2)}}{2}, \frac{a^{(2)}+a^{(3)}}{2}, \ldots, \frac{a^{(v-1)}+a^{(v)}}{2}\}$，计算每个中值进行分裂的信息增益 $EntGain(\mathbb{N}, \mathbf{a}, T_a)$：

$$\begin{aligned}\text{EntGain}(\mathbb{N}, \mathbf{a}, t^*) &= \max_{t\in T_a} \text{EntGain}(\mathbb{N}, \mathbf{a}, t) \\ &= \max_{t\in T_a} \left(\text{Ent}\left(\mathbb{N}\right) - \sum_{\lambda\in\{L,R\}} \frac{|\mathbb{N}_\lambda^t|}{|\mathbb{N}|} \text{Ent}\left(\mathbb{N}_\lambda^t\right) \right)\end{aligned}$$

- 由于信息增益偏好多值属性，因此要对连续属性的信息增益进行修正，选择修正信息增益最大的分裂点作为该属性的划分点对数据划分：

$$\text{EntGainRevise}(\mathbb{N}, \mathbf{a}, t^*) = \text{EntGain}(\mathbb{N}, \mathbf{a}, t^*) - \frac{\log_2(v-1)}{|\mathbb{N}|}$$

1.2.3 缺失值

决策树对于缺失值的处理主要关注两点：

- **有缺失值的属性如何计算分裂指标**
  基于每个属性的无缺失样本 $\bar{N}$ 计算信息增益或基尼系数，而后乘以无缺失比例:

$$\text{EntGain}(\mathbb{N}, \mathbf{a}) = \rho \times \text{EntGain}(\bar{\mathbb{N}}, \mathbf{a}), \quad \rho = \frac{|\bar{\mathbb{N}}|}{|\mathbb{N}|}$$

- **有缺失值的样本如何划分**
  在基于无缺失的样本属性选择之后，将每个缺失样本按照无缺失样本的后验概率的权重划分入各个子节点中。

1.2.4 多变量组合划分

多变量决策树对于非叶结点划分时是对属性的线性组合进行测试，试图在每次属性划分时建立一个线性分类器，多变量决策树中较为常用的是 CART Linear Combination 算法，OC1 算法 (Oblique Classifier 1)，其大致思想如下:

- CART Linear Combination

  (1) 先基于单变量选择最优属性 $\mathbf{x}_{j.}$，建立多变量组合分割平面 H: $\sum_{i=1}^m a_i x_i + a_{m+1} = 0$

  (2) 每次基于当前结点数据优化一个 $\sum_{i=1}^m a_i x_i - \delta\left(x_i + \gamma\right) + a_{m+1} \le 0$, 寻求最优 a_i, a_{m+1} 使得当前结点内的样本分类正确并更新;

  (3) Randomization; 因为在步骤 (2) 中每次更新 a_i 都是局部更新，因此在得到变量组合分割的超平面之后会进行扰动优化，生成随机向量 $(r_1, r_2, \ldots, r_{m+1})$ 加入超平面使得 $H_1 = \sum_{i=1}^m \left(a_i + \alpha r_i\right) x_i + \left(a_{m+1} + r_{m+1}\right)$ 再次优化;

- Recursive Least Square procedure

$$\begin{aligned}W_k &= W_{k-1} - K_k\left(X_k^T W_{k-1} - y_k\right) \\ K_k &= P_k K_k \\ P_k &= P_{k-1} - P_{k-1} X_k \left[1 + X_k^T P_{k-1} X_k\right]^{-1} X_k^T P_{k-1}\end{aligned}$$

- Feature Selection

  特征选取和回归中的 stepwise 类似采取 SBE(Sequential Backward Elimination) 和 SFE(Sequential Forward Selection) 等, 以 SBE 为例，对具有 n 个属性的组合寻求最优参数, 接下来依次删除其中一个属性，选择精度提高最多的一个属性进行剔除，以此类推，再用独立训练集验证;

1.3 叶结点生成

决策树模型的最终目的是产生叶结点并进行预测，而在学习过程中产生叶结点的方式可分为以下几种形式：

- 当前结点所含样本全部属于同一类别；
- 当前结点属性值为空，没有其他属性进行划分，归类为结点中最多的所属类别；
- 当前结点包含样本为空，样本分布参照父结点的样本分布；
- 预剪枝。

1.4 决策树剪枝

机器学习分类方法目的主要是为了提取某一类型样本的一般性特征，提高模型泛化识别能力，而非单一样例的个别特征，因此当决策树模型过深时，所产生的树模型往往会学习到个别特征，因此树模型一般需要进行剪枝处理来防止过拟合现象。

- 预剪枝 (阈值)

  预剪枝即在模型划分前对结点进行估计计算衡量指标，若不能提升则停止进行划分并标记为叶结点，预剪枝的优点在于模型训练的成本降低，但决策树训练时很多分支未展开导致存在欠拟合现象。预剪枝一般有以下几种形式:

  - 设定决策树最大深度；
  - 节点中的样本个数小于阈值；
  - 结点分裂的信息增益小于阈值。

- 后剪枝 (损失函数)

  后剪枝即在模型训练生成完整一决策树时从下而上，对非叶结点进行替换为叶结点计算模型的泛化能力，若有提升则将将该结点替换为叶结点进行。从原理上来看，预剪枝寻求局部优化，而后剪枝以成本开销为代价寻求全局优化。预剪枝一般在模型构建中使用阈值来处理，而后剪枝一般使用树损失函数来进行剪枝。

1.5 决策树算法

决策树算法主要分为二叉树与多叉树模型，前者代表有 CART 和 QUEST 算法，后者有 FACT、C4.5、CHAID、FRIM 算法等。

Decision Tree Pseudo Code

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i, y_i) \,|\, i = 1, 2, \ldots, n;\ y_i = 1, 2, ..., k\}$;
Attributes $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m\}$, Threshold ϵ.
Procedure: TreeGenerate($\mathbf{D}$, $\mathbf{A}$)
Generate a note $\mathbb{N}$
if all instance in $\mathbb{N}$ belong to a class j:
Label node $\mathbb{N}$ as Leaf Node of class j
elif $A = \emptyset$ or values of $\mathbf{x}_i's$ in $\mathbb{N}$ are same:
Label node $\mathbb{N}$ as Leaf Node of the class which has the most instance in $\mathbb{N}$
Select a optimal partitioning attributes $\mathbf{a}_*$
if Gain($\mathbb{N}$, $\mathbf{a}_*$) $< \epsilon$:
pass

for value a_*^v in $\mathbf{a}_*$:
Generate branch for $\mathbb{N}$, denote $\mathbb{N}_v$ as subsample of $\mathbb{N}$ which have value a_*^v
if $\mathbb{N}_v$ is empty:
Label node $\mathbb{N}$ as Leaf Node of class j
else:
Denote TreeGenerate($\mathbb{N}_v$, $\mathbf{A}/\mathbf{a}_*$) as child node;
Output: An Decision Tree $\mathbb{T}$

1.5.1 ID3

最早的决策树类型，采用信息增益指标进行评估和特征选择，也较为粗糙。

- 最优属性分割指标：信息增益；
- 连续属性处理：无法处理;
- 缺失值处理：无法处理；
- 剪枝：未进行后剪枝处理;
- 适用：数据存于内存中，小规模数据集。

FIRM 算法则是在 ID3 算法上每个连续性属性划分为 10 个属性在进行树的生成, 也不进行剪枝。

1.5.2 C4.5

C4.5 决策树算法继承 ID3 决策树并进行改进：

- 最优属性分割指标：**结合信息增益和信息增益率进行属性划分选择，避免偏好取值多的属性；**；
- 属性处理：二分法对连续属性离散化处理，对离散属性则采用多分法;
- 缺失值处理：根据后验概率分配到子结点；
- 剪枝：采用悲观剪枝法 (Pessimistic Error Pruning) 悲观剪枝法主要是在决策树学习过程中通过训练集在结点分裂前后考察决策树的修正误差作为指标，决定是否生成该节点，其中修正误差为决策树的误判样本数目加上叶结点数目作为惩罚项，可近似将决策树修正误差当作二项分布来考察：

  1. 定义 $\|\mathbb{T}\|$ 为决策树 $\mathbb{T}$ 的叶结点数目，$\epsilon(\mathbb{T})$ 为决策树的误判样本数目；
  2. 对任意内部结点 $\mathbb{N}$, 包含样本个数为 $|\mathbb{N}|$, 假定该结点根据最优属性划分生成子树为 $\mathbb{T}_{\mathbb{N}}$；
  3. 计算将结点 $\mathbb{N}$ 作为叶结点的模型修正错误率：

$$\epsilon^{'}(\mathbb{N}) = \epsilon(\mathbb{N}) + \frac{\|\mathbb{N}\|}{2} = \epsilon(\mathbb{N}) + \frac{1}{2}$$

  4. 计算将结点 $\mathbb{N}$ 作为内部结点即子树 $\epsilon(\mathbb{T})$ 的错误率以及标准差：

$$\epsilon^{'}(\mathbb{T}_{\mathbb{N}}) = \epsilon(\mathbb{T}_{\mathbb{N}}) + \frac{\|\mathbb{T}_{\mathbb{N}}\|}{2}$$

$$SE[\epsilon^{'}(\mathbb{T}_{\mathbb{N}})] = \sqrt{\frac{\epsilon^{'}(\mathbb{T}_{\mathbb{N}})\left(|\mathbb{N}| - \epsilon^{'}(\mathbb{T}_{\mathbb{N}})\right)}{|\mathbb{N}|}}$$

  5. 对比结点分裂前后的误差：

$$G(\mathbb{N}) = \epsilon^{'}(\mathbb{T}_{\mathbb{N}}) + SE[\epsilon^{'}(\mathbb{T}_{\mathbb{N}})] - \epsilon^{'}(\mathbb{N})$$

  若 $G(\mathbb{N}) \geq 0$ 则进行剪枝，将内部结点 $\mathbb{N}$ 作为叶结点，反之则不剪枝。

- 适用：在构造树时需要对数据的属性进行多次扫描和排序，占内存、低效。

1.5.3 CART

CART 决策树为分类与回归树，假定决策树是二叉树，内部结点特征为是和否，递归二分每个特征，在进行回归时使用叶结点的样本的均值输出预测值；

- 最优属性分割指标：使用 GINI 系数进行属性划分；
- 属性处理：对于连续属性采用多次二分法，离散属性划分时每次选择其中一种分割数据，对属性特征重复性使用；
- 对于多分类目标，会将目标类别合并成两个超类别；
- 对于树的构建，CART 算法会生成尽量大的数目，而后用验证数据集根据损失函数对已生成的树进行剪枝，选择最优子树；
- 剪枝：采用独立的验证集剪枝处理，否则剪枝结果为深度最大的子树;

  1. 定义决策树 $\mathbb{T}$ 的损失函数 $\ell(\mathbb{T})$：

$$\ell_\alpha(\mathbb{T}) = \ell(\mathbb{T}) + \alpha\|\mathbb{T}\| = \sum_{t=1}^{\|\mathbb{T}\|} |\mathbb{N}_t| H_{\mathbb{N}_t} + \alpha\|\mathbb{T}\|$$
$$H_{\mathbb{N}} = -\sum_k \frac{|\mathbb{N}^k|}{|\mathbb{N}|} \log \frac{|\mathbb{N}^k|}{|\mathbb{N}|}$$

  α 为树模型复杂度的惩罚系数, $\|\mathbb{T}\|$ 为决策树叶结点数目, $|\mathbb{N}|$ 为叶结点 $\mathbb{N}$ 上的样本数，$|\mathbb{N}^k|$ 为叶结点 $\mathbb{N}$ 上为类 k 的样本数，$H_{\mathbb{N}}$ 为叶结点 $\mathbb{N}$ 上的经验熵

  2. 对整棵树分别计算以任意内部结点 $\mathbb{N}$ 为单结点的树 $\mathbb{N}$ 和根节点的树 $\mathbb{T}_{\mathbb{N}}$ 的损失

$$\ell_\alpha(\mathbb{N}) = \ell(\mathbb{N}) + \alpha\|\mathbb{N}\| = \ell(\mathbb{N}) + \alpha, \quad \ell_\alpha(\mathbb{T}_{\mathbb{N}}) = \ell(\mathbb{T}_{\mathbb{N}}) + \alpha\|\mathbb{T}_{\mathbb{N}}\|$$

  3. 当且仅当 $\ell_\alpha(\mathbb{N}) \le \ell_\alpha(\mathbb{T}_{\mathbb{N}})$ 时，决策树会进行剪枝剪除 $\mathbb{T}_{\mathbb{N}}$，将 $\mathbb{N}$ 作为叶结点；
  4. 令 $G(\mathbb{N}) = \frac{\ell_\alpha(\mathbb{N}) - \ell_\alpha(\mathbb{T}_{\mathbb{N}})}{\|\mathbb{T}_{\mathbb{N}}\| - 1}$，计算所有内部结点 $\mathbb{N}$ 的 $G(\mathbb{N})$ 值，依次增加 α，自下而上递归在决策树 $\mathbb{N}_i$ 中剪除 $G(\mathbb{N}_i)$ 最小的 $\mathbb{T}_{\mathbb{N}^i}$ 得到 $\mathbb{N}_{i+1}$ 子树；
  5. 利用独立测试集计算独立子树集合 $\{\mathbb{N}_i\}$ 的损失函数 $\ell(\mathbb{N}_i)$，取最优子树 $\mathbb{N}_i^*$。

- Scikitlearn 模块中使用的决策树算法为 CART 算法：

  DecisionTreeClassifier(criterion = *gini*, splitter = *best*, max_depth = *None*,
  min_samples_split = 2, min_samples_leaf = 1, min_weight_fraction_leaf = 0,
  max_features = *None*, random_state = *None*, max_leaf_nodes = *None*,
  min_impurity_decrease = 0.0, min_impurity_split = *None*,
  class_weight = *None*, presort = *False*)
  多数指标都是针对剪枝进行改进。

- **CART 回归树**

  主要先将决策树的输出空间划分为 M 个空间 $R_1, R_2, \ldots, R_m$, 每个空间对应输出值 $c_1, c_2, \ldots, c_m$，而在对属性 $\mathbf{a}$ 选择和切分点 a^v 时不是选择信息增益或基尼指数，而是最小化:

$$\min_{\mathbf{a},\, a^v} \left[\min_{c_1} \sum\nolimits_{x_i \in \{x | x^a \le a^v\}} (y_i - c_1)^2 + \min_{c_2} \sum\nolimits_{x_i \in \{x | x^a > a^v\}} (y_i - c_2)^2 \right]$$

  其中 c_i 在平方误差损失函数下一般取值为 R_i 区间内样本的均值。

2 Bagging

2.1 RandomTreesEmbedding

Bagging 结合 Bootstrap，使得模型有部分验证集对组合学习器进行包外估计，可以起到降低方差的效果，不易受干扰。

$$\begin{aligned}\text{Dataset D} &\Rightarrow \text{Bootstrap m subsets}\{D_1, D_2, \ldots, D_m\} \\ &\Rightarrow \text{Train m learner}\,\{\mathbb{H}_i(\mathbf{x},\, i = 1, 2, \ldots, m)\} \\ &\Rightarrow H(\mathbf{x}) = \arg\max \sum_{i=1}^{m} \mathbb{I}\left(\mathbb{H}_i(\mathbf{x}) = y\right)\end{aligned}$$

2.2 RandomForest

随机森林在 RandomTreesEmbedding 上基础上进行改进，在构建每克决策树过程中引入**随机属性选择**. 传统决策树在选择属性划分时候对当前结点的属性集合 (d) 中选择最优属性，而在随机森林中，先对当前结点的属性集合中随机选择 k(多数情况下为 $\log_2 d$) 个属性子集，然后再从属性子集中选择最优属性。

2.3 StackingStrategy

Average, WeightedAverage, Majority Vote, MetaLearnerStacking,...

3 Boosting

3.1 基本概念

- Boosting 基本流程: Boosting 是一种基于改变原始数据集分布的迭代串行组合算法，主要流程:
  (1) 从初始训练集用初始权重训练出一个基学习器 $\mathbb{H}_1$，得到训练集输出值 $h(\mathbf{x})$;
  (2) 根据基学习器的学习误差率 ϵ_1 调整更新训练样本权值 ω，使得误差率高的训练样本权值 ω 增加，并确定该学习器的权重 α_1;
  (3) 基于新的样本权值训练基学习器 $\mathbb{H}_2$;
  (4) 迭代步骤 2 和 3，直到基学习器数目到达预设数目 t，然后根据基学习器的权重 α 组合基学习器的训练结果输出;

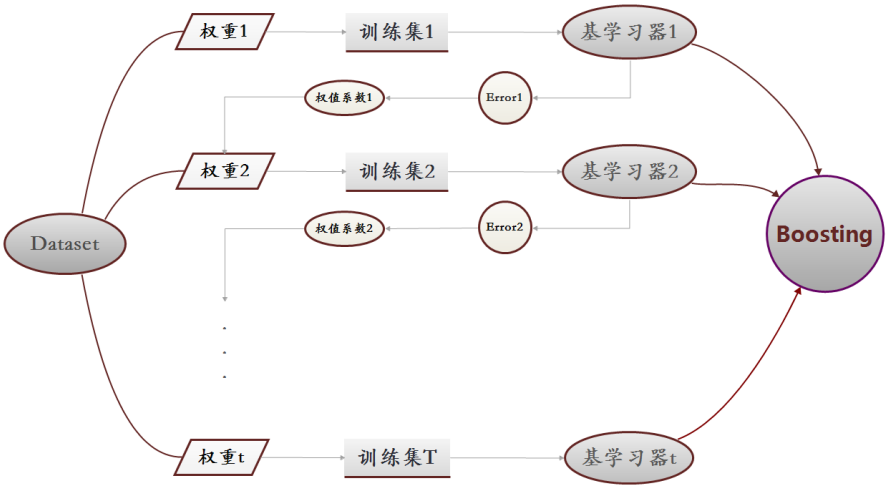

Figure 1: Boosting

所以不同的 Boosting 方法的区别在于以下几点:

- 学习误差率的计算方式即 Loss function 形式 ℓ;
- 如何根据误差率更新样本权重系数 ω;
- 如何确定基学习器的权重系数 α;

针对多数 Boosting 算法，基本都是采用加法模型组合弱分类器，用前向分布算法来更新训练数据的权重和概率分布进行模型学习。

- 前向分布算法前向分布算法是一个组合算法优化的一种求解方案，主要思路是通过从前向后每次学习一个基函数及其系数，逐步逼近优化目标函数来简化组合算法的优化复杂度。

(1) 考虑由 t 个基学习器组成的 Boosting 加法模型:

$$\hat{\mathbf{y}}^t = \sum_{i=1}^{t} \beta_i h^i(\mathbf{x})$$

(2) 解决这一优化问题可采用前向分布算法从前往后每次只学习一个基函数及其系数，逐步逼近优化目标函数来简化复杂度，即

$$Obj^{(t)} = \sum_{i=1}^{n} \ell\left(y_i, \hat{y}_i^t\right) + \sum_{i=1}^{t} \Omega(\mathbb{H}_i) = \sum_{i=1}^{n} \ell\left(y_i, \hat{y}_i^{t-1} + h_i^t(\mathbf{x})\right) + \Omega(\mathbb{H}_t)) + \text{Const.}$$

(3) 因此，最优化目标函数就转化为基于前一学习器的预测来最优化当前损失函数，即最小化

$$\sum_{i=1}^{n} \ell\left(y_i, \hat{y}_i^{t-1} + h_i^t(\mathbf{x})\right) + \Omega(\mathbb{H}_t)$$

(4) 以平方损失函数为例，则模型转为拟合上一轮的残差:

$$\sum_{i=1}^{n} \left(y_i - \hat{y}_i^{t-1} + h_i^t(\mathbf{x}_i)\right)^2 + \Omega(\mathbb{H}_t) = \sum_{i=1}^{n} \left(-\epsilon_i^{t-1} + h_i^t(\mathbf{x})\right)^2 + \Omega(\mathbb{H}_t)$$

(5) 而损失函数 ℓ 的形式及其泰勒展开项数的不同产生就衍生出不同的 Boosting 算法。

3.2 AdaBoost

AdaBoost 即 adaptive boosting，可视为损失函数为指数函数、学习算法为前向分步算法的二分类加法模型，顾名思义即通过调整训练样本的权值分布，为弱学习器定义不同的权值，将其线性组合成一个强学习器。

$$f(\mathbf{x}) = \sum_{i=1}^{t} \alpha_t h_t(\mathbf{x})$$
$$\ell_{\exp}(y, f(\mathbf{x})|\omega) = E_{\mathbf{x}\sim\omega}[\exp(-\mathbf{y}f(\mathbf{x}))]$$

之后经过发展衍生出用于解决多分类的 AdaBoost.m 系列算法以及用于回归问题的 AdaBoost.r 系列算法。

3.2.1 AdaBoost for Binary Classification

AdaBoost 的主要特点在于:

- 权值更新：$\omega_{t+1} \sim \omega_t \exp(-\alpha_t \mathbf{y} f(\mathbf{x}))$
  即若分类正确权值同比于上一轮的 $\exp(-\alpha)$，若分类错误权值增大为上一轮 $\exp(\alpha)$；
- 基学习器权值与样本权值更新系数：$\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$
  单个基学习器分类误差率越低，权值越高。

Algorithm: AdaBoost for Binary Classification

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i,\, y_i)\,|\,i = 1, 2, \ldots, n;\, y_i = -1, 1\}$
WeakLearner $\mathbb{H}$, Loss function ℓ, Iterations T

Procedure:

$\omega_1(\mathbf{x}) = \frac{1}{n}$
for $t = 1, 2, \ldots, T$:
  $h_t(\mathbf{x}) = \mathbb{H}(\mathbf{D}, \omega)$
  $\epsilon_t = \ell(\mathbf{y}, h_t(\mathbf{x})) = P_{\mathbf{x}\sim\omega_t}(h_t(\mathbf{x}) \neq \mathbf{y})$
  if $\epsilon_t > 0.5$:
    break
  $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$
  $\omega_{t+1} = \frac{\omega_t(\mathbf{x})}{Z_t}\exp\left(-\alpha_t \mathbf{y} h_t(\mathbf{x})\right),\ Z_t = E_{\mathbf{x}\sim\omega_t}(\mathbf{x})\exp(-\alpha_t \mathbf{y} h_t(\mathbf{x}))$

Output: $f(\mathbf{x}) = \sum_{i=1}^{t} \alpha_t h_t(\mathbf{x})$

- 指数损失函数 $\ell(\mathbf{y}, f(\mathbf{x}))$ 最小化

$$
\begin{aligned}
\frac{\partial \ell_{\exp}(f(\mathbf{x})|\omega)}{\partial f(\mathbf{x})} &= \frac{\partial E_{\mathbf{x}\sim\omega}[\exp(-\mathbf{y} f(\mathbf{x}))]}{\partial f(\mathbf{x})} \\
&= \frac{\partial\left(\exp(-f(\mathbf{x}))P(\mathbf{y}=1|\mathbf{x}) + \exp(f(\mathbf{x}))P(\mathbf{y}=-1|\mathbf{x})\right)}{\partial f(\mathbf{x})} \\
&= -\exp(-f(\mathbf{x}))P(\mathbf{y}=1|\mathbf{x}) + \exp(f(\mathbf{x}))P(\mathbf{y}=-1|\mathbf{x}) = 0 \\
f(\mathbf{x}) &= \frac{1}{2}\ln\frac{P\left(f(\mathbf{x})=1|\mathbf{x}\right)}{P\left(f(\mathbf{x})=-1|\mathbf{x}\right)} \\
\operatorname{sign}(f(\mathbf{x})) &= \operatorname{sign}\left[\frac{1}{2}\ln\frac{P\left(f(\mathbf{x})=1|\mathbf{x}\right)}{P\left(f(\mathbf{x})=-1|\mathbf{x}\right)}\right] \\
&= \begin{cases} 1, & P(f(\mathbf{x})=1|\mathbf{x}) > P(f(\mathbf{x})=-1|\mathbf{x}) \\ -1, & P(f(\mathbf{x})=1|\mathbf{x}) < P(f(\mathbf{x})=-1|\mathbf{x}) \end{cases} \\
&= \arg\max_{y\in\{-1,1\}} P(f(\mathbf{x})=\mathbf{y})
\end{aligned}
$$

  也即指数损失函数最小化等同于二分类方法中最大化分类正确率。

- 误差率 ϵ_t 更新权值系数 α_t

$$
\begin{aligned}
\ell_{\exp}(\alpha_t h_t(\mathbf{x})|\omega_t) &= E_{\mathbf{x}\sim\omega_t}[\exp(-\alpha_t \mathbf{y} h_t(\mathbf{x})] \\
&= E_{\mathbf{x}\sim\omega_t}[\exp(-\alpha_t)\mathbb{I}(\mathbf{y}=h_t(\mathbf{x})) + \exp(\alpha_t)\mathbb{I}(\mathbf{y}\neq h_t(\mathbf{x}))] \\
&= \exp(-\alpha_t)(1-\epsilon_t) + \exp(\alpha_t)\epsilon_t \\
\frac{\partial \ell_{\exp}(\alpha_t h_t(\mathbf{x}))}{\partial \alpha_t} &= -\exp(-\alpha_t)(1-\epsilon_t) + \exp(\alpha_t)\epsilon_t = 0 \\
\alpha_t &= \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}
\end{aligned}
$$

- 前向分布算法推导 $h_t(\mathbf{x})$, ω_t

$$
\begin{aligned}
h_t(\mathbf{x}) &= \arg\min_h \ell_{\exp}\left(\mathbf{y}, f_{t-1}(\mathbf{x}) + \alpha_t h_t(\mathbf{x})|\omega\right) \\
&= \arg\min_h E_{\mathbf{x}\sim\omega}\left[\exp(-\alpha_t \mathbf{y} f_{t-1}(\mathbf{x}))\exp(-\mathbf{y}\alpha_t h_t(\mathbf{x}))\right] \\
&= \arg\min_h E_{\mathbf{x}\sim\omega}\left[\exp(-\alpha_t \mathbf{y} f_{t-1}(\mathbf{x}))\exp(-\mathbf{y}\alpha_t h_t(\mathbf{x}))\right] \\
&\equiv \arg\min_h E_{\mathbf{x}\sim\omega}\left[\exp\left(-\alpha_t \mathbf{y} f_{t-1}(\mathbf{x})\right)\left(1 - \mathbf{y}h_t(\mathbf{x}) + \frac{\mathbf{y}^2 h_t^2(\mathbf{x})}{2}\right)\right] \\
&= \arg\min_h E_{\mathbf{x}\sim\omega}\left[\exp(-\alpha_t \mathbf{y} f_{t-1}(\mathbf{x}))\left(1 - \mathbf{y}h_t(\mathbf{x}) + \frac{1}{2}\right)\right] \\
&= \arg\min_h E_{\mathbf{x}\sim\omega}\left[\frac{\exp(-\alpha_t \mathbf{y} f_{t-1}(\mathbf{x}))}{E_{\mathbf{x}\sim\omega}[\exp(-\mathbf{y} f_{t-1}(\mathbf{x}))]}\mathbf{y}h_t(\mathbf{x})\right] \\
&= \arg\min_h E_{\mathbf{x}\sim\omega_t}[\mathbf{y}h_t(\mathbf{x})], \text{ where } \omega_t(\mathbf{x})|\omega = \frac{\omega\exp(-\alpha_t \mathbf{y} f_{t-1}(\mathbf{x}))}{E_{\mathbf{x}\sim\omega}[\exp(-\alpha_t \mathbf{y} f_{t-1}(\mathbf{x}))]} \\
&= \arg\min_h E_{\mathbf{x}\sim\omega_t}[1 - 2\mathbb{I}(\mathbf{y} \neq h_t(\mathbf{x}))] \\
&= \arg\max_h E_{\mathbf{x}\sim\omega_t}[\mathbb{I}(\mathbf{y} = h_t(\mathbf{x})] \\
\omega_{t+1} &= \frac{\omega_t \exp(-\alpha_t \mathbf{y} f_t(\mathbf{x}))}{E_{\mathbf{x}\sim\omega_t}[\exp(-\alpha_t \mathbf{y} f_t(\mathbf{x}))]} \\
&= \omega_t \exp(-\alpha_t \mathbf{y} h_t(\mathbf{x}))\frac{E_{\mathbf{x}\sim\omega_t}[\exp(-\alpha_t \mathbf{y} f_{t-1}(\mathbf{x}))]}{E_{\mathbf{x}\sim\omega_t}[\exp(-\alpha_t \mathbf{y} f_t(\mathbf{x}))]} \\
&= \omega_{t-1}\frac{\exp(-\alpha_t \mathbf{y} f_t(\mathbf{x}))\exp(-\alpha_{t-1} \mathbf{y} f_{t-1}(\mathbf{x}))}{Z_t Z_{t-1}} = \cdots
\end{aligned}
$$

AdaBoost 作为弱分类器的组合器，基分类器越弱，效果提升越明显，不容易过拟合，且基于 AdaBoost 框架可以拓展回归和多分类问题。但由于 AdaBoost 的权值更新方式的原理，AdaBoost 对异常数据比较敏感，因异常数据在不断迭代中往往会获得较高的权值，最终影响模型预测准确性。

3.2.2 AdaBoost.M for Multiple Classification

AdaBoost 基本算法仅应用于二分类方法，因此在面对多分类问题需要在权重更新方式上进行改进。

AdaBoost.m1 for Multiclass Classification

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i, y_i)\,|\,i = 1, 2, \ldots, n;\ y_i = 1, 2, \ldots, k\}$
WeakLearner $\mathbb{H}$, Loss function ℓ, Iterations T

Procedure:

$\omega_1(\mathbf{x}) = \frac{1}{n}$
for $t = 1, 2, \ldots, T$:
  $h_t(\mathbf{x}) = \mathbb{H}(\mathbf{D}, \omega)$
  $\epsilon_t = \ell(\mathbf{y}, h_t(\mathbf{x})) = P_{\mathbf{x}\ \omega_t}(h_t(\mathbf{x}) \neq \mathbf{y})$
  if $\epsilon_t > 0.5$:
    break
  $\alpha_t = \ln\frac{1-\epsilon_t}{\epsilon_t}$
  $\omega_{t+1} = \frac{\omega_t}{Z_t}\exp(-\alpha_t(1 - [h_t(\mathbf{x}) \neq \mathbf{y}]))$

Output: $f(\mathbf{x}) = \sum_{t=1}^{T}\alpha_t[h_t(\mathbf{x}) = \mathbf{y}]$

从算法中可以看出 AdaBoost.m1 仅在权值更新时对判断正确的样本调低权重为原有的 $\exp(-\alpha_t)$，而对于判断错误的样本权值不进行更新，从而增加误分类的权值。与 AdaBoost 相比较而言：

- 样本权值放缩权值降低，二分类中以 $\exp(2\alpha_t)$ 的扩张系数更新误判样本的权值，而 AdaBoost.m1 以 $\exp(\alpha_t)$ 的系数增加；
- AdaBoost.m1 仅考虑样本是否分类错误，但不考虑被误判为其他一类别，而二分类中由于只有两个类别不存在此缺陷；
- AdaBoost.m1 也会受到异常值的影响，特别是多分类不平衡的数据难以产生好的效果。

3.2.3 AdaBoost.M2 for Multiple Classification

AdaBoost.M2 则是基于 AdaBoost.M1 的基础上进行改进，比较 AdaBoost.M1 只考虑是否分类正确而言，该更新权值考虑两方面：

- 先是根据判断样本是否分类正确，从而调整权值；
- 再是根据错误样本误判为其他类别，调整其他类别样本的权重；

AdaBoost.m2 for Multiclass Classification

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i,\, y_i)\,|\, i = 1, 2, \ldots, n;\, y_i = 1, 2, \ldots, k\}$
WeakLearner $\mathbb{H}$, Loss function ℓ, Iterations T

Procedure:

$\Omega^1 = \frac{1}{n}$
$\omega^1_{i,y} = \frac{\Omega^1}{k-1}$ for $i = 1, 2, \ldots, n$ and $y \neq y_i$
for i = 1,2,...,T:
$\mathbf{w}^t_i = \sum_{y \neq y_i} \omega^t_{i,y}$
$q^t_{i,y} = \frac{\omega^t_{i,y}}{\mathbf{w}^t_i}$ for $y \neq y_i$
$\Omega_t = \frac{\mathbf{w}^t_i}{\sum_{i=1}^n \mathbf{w}^t_i}$
$h^t(\mathbf{x}, \mathbf{y}) = \mathbb{H}(\mathbf{D}, \Omega_t)$
$\epsilon_t = \frac{1}{2}\sum_{i=1}^n \Omega^t_i(1 - h^t_i(\mathbf{x}, y_i) + \sum_{i,y \neq y_i} q^t_{i,y} h^t_i(\mathrm{x}, y))$
if $\epsilon_t > 0.5$:
break
$\alpha_t = \ln \frac{1-\epsilon_t}{\epsilon_t}$
$\omega^{t+1}_{i,y} = \omega^t_{i,y} \exp(-\frac{1}{2}\alpha_t(1 + h_t(\mathbf{x}, y_i) - h_t(\mathbf{x}, y)))$ for $y \neq y_i$

Output: $f(\mathbf{x}) = \sum_{t=1}^T \alpha_t h_t(\mathbf{x}, y)$

算法中的符号含义：

- Ω^t：表示第 t 个学习器输入的训练数据样本的权值，大小为 $n \times 1$；
- $\omega^t_{i,y}$：表示第 t 个学习器输入的第 i 个样本里若被错判为其他类别的损失权重；
- $\mathbf{w}^t_i$：表示不属于 y_i 类别的样本权重之和，这里是为了 Normalize 权重；
- $q^t_{i,y}$：Normalized $\omega^t_{i,y}$。

算法中学习器 $\mathbb{H}(\mathbf{D}, \Omega_t)$ 输出一个 $n \times k$ 浮点数矩阵而不是 AdaBoost 中的类别，而在计算误差率时要根据被错判的类别 $h^t(\mathrm{x}_i, y_i)$、易被错归类的类别 $h^t(\mathrm{x}_i, y)$ 及其损失权重 $q^t_{i,y}$ 计算总损失双向更新权重。

3.2.4 AdaBoost.R1 for Regression

AdaBoost.R1 则是基于 AdaBoost 算法应用于回归问题的改进。

AdaBoost.R for Regression

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i,\, y_i)\,|\,i = 1, 2, \ldots, n;\, y_i \in [0,\, 1]\}$
WeakLearner $\mathbb{H}$, Loss function ℓ, Iterations T

Procedure:

$\Omega_i^1 = \frac{1}{n}$
$\omega_{i,y}^1 = \frac{\Omega_i^1 * |y - y_i|}{Z_1}$, where $Z_1 = \sum_{i=1}^n \Omega_i^1 \int_0^1 |y - y_i| dy$
for $t = 1, 2, \ldots, T$:
  $\Omega^t = \frac{\omega^t}{\sum_{i=1}^n \int_0^1 \omega_{i,y}^t dy}$
  $h^t(\mathbf{x}) = \mathbb{H}(\mathbf{D}, \Omega_t)$
  $\epsilon_t = \sum_1^n \left| \int_{y_i}^{h^t(\mathbf{x}_i)} \Omega_{i,y}^t \right| dy$
  if $\epsilon_t > 0.5$:
    break
  $\alpha_t = \ln \frac{1-\epsilon_t}{\epsilon_t}$
  $\omega_{i,y}^{t+1} = \omega_{i,y}^t \exp(-\alpha[1 - \mathbb{I}(y \in [y_i, h_t(\mathbf{x}_i)] \text{ or } [h_t(\mathbf{x}_i), y_i])])$

Output: $f(\mathbf{x}) = \inf\{y : \sum_{t:h_t(\mathbf{x}) \le y} \alpha_t \ge \frac{1}{2} \sum_t \alpha_t\}$

主要思想为：

- 初始化损失权重：每个样本定义权重向量，权值正比于该样本与其他样本的回归变量的距离，作为初始的损失权重，若初始绝对误差越大的样本被归类为同一输入空间则误差越大；
- 权值更新：每个样本对应的权重向量单独扩大或缩小 $\exp(\alpha)$，而决定增减方向的判别为权重向量对应的其他样本的输出值是否落于该样本预测值与真实值的区间内；

3.2.5 AdaBoost.R2 for Regression

AdaBoost.R2 和 AdaBoost.R1 的不同点在于误差的衡量方式不同，AdaBoost.R1 与整体误差均值做比较，而 AdaBoost.R2 与最大误差作比较，而在更新 1 权值方式上 AdaBoost.R1 关注其他样本是否在某一样本的真实值与预测值区间内来更新权值，而 AdaBoost.R2 关注样本预测值与真实值的距离来调整权值。

AdaBoost.R2 for Regression

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i,\, y_i)\,|\,i = 1, 2, \ldots, n;\, y_i \in [0,\, 1]\}$
WeakLearner $\mathbb{H}$, Loss function ℓ, Iterations T

Procedure:

$\Omega_i^1 = \frac{1}{n}$
$\omega_{i,y}^1 = \frac{\Omega_i^1 |y - y_i|}{Z_1}$, where $Z_1 = \sum_{i=1}^n \Omega_i^1 \int_0^1 |y - y_i| dy$
for $t = 1, 2, \ldots, T$:
  $h_t = \mathbb{H}(\mathbf{D}, \Omega_t)$
  $Z^t = \max_{i \in \{1,2,\ldots,n\}} |y_i - h_t(\mathbf{x}_j)|$
  $e_i^t = \frac{|y_i - h_t(\mathbf{x}_j)|}{Z_t}$ for absolute loss
  $\epsilon_t = \sum_{i=1}^n e_i^t \omega_i^t$
  if $\epsilon_t > 0.5$:
    break
  $\alpha_t = \ln \frac{1-\epsilon_t}{\epsilon_t}$
  $\omega_{i,y}^{t+1} = \frac{\omega_{i,y}^t \exp(-\alpha_t(1-e_i^t))}{\sum \omega_{i,y}^t \exp(-\alpha_t(1-e_i^t))}$

Output: $f(\mathbf{x}) = median(\{\alpha_t h_t(\mathbf{x}),\, i = 1, 2, ...\})$

3.2.6 AdaBoost.RT for Regression

AdaBoost.RT 算法在 AdaBoost.R 的基础上计算误差项时会先设置一个阈值 ϕ，当一个样本的标准化数值与预测值的残差高于 ϕ 时会将其定义为误差，若低于阈值 ϕ 时则在计算学习器误差时忽略该项。对于残差小于阈值的样本权值缩小为原有的 $\exp(\alpha_t)$，而高于阈值的样本误差则样本权值不变，从而提高残差值较大的样本的相对权重，该权值更新方式与 AdaBoost.m1 类似。

AdaBoost.RT for Regression

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i,\, y_i) \,|\, i = 1, 2, \ldots, n;\, y_i \in [0,\, 1]\}$

WeakLearner $\mathbb{H}$, Loss function ℓ, Iterations T, threshold ϕ

Procedure:

$\omega_i^1 = \frac{1}{n}$

for $t = 1, 2, \ldots, T$:

$h_t = \mathbb{H}(\mathbf{D}, \omega_t)$

$\epsilon_t = \sum_{i=1}^{n} \omega_i^1 \mathbb{I}\left[\left|\frac{y_i - h_t(\mathbf{x}_j)}{y_i}\right| > \phi\right]$

$\alpha_t = -2 \ln \epsilon_t$

$\omega_i^{t+1} = \frac{\omega_i^t}{Z_t} \exp\left(\alpha_t \mathbb{I}\left[\left|\frac{y_i - h_t(\mathbf{x}_j)}{y_i}\right| > \phi\right]\right)$

Output: $f(\mathbf{x}) = \frac{\sum_{i=1}^{t} \alpha_t h_t(\mathbf{x})}{\sum_{i=1}^{t} \alpha_t}$

3.3 Boosting Decision Tree

Boosting Decision Tree 是基于决策树的加法模型，优化方式仍是采取前向分布算法，在确定初始提升树之后对第 t 个学习器的模型学习方式为:

$$f_t(\mathbf{x}) = f_{t-1}(\mathbf{x}) + h_t(\mathbf{x}) = f_{t-1}(\mathbf{x}) + \mathbb{H}(\mathbf{x}, \hat{\Theta}_t)$$

$$\hat{\Theta}_t = \arg\min_{\Theta_t} \sum_{i=1}^{n} \ell(y_i, f_{t-1}(\mathbf{x}_i) + \mathbb{H}(\mathbf{x}_i, \Theta_t))$$

提升树算法可用于分类或回归，默认提升树采用平方误为损失函数，因此根据目标函数的倒数可知新一轮迭代的目标是拟合之前拟合结果的残差，即:

Boosting Decision Tree

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i,\, y_i) \,|\, i = 1, 2, \ldots, n\}$

WeakLearner $\mathbb{H}$, Loss function ℓ, Iterations T

Procedure:

$f_0(x) = 0$

for $t = 1, 2, \ldots, T$:

$\epsilon_t = \mathbf{y} - f_{t-1}(\mathbf{x})$

$h_t(\mathbf{x}) = \mathbb{H}(\mathbf{x}, \hat{\Theta}_t)$ 拟合残差 ϵ_t

Update $f_t(\mathbf{x}) = f_{t-1}(\mathbf{x}) + h_t(\mathbf{x})$

Output: Regression Tree $f(\mathbf{x}) = \sum_{i=1}^{T} h_i(\mathbf{x})$

3.4 Gradient Boosting Decision Tree

Gradient Boosting 是一种 Boosting 方法，其主要思想是每一次建立模型是在之前模型目标函数的梯度下降方向。目标函数函数是模型性能的评价，一般包括损失函数以及正则项，目标函数越低，性能越好。Gradient Boosting 主要是随着基学习器的增加使得模型的性能不断改进，对应的就是让目标函数沿着负梯度方向下降，因在负梯度方向上下降速度最快。

事实上 Boosting Decision Tree 就是 Gradient Boosting 在目标函数为平方误差时的一个特例，此时目标函数的负梯度就是模型拟合的残差。但若损失函数改变，则 Boosting Decision Tree 算法不在适用，而 GBDT 就是基于不同的损失函数上，先将损失函数进行泰勒一阶展开：

$$f(x+\Delta x) = f(x) + \frac{\partial f(x)}{\partial x}\Delta x$$

再利用 Gradient Desecent 方法展开目标函数进行优化，即：

$$\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} \ell(y_i, f_t(\mathbf{x}_i)) \\
&= \sum_{i=1}^{n} \ell(y_i, f_{t-1}(\mathbf{x}_i) + h_t(\mathbf{x}_i)) \\
&= \sum_{i=1}^{n} \left[\ell(y_i, f_{t-1}(\mathbf{x}_i)) + \frac{\partial \ell(y_i, f_{t-1}(\mathbf{x}_i)}{\partial f_{t-1}(\mathbf{x}_i)} h_t(\mathbf{x}_i) \right] \\
&= Obj_i^{(t-1)} + \sum_{i=1}^{n} \frac{\partial \ell(y_i, f_{t-1}(\mathbf{x}_i))}{\partial f_{t-1}(\mathbf{x}_i)} h_t(\mathbf{x}_i) \\
\frac{\partial Obj^{(t)}}{\partial h_t(\mathbf{x}_i)} &= \frac{\partial \ell(y_i, f_{t-1}(\mathbf{x}_i))}{\partial f_{t-1}(\mathbf{x}_i)} \\
Obj^{(t)} \le Obj^{(t-1)} &\Leftrightarrow h_t(\mathbf{x}_i) = -\eta \frac{\partial \ell(y_i, f_{t-1}(\mathbf{x}_i))}{\partial f_{t-1}(\mathbf{x}_i)}, \forall \eta > 0
\end{aligned}$$

目标函数 $Obj^{(t)}$ 的梯度上升方向为 $\frac{\partial \ell(y_i, f_{t-1}(\mathbf{x}_i))}{\partial f_{t-1}(\mathbf{x}_i)}$，只需令 $h_t(\mathbf{x})$ 沿着目标函数负梯度方向拟合即可提高模型的性能，η 为我们说是的步长，其决定模型的收敛速度和精度。

Gradient Boosting Decision Tree

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i,\, y_i) \,|\, i = 1, 2, \ldots, n\}$

WeakLearner $\mathbb{H}$, Loss function ℓ, Iterations T

Procedure:

$f_0(x) = 0$

for $t = 1, 2, \ldots, T$:

$\epsilon_t = -[\frac{\partial \ell(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}]'_{f=f_{t-1}}$

Train $\mathbb{H}(\mathbf{x}, \hat{\Theta}_t)$ to fit ϵ_t, obtain the leaf nodes of $\mathbb{H}(\mathbf{x}, \hat{\Theta}_t)$: $\{R_j^t;\, j = 1, 2, \ldots, J\}$

for $j = 1, 2, \ldots, J$:

Calculate the optimal output for each leaf node

$c_j^t = \arg\min_c \sum_{x_i \in R_j^t} \ell(f_{t-1}(\mathbf{x}_i), c)$

$\alpha_t = \arg\min_\alpha \sum_i^n \ell(y_i, f_{t-1}(\mathbf{x}_i) + \alpha h_t(\mathbf{x}_i))$

Update $f_t(\mathbf{x}) = f_{t-1}(\mathbf{x}) + \alpha_t h_t(\mathbf{x})$;

Output: Regression Tree $f(\mathbf{x}) = \sum_{i=1}^{T} \alpha_t h_i(\mathbf{x})$

从算法来看 GBDT 与 Boosting Decision Tree 的区别：

- Boosting Decision Tree 是 GBDT 在损失函数为平方误差的特例，Boosting Decision Tree 只需要

拟合残差即可；

- 由于平方误差的一阶段为一元线性代数，因此 Boosting Decision Tree 的叶结点输出在平方误差下只需要取均值即可，且每个基学习器求和就是最小化初始 $\mathbf{y}$，因此不需要将叶结点的输出 c_j^t 及基学习器的权重 α_t 额外求解一次，但 GBDT 随着损失函数的不同叶结点的输出也不同，需要单独求解叶结点 c_j^t。
- 要实现目标函数尽可能下降，GBDT 中基学习器的最优属性分裂为选取方差增益最高的属性 $\mathbf{a}^*$：

$$\begin{aligned}\mathbf{a}^* &= \arg\max_{\mathbf{a}\in\mathbf{A}} \text{SplitGain}(\mathbb{N}, \mathbf{a}) \\ &= \arg\max_{\mathbf{a}} \frac{1}{\|\mathbb{N}\|}\left(\frac{\sum_{\mathbb{N}_L}[\frac{\partial \ell(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}]^2}{\|\mathbb{N}_L\|} + \frac{\sum_{\mathbb{N}_R}[\frac{\partial \ell(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}]^2}{\|\mathbb{N}_R\|}\right) \\ &\equiv \arg\max_{\mathbf{a}} \frac{1}{\|\mathbb{N}\|}\left(\frac{\sum_{\mathbb{N}_L}[g_L(\mathbf{x}_i)]^2}{\|\mathbb{N}_L\|} + \frac{\sum_{\mathbb{N}_R}[g_R(\mathbf{x}_i)]^2}{\|\mathbb{N}_R\|}\right)\end{aligned}$$

3.5 XGBoost

XGBoost 作为 Kaggle 竞赛比较吃香的模型，是 GBDT 的一个拓展，主要在目标函数正则化、损失函数展开、贪心分裂和预排序等进行改进。

(2) 目标函数正则化

XGBoost 算法在目标函数中加入正则化项，当基学习器为 CART 树时，正则化项与树的叶子节点的数目和叶子节点的值相关：

$$\begin{aligned}Obj^{(t)} &= \sum_i^n \ell(y_i, f_{t-1}(\mathbf{x}_i) + h_t(\mathbf{x}_i)) + \sum_{i=1}^t \Omega(\mathbb{H}_i) \\ &= \sum_i^n \ell\left(y_i, f_{t-1}(\mathbf{x}_i) + h_t(\mathbf{x}_i)\right) + \sum_{i=1}^t \left(\alpha|\mathbb{H}_i| + \frac{1}{2}\lambda\|R_j^t\|^2\right) \\ &= \sum_i^n \ell\left(y_i, f_{t-1}(\mathbf{x}_i) + h_t(\mathbf{x}_i)\right) + \sum_{i=1}^t \left(\alpha|\mathbb{H}_i| + \frac{1}{2}\lambda\sum_{j=1}^J (c_j^t)^2\right)\end{aligned}$$

其中 $|\mathbb{H}_t|$ 为树 $\mathbb{H}_t$ 的叶结点个数，$\|R_j^t\|^2$ 为 $|\mathbb{H}_t|$ 叶结点 j 的 L_2 范数，c_j^t 为 $|\mathbb{H}_t|$ 叶结点 j 的输出值。

(2) 损失函数泰勒二阶展开优化

GBDT 是对损失函数上利用泰勒一阶展开的梯度优化算法，而 XGBoost 则对损失函数利用牛顿法

对目标函数进行二阶泰勒展开而进行的优化：

$$
\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} \ell(y_i, f_{t-1}(\mathbf{x}_i) + h_t(\mathbf{x}_i)) + \sum_{i=1}^{t} \Omega(\mathbb{H}_i) \\
&= \sum_{i=1}^{n} \ell(y_i, f_{t-1}(\mathbf{x}_i) + h_t(\mathbf{x}_i)) + \Omega(\mathbb{H}_t) + \sum_{i=1}^{t-1} \Omega(\mathbb{H}_i) \\
&= \sum_{i=1}^{n} \left[\ell(y_i, f_{t-1}(\mathbf{x}_i)) + \frac{\partial \ell(y_i, f_{t-1}(\mathbf{x}_i)}{\partial f_{t-1}(\mathbf{x}_i)} h_t(\mathbf{x}_i) + \frac{1}{2} \frac{\partial^2 \ell(y_i, f_{t-1}(\mathbf{x}_i)}{\partial f_{t-1}^2(\mathbf{x}_i)} h_t^2(\mathbf{x}_i) \right] + \Omega(\mathbb{H}_t) + \sum_{i=1}^{t-1} \Omega(\mathbb{H}_i) \\
&= \sum_{i=1}^{n} \left[\ell(y_i, f_{t-1}(\mathbf{x}_i)) + \frac{\partial \ell(y_i, f_{t-1}(\mathbf{x}_i)}{\partial f_{t-1}(\mathbf{x}_i)} h_t(\mathbf{x}_i) + \frac{1}{2} \frac{\partial^2 \ell(y_i, f_{t-1}(\mathbf{x}_i)}{\partial f_{t-1}^2(\mathbf{x}_i)} h_t^2(\mathbf{x}_i) \right] + \alpha |\mathbb{H}_t| + \frac{1}{2} \lambda \|R_j^t\|^2 + \sum_{i=1}^{t-1} \Omega(\mathbb{H}_i) \\
&= \sum_{i=1}^{n} \left[\ell(y_i, f_{t-1}(\mathbf{x}_i)) + \frac{\partial \ell(y_i, f_{t-1}(\mathbf{x}_i)}{\partial f_{t-1}(\mathbf{x}_i)} h_t(\mathbf{x}_i) + \frac{1}{2} \frac{\partial^2 \ell(y_i, f_{t-1}(\mathbf{x}_i)}{\partial f_{t-1}^2(\mathbf{x}_i)} h_t^2(\mathbf{x}_i) \right] + \alpha |\mathbb{H}_t| + \frac{1}{2} \lambda \sum_{j=1}^{J} (c_j^t)^2 + \sum_{i=1}^{t-1} \Omega(\mathbb{H}_i) \\
&= Obj^{(t-1)} + \sum_{i=1}^{n} \left[\frac{\partial \ell(y_i, f_{t-1}(\mathbf{x}_i)}{\partial f_{t-1}(\mathbf{x}_i)} h_t(\mathbf{x}_i) + \frac{1}{2} \frac{\partial^2 \ell(y_i, f_{t-1}(\mathbf{x}_i)}{\partial f_{t-1}^2(\mathbf{x}_i)} h_t^2(\mathbf{x}_i) \right] + \alpha |\mathbb{H}_t| + \frac{1}{2} \lambda \sum_{j=1}^{J} (c_j^t)^2 \\
&\equiv Obj^{(t-1)} + \sum_{i=1}^{n} \left[g_i c_{q(i)}^t + \frac{1}{2} h_i (c_{q(i)}^t)^2 \right] + \alpha |\mathbb{H}_t| + \frac{1}{2} \lambda \sum_{j=1}^{J} (c_j^t)^2 \\
&= Obj^{(t-1)} + \sum_{i=1}^{t} \left[(\sum_{j \in R_i^t} g_j) c_i^t + \frac{1}{2} (\sum_{j \in R_i^t} h_j + \lambda) (c_i^t)^2 \right] + \alpha |\mathbb{H}_t| \\
&\equiv Obj^{(t-1)} + \sum_{i=1}^{t} \left[G_i c_i^t + \frac{1}{2} (H_i + \lambda) (c_i^t)^2 \right] + \alpha |\mathbb{H}_t|
\end{aligned}
$$

其中 $|\mathbb{H}_t|$ 为树 $\mathbb{H}_t$ 的叶结点个数，$\|R_j^t\|^2$ 为 $|\mathbb{H}_t|$ 叶结点 j 的 L_2 范数，c_j^t 为 $|\mathbb{H}_t|$ 叶结点 j 的输出值，$q(i)$ 表示样本 i 所属叶结点 q，G_i 表示叶结点 R_i 内的样本一阶导之和，H_i 为二阶导之和。在对梯度拟合求解最优得到

$$
c_j^* = -\frac{G_j}{H_j + \lambda} \quad \Rightarrow \quad Obj = -\frac{1}{2} \sum_{t=1}^{T} \frac{G_j^2}{H_j + \lambda} + \alpha |\mathbb{H}|
$$

(3) 贪心分裂与直方图估计分裂

XGBoost 和 GBDT 分裂时选择最优属性方式类似，基于目标函数寻找分割点的标准是最大化分裂收益。

$$
\begin{aligned}
\text{obj\_before\_split} &= -\frac{1}{2} \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} + \alpha \\
\text{obj\_after\_split} &= -\frac{1}{2} \left[\frac{(G_L)^2}{H_L + \lambda} + \frac{(G_R)^2}{H_R + \lambda} \right] + 2\alpha \\
\text{node\_obj\_reduction} &= \frac{1}{2} \left[\frac{(G_L)^2}{H_L + \lambda} + \frac{(G_R)^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \alpha
\end{aligned}
$$

因此 XGBoost 的结点分裂最优属性为寻找 *node_obj_reduction* 值最大的属性，XGBoost 一般采用贪心法 greedy algorithm 枚举所有可能的分割点计算每个特征的每个分割点对应的增益，从中寻找最优分割点，进而寻找最优属性：

Optimal Splits Search: Greedy Algorithm

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i, y_i);\ i = 1, 2, \ldots, n\}$

Attribute Set $\mathbf{A} = \{\mathbf{a}_i;\, i = 1, 2, \ldots, m\}$, L_2 Regularization Coef. λ

Procedure:

$G = \sum_{i=1}^{n} g_i$, $H = \sum_{i=1}^{n} h_i$

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

$G_L = 0$, $H_L = 0$

for j in sorted($\mathbf{D}$):

$G_L = G_L + g_j$, $G_R = G - G_L$

$H_L = H_L + h_j$, $H_R = H - H_L$

Score = max{Score, $\frac{(G_L)^2}{H_L+\lambda} + \frac{(G_R)^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}$}

Output: Attribute with max Score

但若训练数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，对此 XGBoost 还提出了一种可并行的近似 approximate algorithm 分位数草图的近似贪婪算法 Weighted Quantile Sketch，即将属性排序后按照百分位分割计算属性的分裂增益，减少计算量，用于高效地生成候选的分割点。

Optimal Splits Search: Approximately Algorithm

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i, y_i);\, i = 1, 2, \ldots, n\}$

Attribute Set $\mathbf{A} = \{\mathbf{a}_i;\, i = 1, 2, \ldots, m\}$, L_2 Regularization Coef. λ

Procedure:

$G = \sum_{i=1}^{n} g_i$, $H = \sum_{i=1}^{n} h_i$

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

Propose sorted $\mathbf{A}_i = \{a_1^i, a_2^i, \ldots, a_K^i\}$ by percentiles on $\mathbf{a}_i$ on per tree or split

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

$G_i^k = \sum_{i \in \{i \,|\, a_k^i \geq x_{ik} \geq a_{k-1}^i\}} g_i$

$H_i^k = \sum_{i \in \{i \,|\, a_k^i \geq x_{ik} \geq a_{k-1}^i\}} h_i$

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

$G_L = 0$, $H_L = 0$

for $j = 1, 2, \ldots, K(i)$:

$G_L = \sum_{k=1}^{j} G_i^k$, $G_R = G - G_L$

$H_L = \sum_{k=1}^{j} H_i^k$, $H_R = H - H_L$

Score = max{Score, $\frac{(G_L)^2}{H_L+\lambda} + \frac{(G_R)^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}$}

Output: Attribute with max Score

---

(4) 缺失值处理

对于缺失值处理，XGBoost 在计算分裂增益的过程中会将包含缺失值的数据放到左叶结点中计算信息增益，再放到右结点中计算信息增益，从中选择信息增益高的方式进行分裂。

(5) 支持 CART(gbtree) 树作为基学习器，也支持线性分类器 (gblinear)

(6) 预排序，*level_wise* 结点生长方式与单一学习器内特征并行分裂

XGBoost 算法在确定分裂方式为 Exact Split 或 Approximate Percentile Split 后，对所有属性进行预排序后以 column block 形式即 $\{(\mathbf{y}, \mathbf{a}_j)|,\, j = 1, 2, \ldots, m\}$ 形式存储于内存中。

XGBoost 算法在树生长策略上采用 *level_wise* 策略，即对同一深度的结点同时分裂，而基于此生长策略，基决策树在结点分裂时可以并行调用存储于内存中的 column block 计算属性分类，加快模型的训练过程。

(7) XGBoost 可以采用列抽样和行抽样进行学习，减小过拟合的概率。

(8) XGBoost 支持 Dart 算法，引入随机性起到减小过拟合的概率。
Dart Booster 与原有的相比有两点不同：

- 在经过 t 轮迭代学习之后，得到的 $f_t(\mathbf{x} = \sum_{i=1}^{t} \mathbb{H}(\mathbf{x}, \mathbf{y})$，Dart 首先从 t 个基学习器中随机选择 **K** 个学习器计算得到 $\hat{f}_t(\mathbf{x} = \sum_{i \in K} \mathbb{H}(\mathbf{x}, \mathbf{y})$，计算负梯度，并进行下一轮学习；

$$Obj^{(t)} = \sum_{i=1}^{t} \ell \left(y_i, f_{t-1}(\mathbf{x}) - \sum_{j \notin K} h_j(\mathbf{x} \right) + h_t(\mathbf{x})) + \Omega(\mathbb{H}_t)$$

- 由于 dropout 的步骤，需要对新生成的树进行拉伸，平衡每个学习器的权重；

DART Booster

Procedure:

Let N be the total number of trees to be added to the ensemble
$S_1 = \{x, -L_x^{'}(0)\}$
T_1 be a tree trained on the dataset S_1
M = $\{T_1\}$
for $t = 1\, to\, N$:
D = the subset of M such that $T \in M$ is in D with probability p_{drop}
if D = $\emptyset$:
D = a random element from M
$\hat{M} = M/D$
$S_t = \{x, -L_x^{'}(\hat{M}(x))\}$
T_t be a tree trained on the dataset S_t
$M = M \cup \{\frac{T_t}{|D|+1}\}$
for $T \in D$:
Multiply T in M by a factor of $\frac{|D|}{|D|+1}$

Output: M

3.6 LightGBM

LightGBM 是 GBDT 的一个拓展算法，和 XGBoost 相比在属性处理、类别属性、树生长策略、并行上有所不同：

(1) 对特征使用直方图算法，即将连续属性保存为离散的直方图

- 占用内存减少且不用保存预排序 index 等；
- 使用直方图算法在计算属性的分割是计算复杂度从 $O(\#data)$ 减小为 $O(\#bins)$；
- 直方图算法也能减少计算量，结点分裂前后可直接由直方图做差得到左右两个结点的直方图；

Histogrambased Algorithm

Input: Training Data $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$
Tree Max Depth d
$NodeSet$: tree nodes in current level, $RowSet$: Data Indices in Tree Nodes

Procedure:

for $i = 1\,to\,d$:
  for $node$ in $NodeSet$:
    usedRows = $RowSet[node]$
    for $k = 1\,to\,m$:
      H = new Histogram(); Build histogram
      for j in usedRows:
        $bin = D.f[k][j].bin$
        $H[bin].y = H[bin].y + D.y[j]$
        $H[bin].n = H[bin].n + 1$
      Find the best split on histogram H
  Update rowSet and NodeSet according to the best split points

---

(2) 树生长策略
XGBoost 采取 $level\_wise$ 策略 [# :XGBoost 现在已经支持 $leaf\_wise$] 策略，LightGBM 采取 $leaf\_wise$ ，相对来说有利于生成精度更好的学习器，但也容易过拟合，可以通过控制树的深度来反之过拟合。

(3) 支持类别特征
大多数 Boosting 方法对于类别属性采用 OneHot 编码之后进行树模型学习，会导致树模型需要生成很深且不平衡的树才能达到较高准确率，而大多数模型又会控制树模型深度，使得树的学习性能变差，而 LightGBM 通过对类别按照与目标标签的相关性进行重排序，具体一点是对于保存了类别特征的直方图根据其累计值 ($sum\_gradient\backslash sum\_hessian$) 重排序，在排序好的直方图上选取最佳切分位置。

(4) 特征并行计算最优属性
XGBoost 的特征并行：

- 对数据列采样，将不同的特征子集分配到不同的计算单元；
- 各个计算单元计算特征子集的局部最优切分点包括特征和分割点；
- 在计算单元间互相通信寻求全局最佳特征及其分割点；
- 最佳分割点的计算单元进行切分数据，并将切分结果传给其他计算单元。

LightGBM 的特征并行主要在于减少了计算单元之间的通信成本从而加速模型训练：

- 每个计算单元保留完整数据集，这样避免通信成本；
- 每个计算单元从分配的特征子集中寻找局部最优切分包括特征和切分点；
- 互相通信寻找最优切分；
- 每个计算单元执行最优切分操作。

FindBestSplit

Input: Training Data $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$
  Attribute Set $A = \{A_i;\ i = 1, 2, \ldots, m\}$
Procedure:
  for all A_i:
    Construct Histogram()
    H = new Histogram()
    for all a in A_i:

H.binAt(x.bin).Put(x.label)
Find Best Split
leftSum = new HistogramSum()
for all bin in H:
leftSum = leftSum + H.binAt(bin)
rightSum = H.AllSum - leftSum
split.gain = CalSplitGain(leftSum, rightSum)
bestSplit.gain = ChoiceBetterOne(split, bestSplit)

Output: bestSplit

PV Tree FindBestSplit

Input: Training Data $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$
Procedure:

localHistograms = ConstructHistogramS(D)
Local Voting
splits = []
for all H in localHistograms:
splits.Push(H.FindBestSplit())
localTop = splits.TopKByGain(K)
Gather all candidates
allCandidates = AllGather(localTop)
Global Voting
globalTop = all.Candidates.TopKByMajority(2*K)
Merge Global Histograms
globalHistograms = Gather(globalTop, localHistograms)
bestSplit = globalHistograms.FindBestSplit()

Output: BestSplit

---

(5) 数据并行
XGBoost 的直方图算法数据并行方式：

- 对数据行采样，分配到不同的计算单元；
- 计算单元根据分配的数据子集构建局部直方图；
- 计算单元合并局部直方图得到全局直方图；
- 对全局直方图寻求最优切分点并进行切分；

LightGBM 数据并行在于 LightGBM 不直接合并局部直方图，LightGBM 通过合并不同计算单元无交叉的不同特征的直方图，找到局部最优之后同步到全局最优切分点。

(6) GOSS Booster
LightGBM 支持 Goss 即 Gradient one side sampling 方法，在计算损失、下一轮迭代学习以及最佳属性分裂是根据学习器的学习结果计算样本的梯度大分层抽样计算，其主要思想是前若干轮迭代训练不进行 Bagging，在指定若干此迭代之后将样本分为大梯度和小梯度部分，而在计算误差及下一轮迭代学习是挑选全部大梯度样本和部分小梯度样本，并将样本的权值进行拉伸变化。

Gradient Based One Side Sampling

Input: Dataset $\mathbf{D} = \{(\mathbf{x}_i, y_i);\ i = 1, 2, \ldots, n\}$

Iterations: T, Loss function: ℓ, WeakLearner: $\mathbb{H}$
Sampling ratio of large gradient data: a
Sampling ratio of small gradient data: b

Procedure:

Initialize $models = \{\}$, $factor = \frac{1-a}{b}$, $\omega = \{1, 1, \ldots, 1\}$
$topN = a \times len(\mathbf{D})$, $randN = b \times len(\mathbf{D})$
for $t = 1, 2, \ldots T$:
$h_t(\mathbf{x}) = \mathbb{H}(\mathbf{x}, \omega_t)$
$\epsilon_t = \ell(\mathbf{x}, h_t(\mathbf{x}), \omega_t)$
sorted = GetSortedIndices(abs(ϵ_t))
topSet = sorted[1:topN]
randSet = RandomPick(sorted[topN:len($\mathbf{D}$)],randN)
usedSet = topSet + randSet
$\omega_t[randSet]$ *= factor
$newModel = \ell(\mathbf{D}[usedSet], -\epsilon_t[usedSet], \omega_t[usedSet])$
models.append(newModel)

Goss Booster 算法的优点在于假定梯度较小的样本训练误差较小，因此可以适当忽略部分小梯度的数据从而加速模型训练，较小模型误差。而基于 Goss Booster，LightGBM 的最优属性选择标准为

$$\text{SplitGain}(\mathbb{N}, \mathbf{a}) = \frac{1}{|\mathbb{N}|}[\frac{(\sum_{x_i \in LargeGradient_L} g_i + \frac{1-a}{b}\sum_{x_i \in SmallGradient_L} g_i)^2}{|\mathbb{N}|_L} + \frac{(\sum_{x_i \in LargeGradient_R} g_i + \frac{1-a}{b}\sum_{x_i \in SmallGradient_R} g_i)^2}{|\mathbb{N}|_R}]$$

3.7 CatBoost

CatBoost 算法的有点在于对类别属性的特殊处理，在训练过程中使用 Greedy TBS 将 category 转化为数值型属性。TBS 对离散属性数据连续化，处理方法为根据属性取值对应的 TARGET 取均值:

$$\hat{x}_i^k = \frac{\sum_{j=1}^n y_i I(x_j^k = x_i^k)}{\sum_{j=1}^n I(x_j^k = x_i^k)}$$

但以该方法进行处理可能会出现一个问题对于之前类别未出现的数据则在数据转化后的取值为零，所以需要进行平滑，假定先将目标变量 $\mathbf{y}$ 划分为 $\{\mathbf{D}_k\}$ 个区间 (在二分类下则不需要):

$$\hat{x}_i^k = \frac{\sum_{x_j \in D_k}^n y_j I(x_j^k = x_i^k) + aPrior}{\sum_{x_j \in D_k}^n I(x_j^k = x_i^k) + a}$$
$$\hat{x}_i^k = \frac{\sum_{x_j \in D_k}^n y_j I(x_j^k = x_i^k) + aPrior}{\sum_{j=1}^n I(x_j^k = x_i^k) + a}$$

$Prior$ 一般取值为样本目标变量均值，a 为模型训练设定的参数。

4 Loss Function

机器学习中常用损失函数包括平方损失 (L_2 Loss)、绝对损失 (L_1 Loss)、指数损失、Logloss、Hinge Loss 等

- Square Error Loss

$$\ell(y, \hat{y}) = \sum_i (y_i - \hat{y}_i)^2$$

- Aboslute Error Loss

$$\ell(y, \hat{y}) = \sum_i |y_i - \hat{y}_i|$$

- Logloss

$$\ell(y, \hat{y}) = \sum_i [y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i)]$$

- Exponential Loss

$$\ell(y, \hat{y}) = \exp(-y\hat{y})$$