WISER Club 2018, Xiamen University

# An Introduction to Tree–Based Models

Chunguang Zhang

Sept 18, 2018

# Outline

# Decision Tree

# 决策树—基本流程

**Definition**

决策树 = 决策树生成（结点分裂递归产生叶结点）+ 决策树剪枝（防止过拟合）。

- **结点如何分裂**：结点分裂最优属性选择；
- **如何生成叶结点**：生成叶结点条件；
- **剪枝**。

# 决策树——结点分裂

结点分裂的目的是为了减小模型的不确定性，不确定性衡量主要基于信息论方法和统计检验方法。

1. 信息论
   - **信息增益**：样本不充分大时，偏好取值种类多的属性；
   - **信息增益率**: 偏好取值不平衡的属性；
   - **基尼指数**：样本不充分大时，偏好取值种类多的属性；
   - TwoValuing
2. 非信息论
   - **卡方检验**：QUEST 决策树，基于信息论的属性选择有偏；统计检验大法好；

# 决策树——结点分裂：信息增益

假设一节点 N 包含 K 类样本，记 $p_k$ 为第 k 类样本的比例，假定离散属性 a 有 V 个不同取值，记 $N^v$ 为 N 中属性 a 上取值为 $a^v$ 的样本，记 $|S|$ 为 S 的样本容量，则：

- 信息熵计算：

$$Ent(N) = -\sum_{k=1}^{K} p_k log_2 p_k$$

- 信息增益计算：

$$Gain(N, a) = Ent(N) - \sum_{v=1}^{V} \frac{|N^v|}{|N|} Ent(N^v)$$

从信息增益准则公式可以看出**当样本量不充分大**时，若一个划分属性的取值数目越多，则信息增益会越小，因此信息增益偏好属性取值较多的属性进行划分。

# 决策树——结点分裂：信息增益率

为消除信息增益准则的偏好性，衍生的 C4.5 算法使用信息增益率为基础计算纯度的提升.

- 属性分离信息计算：

$$SplitInformation(N, a) = -\sum_{v=1}^{V} \frac{|N^v|}{|N|} log_2 \frac{|N^v|}{|N|}$$

- 信息增益率计算：

$$Gain\_ratio(N, a) = \frac{Gain(N, a)}{SplitInformation(N, a)}$$

其中 SplitInformation(N, a) 称为分离信息，一般随着属性 a 取值个数增多而变大。

# 决策树——结点分裂：Gini 指数

- Gini 信息：

$$Gini(N^v) = -\sum_{v=1}^{V} \frac{|N^v|}{|N|}(1 - \frac{|N^v|}{|N|})$$

- Gini 指数计算：

$$Gini\_index(N, a) = \sum_{v=1}^{V} \frac{|N^v|}{|N|} Gini(N^v)$$

Gini 指数本身计算也偏好多值属性，但一般只用在二叉树中。

# 决策树——属性分裂：连续属性

决策树本质是分类，因此对于连续型属性先进行 (伪) 离散化处理，通过二分法计算属性最优分割点的信息增益：

1. 对特征 A 进行排序；
2. 假定特征有 V 个不同特征值 $a^v$，则有 V-1 个中值 $T_A$，计算对每个中值进行分裂的信息增益；
3. 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。

$$
\begin{aligned}
Gain(N, A) &= max_{t \in T_A} Gain(N, A, t) \\
&= max_{t \in T_A} (Ent(N) - \sum \frac{|N_t^\lambda|}{|N|} Ent(N_t^\lambda)) \\
T_A &= \{\frac{a^i + a^{i+1}}{2} | 1 \leq i \leq V - 1\} \\
GainRevise(N, A) &= Gain(N, A) - \frac{log_2(V-1)}{|N|}
\end{aligned}
$$

# 决策树——属性分裂：缺失值处理

决策树在学习过程中对缺失值的处理主要从以下两方面:

- **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:

$$Gain(N, a) = \rho \times Gain(\bar{N}, a), \quad \rho = \frac{|\bar{N}|}{|N|}$$

- **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。

# 决策树——属性分裂：多变量

多变量决策树中采用属性组合进行划分，常用 OC1 算法：

- Coefficients Learning
  - CART Linear Combination:
    1. 基于单变量最优属性 $\mathbf{x}_{j.}$，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 $\sum_{i=1}^{m} a_i x_i - \delta(x_i + \gamma) + a_{m+1} <= 0$，寻求最优 $a_i$, $a_{m+1}$;
    3. Randomization; 因在步骤 (2) 中 $a_i$ 都是局部更新，在得到变量组合分割的超平面之后进行扰动优化，生成 $(r_1, r_2, \ldots, r_{m+1})$ 加入超平面 $H_1 = \sum_{i=1}^{m}(a_i + \alpha r_i)x_i + (a_{m+1} + r_{m+1})$ 再次优化;
  - Recursive Least Square Procedure

$$W_k = W_{k-1} - K_k(X_k^T W_{k-1} - y_k), \quad K_k = P_k K_k$$
$$P_k = P_{k-1} - P_{k-1} X_k [1 + X_k^T P_{k-1} X_k]^{-1} X_k^T P_{k-1}$$

- Feature Selection: 特征选取类似 stepwise。

# 决策树——叶结点生成

- 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。

# 决策树——剪枝

决策树剪枝目的是为防止过拟合：

(1) 预剪枝

在结点分裂前计算泛化指标，无法提升则停止分裂，局部最优化，优点在于模型训练成本降低，但分支未展开可能导致模型欠拟合：

- 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；

(2) 后剪枝

全局最优化，使用目标损失函数正则化来进行剪枝，训练成本高，在生成完整一决策树后从下而上，将非叶结点替换为叶结点计算模型的泛化指标，决定是否转为叶结点。

# 决策树——基本算法

---

**Input:** Train dataset $D = \{(\mathbf{x}_i, y_i) \,|\, i = 1, 2, \ldots, n;\ y_i = 1, 2, ..., k\}$;
Attributes set $A = \{A_1, A_2, \ldots, A_m\}$; Threshold $\epsilon$.
**Procedure:** TreeGenerate($D$, $A$); Current Node $N$

**if** all instance in $D$ belong to a class j:
    Label node N as Leaf Node of class j
**elif** $A = \emptyset$ or values of $\mathbf{x}_i's$ in $D$ are same:
    Label node N as Leaf Node with the most class
Select a optimal partitioning attributes $A_*$
**if** $Gain(D, A_*) < \epsilon$: **break**
**for** value $a_*^v$ in $A_*$:
    Generate branch for N; denote $D_v$ as subsample
    **if** $D_v$ is empty:
        Label node N as Leaf Node of class j
    **else** Denote TreeGenerate($D_v$, $A \backslash a_*^v$) as child node.

---

# 决策树——基本算法

不同决策树的算法的区别主要在于:

- 是否二叉树;
- 结点分裂标准;
- 连续属性处理;
- 是否剪枝方式;
- 缺失值处理。

# 决策树——算法：ID3

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。

## 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。

1. 最优属性分割指标：信息增益；
2. 连续属性处理：无法处理;
3. 缺失值处理：无法处理；
4. 剪枝：未进行后剪枝处理;
5. 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。

# 决策树——算法：C4.5

## C4.5

1. 最优属性分割指标：**信息增益结合信息增益率**；
2. 属性处理：二分法处理计算信息增益；
3. 缺失值处理：根据后验概率分配到子结点；
4. 剪枝：采用悲观剪枝法 (Pessimistic Error Pruning)；
   - 内部结点 t 的样本个数为 $N(t)$，最优子树为 $N_t$；
   - 作为叶结点的错误率 $error_t = n(t) + \frac{1}{2}$；
   - 作为内部结点的错误率 $error_{T_t} = n(T_t) + \frac{N_t}{2}$ 以及标准差 $SE = \sqrt{\frac{n(T_t)(N(t)-n(T_t))}{N(t)}}$
   - 若 $error_{T_t} + SE - error_t > 0$，则不分裂；
5. 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。

# 决策树——算法：CART

## CART

1. 最优属性分割：一般为 GINI 指数；
2. 属性处理：二分法处理连续和离散属性，重复性使用；
3. 缺失值处理：根据后验概率分配到子结点；
4. 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；
5. 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。

# 决策树——算法：CART 剪枝

## CART 剪枝

- 计算内部结点 t 为单结点树 (t) 和根节点 ($T_t$) 的损失

$$C_\alpha(t) = C(t) + \alpha, \quad C_\alpha(T_t) = C(T_t) + \alpha|T_t|$$

- 当且仅当 $C_\alpha(t) < C_\alpha(T_t)$，进行剪枝剪除 $T_t$;
- 令 $g(t) = \frac{C(t)-C(T_t)}{|T_t|-1}$，计算所有内部结点的 g(t)，依次增加 $\alpha$，自下而上在 $T_i$ 中剪除 g(t) 最小的 $T_t$ 得到 $T_{i+1}$ 子树;
- 利用独立测试集计算独立子树的损失函数，取最优子树。

$C_\alpha(T) = \sum_{t=1}^{|T|} N_t H_t + \alpha|T| = \sum_{t=1}^{|T|} N_t(-\sum_k \frac{N_{tk}}{N_t} log \frac{N_{tk}}{N_t}) + \alpha|T|$
$\alpha$ 为树复杂度惩罚系数，$|T|$ 为结点数目，$N_t$ 为结点 t 的样本数，$N_{tk}$ 为结点 t 内类 k 的样本数，$H_t$ 为结点 t 的经验熵。



## CART 回归树

**CART 回归树**主要先将输入空间划分为 M 个空间 $R_1, R_2, \ldots, R_M$，每个空间对应输出值 $c_1, c_2, \ldots, c_M$，对属性 A 选择和切分点 v 时不是选择信息增益或基尼指数，而是最小化:

$$min_{A,s}[min_{c_1} \sum_{x_i \in \{x|x^A \leq s\}} (y_i - c_1)^2 + min_{c_2} \sum_{x_i \in \{x|x^A \geq s\}} (y_i - c_2)^2]$$

$c_i$ 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。

# 决策树算法：QUEST

- QUEST 会以不同的策略处理特征变量和切分点的选择问题，其算法运行过程要比 CART 更加快速有效。
- QUEST 算法在选择特征变量时，对于定类属性采用卡方检验方法，这点和 CHAID 类似，对于定矩属性则采用 F 检验方法，选择对应 p 值最小且小于显著性水平的特征变量作为最佳分类变量。

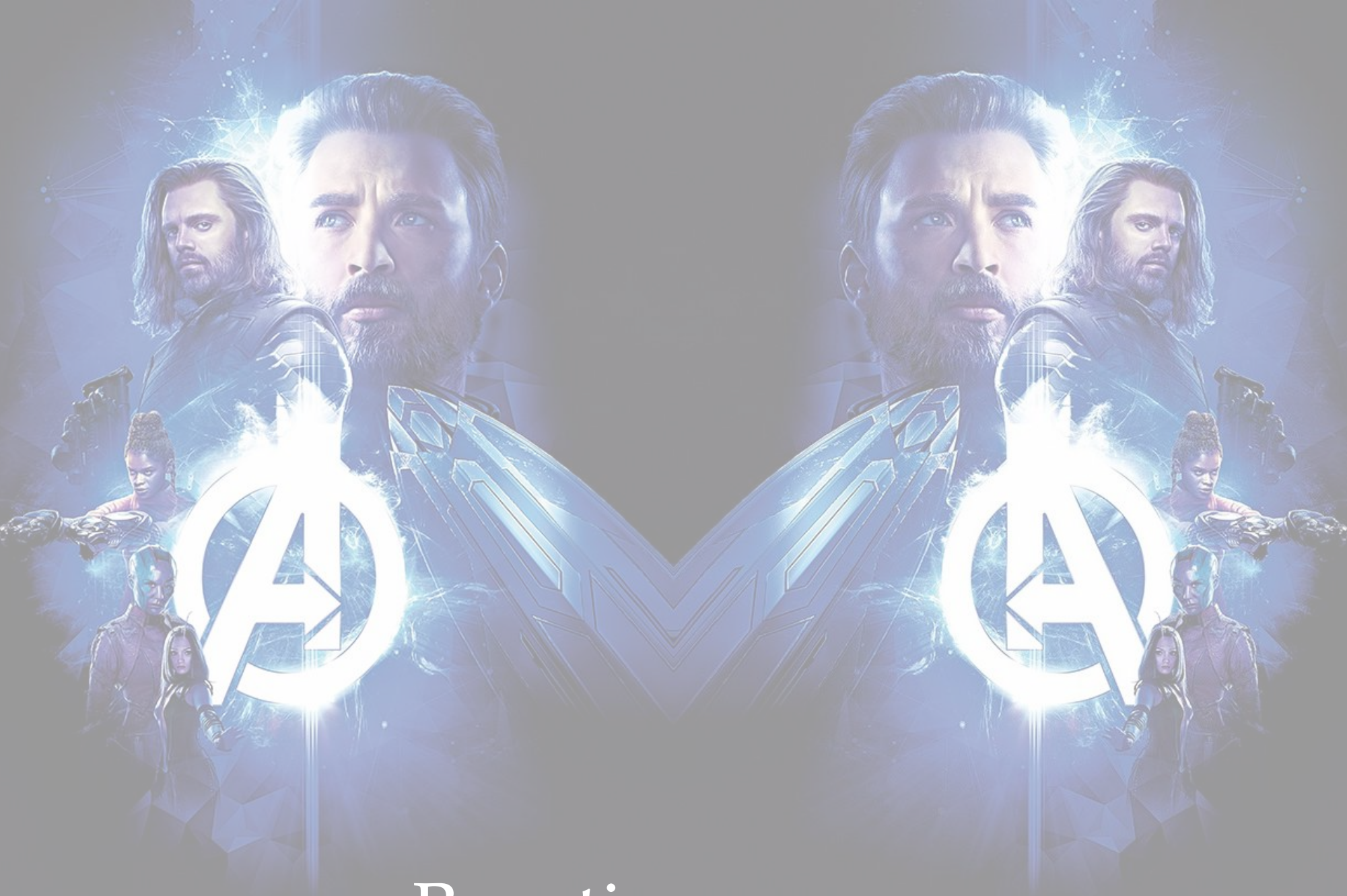
# 决策树——Sklearn Decision Tree

Scikit–learn 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,
        max_features=None, random_state=None,
        max_leaf_nodes=None, min_impurity_decrease=0.0,
        min_impurity_split=None,class_weight=None,
        presort=False)
```

# Bagging

# Bootstrap& 随机属性选择



# Boosting

# Boosting——基本流程

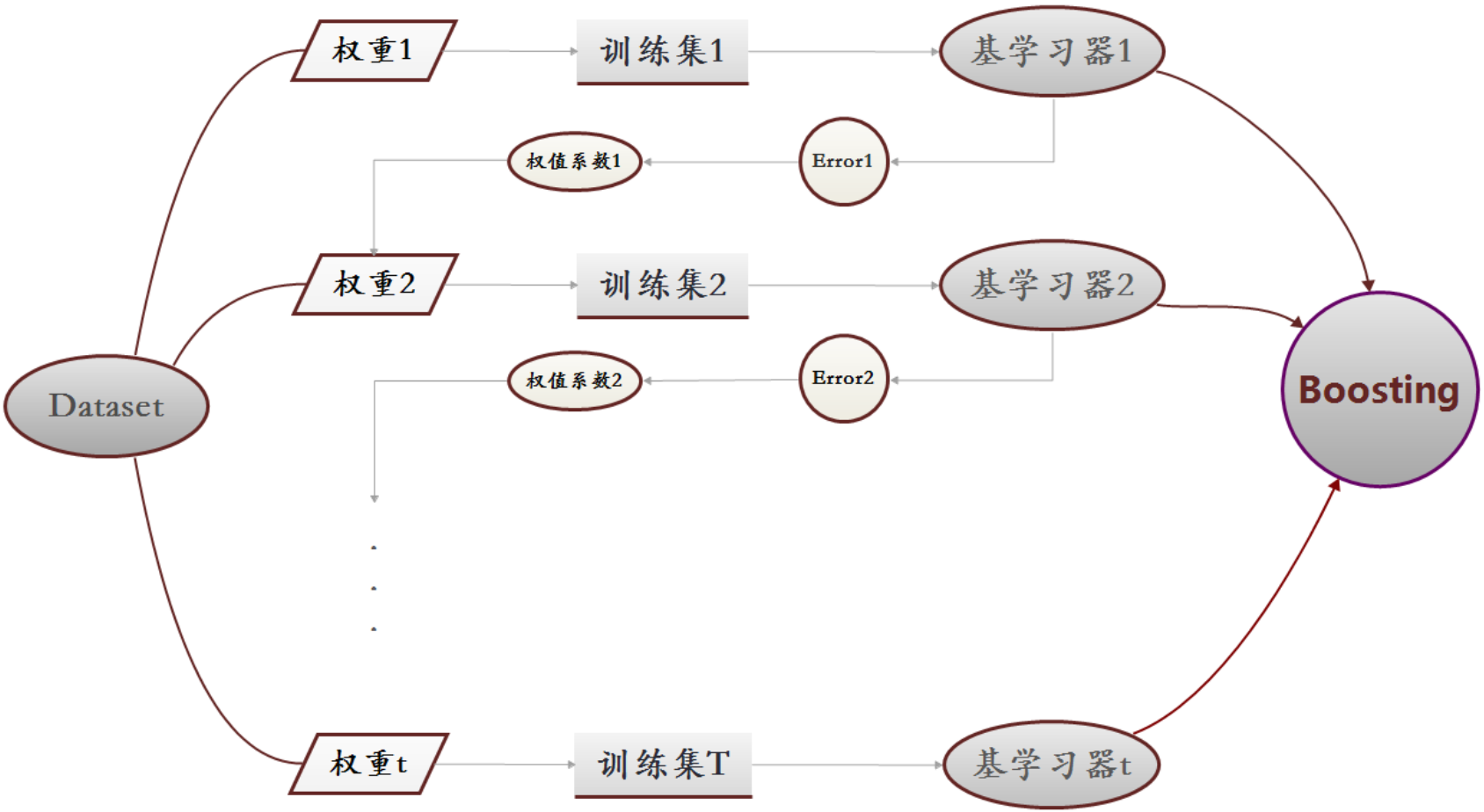



Figure: Boosting

# Boosting——基本流程

- 从初始训练集用初始权重训练出一个基学习器 $h_1$；
- 根据基学习器的学习误差率 $\epsilon$ 表现更新训练样本权值，使得误差率高的训练样本权值 $W_i$ 增加;
- 基于新的样本权值训练基学习器 $h_2$;
- 迭代进行，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 $\alpha_i$ 组合基学习器的训练结果输出。

# Boosting——基本流程

所以不同的 Boosting 方法的区别在于以下几点:

- 基学习器误差率的计算即 Loss function 形式;
- 如何根据误差率更新样本权重系数 W 或计算梯度;
- 如何确定基学习器的权重系数。

针对大部分 Boosting 算法，基本都是采用加法模型来组合弱分类器，用前向分布算法来改变训练数据的权重和概率分布;

# Boosting——加法模型

- 考虑 K 棵树的加法模型：$\hat{y} = \sum_{k=1}^{K} \beta_k f(x)$
- 解决这一优化问题采用前向分布算法，每次只学习一个基函数及系数，逐步逼近优化目标函数最小化，即

$$Obj^t = \sum_{i=1}^{n} l(y_i, \hat{y}_i^t) + \sum_{i=1}^{t} \Omega(f_i) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t) + Const.$$

- 优化问题转为基于前一学习器预测来最优化当前损失函数，即最小化 $\sum_{i=1}^{n} l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t)$；
- 以平方损失函数为例，则模型转为拟合上一轮的残差

$$\sum_{i=1}^{n} (y_i - \hat{y}_i^{t-1} + f_t(x_i))^2 + \Omega(f_t) = \sum_{i=1}^{n} (\epsilon_i - f_i(x))^2 + \Omega(f_t)$$

- 损失函数形式及其泰勒展开的不同衍生不同的 Boosting。

# AdaBoost

AdaBoost 采用损失函数为最小化指数损失函数 $\ell_{exp}(H|W)$ 的加权加法的**前向分布算法**模型输出 H(x)，假定 f 为真实分布，即

$$H(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$$

$$\ell_{exp}(H|W) = E_{\mathbf{x}|W}[exp(-f(x)H(x))]$$

AdaBoost 作为弱分类器的组合器，基分类器越弱，效果提升越明显，结果也越好，且不容易过拟合，但由于 AdaBoost 的权值更新方式的原理，AdaBoost 对异常数据比较敏感，因异常数据在权值更新中往往会获得较高的权值。

# AdaBoost for Binary

**Algorithm: AdaBoost for Binary Classification**

**Input:** Train dataset $D = \{(\mathbf{x}_i, y_i) \,|\, i = 1, 2, \ldots, n;\ y_i = -1, 1\}$
WeakLearn Algorithm I
Number of Weak Learn Algorithm T

**Procedure:** Initializing sample weight vector $W_1(\mathbf{x}) = \frac{1}{n}$
**for** $t = 1, 2, \ldots, T$:
$h_t = I(D, W_t)$
$\epsilon_t = P_{\mathbf{x}|W_t}(h_t(\mathbf{x}) \neq f(\mathbf{x}))$
**if** $\epsilon_t > 0.5$: **break**
$\alpha_t = \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t}$
$W_{t+1} = \frac{W_t(\mathbf{x})}{Z_t} exp[-\alpha f(\mathbf{x}) h_t(\mathbf{x})]$

**Output:** $H(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$

# AdaBoost for Binary

样本权值更新方式: $W_t$

$$
\begin{aligned}
W_{t+1} &= \frac{W_t(\mathbf{x})}{Z_t} \times \begin{cases} exp(-\alpha_t), & if\ h_t(\mathbf{x}) = f(\mathbf{x}) \\ exp(\alpha_t), & if\ h_t(\mathbf{x}) \neq f(\mathbf{x}) \end{cases} \\
&= \frac{W_t(\mathbf{x})}{Z_t} exp(-\alpha f(\mathbf{x}) h_t(\mathbf{x}))
\end{aligned}
$$

# AdaBoost for Binary

## 损失函数解释

$$\frac{\partial \ell_{exp}(H|W)}{\partial H(\mathbf{x})} = -exp(H(\mathbf{x}))P(f(\mathbf{x}) = 1|\mathbf{x}) + exp(H(\mathbf{x}))P(f(\mathbf{x}) = -1|\mathbf{x})$$

$$H(\mathbf{x}) = \frac{1}{2} ln \frac{P(f(\mathbf{x}) = 1|\mathbf{x})}{P(f(\mathbf{x}) = -1|\mathbf{x})}$$

$$sign(H(\mathbf{x})) = sign[\frac{1}{2} ln \frac{P(f(\mathbf{x}) = 1|\mathbf{x})}{P(f(\mathbf{x}) = -1|\mathbf{x})}]$$

$$= \begin{cases} 1, & \mathrm{P(f(x)=1 \mid x) > P(f(x)=-1 \mid x);} \\ -1, & \mathrm{P(f(x)=1 \mid x) < P(f(x)=-1 \mid x).} \end{cases}$$

$$= argmax_{y \in \{-1,1\}} P(f(\mathbf{x}) = y)$$

# AdaBoost for Binary

## 权值更新方式：$\alpha_t$

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

# AdaBoost for Binary

$$
\begin{aligned}
h_t(\mathbf{x}) &= argmin_h \ell_{exp}(H_{t-1} + \alpha_t h(\mathbf{x})|W) \\
&= argmin_h E_{\mathbf{x}|W}[exp(-f(\mathbf{x}))(H_{t-1} + \alpha_t h(\mathbf{x}))] \\
&= argmin_h E_{\mathbf{x}|W}[exp(-f(\mathbf{x})H_{t-1})exp(-f(\mathbf{x})h(\mathbf{x}))] \\
&\equiv argmin_h E_{\mathbf{x}|W}[exp(-f(\mathbf{x})H_{t-1})(1 - f(\mathbf{x})h(\mathbf{x}) + \frac{f^2(\mathbf{x})h^2(\mathbf{x})}{2})] \\
&= argmax_h E_{\mathbf{x}|W}[exp(-f(x)H_{t-1})f(x)h(x)] \\
&= argmax_h E_{\mathbf{x}|W_t}[f(\mathbf{x})h(\mathbf{x})], \, where \, W_t(\mathbf{x}) = \frac{exp(-f(\mathbf{x})H_{t-1})}{E_{\mathbf{x}|W}[exp(-f(\mathbf{x})H_{t-1}(\mathbf{x}))]} \\
&= argmax_h E_{\mathbf{x}|W_t}[1 - 2I(f(\mathbf{x}) \neq h(\mathbf{x}))] = argmin_h E_{\mathbf{x}|W_t}[I(f(\mathbf{x}) \neq h(\mathbf{x}))]
\end{aligned}
$$

$$
W_{t+1}(\mathbf{x}) = \frac{W_t exp(-f(\mathbf{x})H_t(\mathbf{x}))}{E_{\mathbf{x}|W}[exp(-f(\mathbf{x})H_t(\mathbf{x}))]} = W_t exp(-f(\mathbf{x})\alpha_t h_t) \frac{E_{\mathbf{x}|W}[exp(-f(\mathbf{x})H_{t-1}(\mathbf{x})]}{E_{\mathbf{x}|W}[exp(-f(\mathbf{x})H_t(\mathbf{x}))]}
$$

# AdaBoost for Multiclass Classification

AdaBoost 是基于二分类的分类方法，而在多分类中一般有以下几种思路

- 只判断是否分类正确与否改变权重；
- 判断分类错误为哪一类别，并调整权重；

基于以上的思路就产生了 AdaBoost.m1 和 AdaBoost.m2 算法。

# AdaBoost.m1 for Multiclass Classification

AdaBoost.m1 是基于 AdaBoost 的多分类方法，在更新权值时，分类错误样本权值不更新，而分类正确的样本权重乘以 $\frac{\epsilon_t}{1-\epsilon_t} < 1$。

---

**Input:** Train dataset $D = \{(\mathbf{x}_i, y_i) \,|\, i = 1, \ldots, n;\ y_i = 1, \ldots, k\}$
WeakLearn Algorithm I
Number of Weak Learn Algorithm T

**Procedure:** Initializing sample weight vector $W_1(\mathbf{x}) = \frac{1}{n}$
**for** $t = 1, 2, \ldots, T$ **do**
$h_t = I(D, W_t)$
$\epsilon_t = P_{\mathbf{x}|W_t}(h_t(\mathbf{x}) \neq f(\mathbf{x}))$
**if** $\epsilon_t > 0.5$: **break**
$\alpha_t = ln\frac{1-\epsilon_t}{\epsilon_t}$
$W_{t+1} = \frac{W_t}{Z_t} exp(-\alpha_t(1 - [h_t(\mathbf{x}) \neq y]))$

**Output:** $H(\mathbf{x}) = argmax_{y \in C} \sum_{t=1}^{T} \alpha_t [h_t(\mathbf{x}) = y]$

---

# AdaBoost.m2 for Multiclass Classification

**Input:** Train dataset $D = \{(\mathbf{x}_i, y_i) \,|\, i = 1, \ldots, n;\ y_i = 1, \ldots, k\}$;
WeakLearn I; Nums of Iteration T.

**Procedure:** Initializing sample weight vector $W_1(i) = \frac{1}{n}$;

$\omega_{i,y}^1 = \frac{W_i}{k-1}$ for $i = 1, 2, \ldots, n; \quad y \in \{1, 2, \ldots, k\}/y_i$;

**for** $t = 1, 2, \ldots, T$ **do**

$\Omega_i^t = \sum_{y \neq y_i} \omega_{i,y}^t; \quad q_t(i, y) = \frac{\omega_{i,y}^t}{\Omega_i^t}$ *for* $y \neq y_i$;

$W_t^i = \frac{\Omega_i^t}{\sum_{i=1}^N \Omega_i^t}; \quad h_t = I(D, W_t)$;

$\epsilon_t = \frac{1}{2} \sum_{i=1}^N W_t(i)(1 - h_i(x_i, y_i) + \sum_{i, y \neq y_i} q_t(i, y) h_t(x_i, y))$;

**if** $\epsilon_t > 0.5$: **break**

$\alpha_t = ln \frac{1-\epsilon_t}{\epsilon_t}$;

$\omega_{i,y \neq y_i}^{t+1} = \omega_{i,y \neq y_i}^t exp(-\frac{1}{2}\alpha_t(1 + h_t(x_i, y_i) - h_t(x_i, y)))$;

**Output:** $H(\mathbf{x}) = argmax_{y \in C} \sum_{t=1}^T \alpha h_t(x, y)$;

# AdaBoost.m2 for Multiclass Classification

和 AdaBoost.m1 原理比较，AdaBoost 只关注分类是否错误，而不关注错分在什么样本上，AdaBoost.m2 不仅关注样本分类是否错误，而且关注样本分类错误是在错分在哪一类别上，因此在更新权值时会更新错误分类的权值（如下所示在更新 $\epsilon$ 时更新 $\beta$ 进而提高错误分类样本的权重，同时在最后一步更新 $\omega$ 是降低修改其他类的权重）。

# AdaBoost.R for Regression

**Input:** Train dataset $D = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, \ldots, n;\ y_i \in [0, 1]\}$;
WeakLearn I; Nums of Iteration T.

**Procedure:** Initializing sample weight vector $W_1(i) = \frac{1}{n}$;

$\omega_{i,y}^1 = \frac{W_i * |y - y_i|}{Z}$, where $Z = \sum_{i=1}^{n} W(i) \int_0^1 |y - y_i| dy$;

**for** $t = 1, 2, \ldots, T$ **do**

$W^t = \frac{\omega^t}{\sum_{i=1}^{n} \int_0^1 \omega_{i,y}^t} dy$;

$h_t = I(D, W_t)$;

$\epsilon_t = \sum_1^n | \int_{y_i}^{h_t(x_i)} W_{i,y}^t | dy$;

**if** $\epsilon_t > 0.5$: **break**

$\alpha_t = ln \frac{1-\epsilon_t}{\epsilon_t}$;

$\omega_{i,y}^{t+1} = \omega_{i,y}^t exp(-\alpha[1 - I(y \in [y_i, h_t(x_i)] or [h_t(x_i), y_i])])$;

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

# AdaBoost.R2 for Regression

**Input:** Train dataset $D = \{(\mathbf{x}_i, y_i) \,|\, i = 1, \ldots, n;\ y_i \in [0, 1]\}$;
WeakLearn I; Nums of Iteration T.

**Procedure:** Initializing sample weight vector $W_1(i) = \frac{1}{n}$;

$\omega_{i,y}^1 = \frac{W_i|y-y_i|}{Z}$, where $Z = \sum_{i=1}^{n} W(i) \int_0^1 |y - y_i| dy$;

**for** $t = 1, 2, \ldots, T$ **do**

$h_t = I(D, W_t)$; $Z^t = max_{i \in \{1,2,\ldots,n\}} |y_i - h_t(\mathbf{x}_j)|$;

$e_i^t = \frac{|y_i - h_t(\mathbf{x}_j)|}{Z_t}$ for linear 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:** $H(\mathbf{x}) = median(\{\alpha_t h_t(x),\ i = 1, 2, ...\})$;

# AdaBoost.RT for Regression

**Input:** Train dataset $D = \{(\mathbf{x}_i, y_i) \,|\, i = 1, 2, \ldots, n;\ y_i \in [0, 1]\}$;
WeakLearn I; Nums of Iteration T.
threshold $\phi$.

**Procedure:** Initializing sample weight vector $W_1(i) = \frac{1}{n}$;
**for** $t = 1, 2, \ldots, T$ **do**

$h_t = I(D, W_t)$;

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

$\alpha_t = -2ln\epsilon_t$;

$W_{t+1}(i) = \frac{W_t(i)}{Z_t} exp(-\alpha_t I[|\frac{y_i - h_t(\mathbf{x}_j)}{y_i}| > \phi])$;

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

# AdaBoost——Sklearn

## Code

```
AdaBoostClassifier(base_estimator, n_estimators,
                   learning_rate, algorithm)
```

## Algorithm

SAMME&SAMME.R

# Boosting——Boosting Decision Tree

采用前向分布算法，确定初始提升树之后对第 m 步的建模为:

$$f_m(x) = f_{m-1}(x) + T(x;\Theta_m),\ s.t.$$

$$\hat{\Theta}_m = argmin_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m))$$

---

- 训练集 $T = (x_i, y_i)$; 初始化 $f_0(x) = 0$
- for m in 1, 2, ..., M
  - 计算残差 $r_{mi} = y_i - f_{m-1}(x_i)$
  - 训练一个回归树 $T(x;\Theta_m)$ 拟合残差 $r_{mi}$
  - 更新 $f_m(x) = f_{m-1}(x) + T(x;\Theta_m)$
- 得到回归树 $f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m)$

---

# Boosting——GBDT

GBDT 在基于不同的损失函数上，利用梯度下降法展开目标函数进行优化，进行一阶泰勒展开$f(x+\Delta x)=f(x)+f^{'}(x)\Delta x$，则有

$$
\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} \ell_{(}y_i, \hat{y}_i^{t-1} + f_t(x_i)) \\
&= \sum_{i=1}^{n} [\ell(y_i, \hat{y}_i^{t-1}) + \frac{\partial \ell(y_i, \hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}} f_t(x_i)] \\
&= \sum_{i=1}^{n} [Obj_i^{t-1} + \frac{\partial \ell(y_i, \hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}} f_t(x_i)]
\end{aligned}
$$

梯度上升方向为$\frac{\partial l(y_i,\hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}}$，若 f 取负梯度有$Obj^{(t)} < Obj^{(t-1)}$, 即$f_t = -\epsilon \frac{\partial l(y_i,\hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}}$, $\epsilon > 0$,$\epsilon$ 一般就是学习率;

# Boosting Decision Tree

- 训练集 $T = (x_i, y_i)$
- 初始化 $f_0(x) = 0$
- for m in 1, 2, ..., M
  - 计算负梯度 $r_{mi} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f=f_{m-1}}$
  - 训练一个回归树 $T(x; \Theta_m)$ 拟合梯度 $r_{mi}$, 得到 $T(x; \Theta_m)$ 的叶结点区域 $R_{mj}$
  - 对 $j = 1, 2, \ldots, J$, 计算出区域加权输出值使得 $c_{mj} = argmin_c \sum_{x_i \in R_{mj}} L(f_{m-1}(x_i), c)$;
  - $\alpha_m = argmin_\alpha \sum_i^n L(y_i, f_{m-1}(x_i) + \alpha h_t(x_i))$;
  - 更新 $f_m(x) = f_{m-1}(x) + \alpha_m h_m(x)$
- Return $f_M(x) = \sum_{m=1}^M \alpha_m h_m(x)$

# Boosting Decision Tree

- GBDT 的基学习器分裂原则为选择方差增益最高的属性, 即:

$$V_{A_k|D}(d) = \frac{1}{n}\left(\frac{G_L(A_k)^2}{n_L} + \frac{G_R^2(A_k)}{n_R}\right)$$

- GBDT 和 Boosting Decision Tree 的区别在于损失函数取平方误时 $c_i$ 是否直接在基学习器的树中进行优化（平方误的导函数为线性）；

# XGBoost——正则化

XGBoost 是 GBDT 的一个拓展，主要对损失函数展开、树模型正则化以及伪并行化进行改进。

- 正则化

$$
\begin{aligned}
Obj &= \sum_{i}^{n} \ell(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{t=1}^{T} \Omega(f_t) \\
&= \sum_{i}^{n} \ell(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{t=1}^{T} (\gamma |T_t| + \frac{1}{2}\lambda \|\omega\|^2) \\
&= \sum_{i}^{n} \ell(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{t=1}^{T} (\gamma |T_t| + \frac{1}{2}\lambda \sum_{m=1}^{M} \omega_m^2)
\end{aligned}
$$

$|T_t|$ 为树 $T_t$ 的叶结点个数，$\omega_m(x) = f_t(x)$ 为 m 叶结点的取值

# XGBoost——损失函数展开

- 对损失函数利用牛顿法对目标函数进行二阶泰勒展开优化；

$$
\begin{aligned}
Obj^{(t)} &= \sum_i \ell(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_k) + Constant \\
&= \sum_i [\ell(y_i, \hat{y}_i^{(t-1)}) + \frac{\partial \ell(y_i, \hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}} f_t(x_i) + \frac{1}{2} \frac{\partial^2 l(y_i, \hat{y}_i^{t-1})}{\partial (\hat{y}_i^{t-1})^2} f_t^2(x_i)] + \Omega( \\
&\equiv \sum_i [\ell(y_i, \hat{y}_i^{(t-1)}) + g_i \omega_{q(x_i)} + \frac{1}{2} h_i \omega_{q(x_i)^2}] + \gamma |T| + \frac{1}{2} \sum_{t=1}^{T} \omega_t^2 \\
&= \sum_{t=1}^{T} [(\sum_{i \in R_t} g_i) \omega_t + \frac{1}{2} (\sum_{i \in R_t} + \lambda) \omega_t^2] + \gamma |T| \\
&\equiv \sum_{t=1}^{T} [G_j \omega_t + \frac{1}{2} (H_t + \lambda) \omega_t^2] + \gamma |T|
\end{aligned}
$$

# XGBoost——属性分裂

在对梯度拟合求解最优得到

$$\omega_t^* = -\frac{G_j}{H_j+\lambda} \quad \Rightarrow \quad Obj = -\frac{1}{2}\sum_T \frac{G_j^2}{H_j+\lambda} + \gamma|T|;$$

- GBDT 以 CART 回归树作为基学习器时对属性的最佳分割点的选择衡量标准为最小化均方差，而 XGBoost 在加入正则化项之后寻找分割点的标准是最大化分裂收益:

$$Obj\_before\_split = -\frac{1}{2}\frac{(G_L+G_R)^2}{H_L+H_R+\lambda} + \gamma \times 1$$

$$Obj\_after\_split = -\frac{1}{2}[\frac{(G_L)^2}{H_L+\lambda} + \frac{(G_R)^2}{H_R+\lambda}] + \gamma \times 2$$

$$node\_split\_gain = \frac{1}{2}[\frac{(G_L)^2}{H_L+\lambda} + \frac{(G_R)^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}] - \gamma$$

# XGBoost——贪心分裂

Optimal Splits Search: Greedy Algorithm

**Input:** Dataset $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$;
Attribute Set $A = \{A_i;\ i = 1, 2, \ldots, m\}$; Initialize $\lambda$;
**Procedure:**
$Gain = 0, G = \sum_{i=1}^{n} g_i,\ H = \sum_{i=1}^{n} h_i$;
**for** $i = 1, 2, \ldots, m$ **do**
$G_L = 0,\ H_L = 0$;
**for** j in sorted(D) **do**
$G_L = G_L + g_j,\ G_R = G - G_L$;
$H_L = H_L + h_j,\ H_R = H - H_L$;
$score = max\{score, \frac{(G_L)^2}{H_L+\lambda} + \frac{(G_R)^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\}$;
**Output:** Split with max Score

# XGBoost——分位数估计

Optimal Splits Search: Approximately Algorithm

**Input:** Dataset $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$;
Attribute Set $A = \{A_i;\ i = 1, 2, \ldots, m\}$; Initialize $\lambda$;
**Procedure:**
**for** $i = 1, 2, \ldots, m$ **do**
Propose $S_k = \{s_{k1}, s_{k2}, \ldots, s_{kl}\}$ by percentiles on feature $A_k$ on per tree or per split;
**for** $i = 1, 2, \ldots, m$ **do**
$G_{kv} = \sum_{i \in \{i \,|\, s_{k,v} \geq x_{ik} \geq s_{k,v-1}\}} g_i$
$H_{kv} = \sum_{i \in \{i \,|\, s_{k,v} \geq x_{ik} \geq s_{k,v-1}\}} h_i$
Follow the same step in Greedy Algorithm.
**Output:** Split with max Score

# XGBoost——其他特征

- 支持 CART(gbtree) 树作为基学习器，也支持线性分类器 (gblinear);
- PreSorted+Exact Split/Approximate Percentile Split 对所有数据进行预排序后以 columb block 形式存于内存中，每个变量和标签作为一个 Block，之后每次计算分裂时只需要调用即可;

# LightGBM

LightGBM 也是 GBDT 的变种，在算法原理上并未进行改进但有其他特征；

(1) 在数据输入模型时，对连续属性进行离散化成 k 个整数，同时构造宽度为 k 的直方图存于内存中 [bin mapper]，后续遍历数据计算分裂时直接使用直方图的累计统计量即可，因此模型时间运行大幅降低；

(2) 与 XGBoost 采用 Level Wise 树生长后进行剪枝的策略不同，LightGBM 使用 Leaf Wise 生成策略；

# LightGBM

LightGBM 也是 GBDT 的变种，在算法原理上并未进行改进但有其他特征；

- Goss：基于梯度的抽样 Boosting;
- Exclusive Feature Bundleing：属性集合选择；
- Histogram Based Algorithm：直方图算法；
- PV Tree FindBestSplit 最优属性划分；
- DART 算法；

# LightGBM——GOSS Sampling

**Input:** Dataset $D = \{(\mathbf{x}_i, y_i);\ i = 1, \ldots, n\}$; Iterations: T;
Sampling ratio of large and small gradient: a, b;
Loss function: $\ell$; Weak Learn: I;
**Procedure:** Initialize $models = \{\}$, $fact = \frac{1-a}{b}$, $W = \{1, \ldots, 1\}$;
$topN = a \times len(D)$, $randN = b \times len(D)$
**for** $t = 1$ *to* $T$ **do**
$h_t(\mathbf{x}) = I(\mathbf{x}|W_t)$; $\epsilon_t = \ell(y, h_t(\mathbf{x})|W_t)$;
sorted = SortedIndice(abs($\epsilon_t$));
randSet = RandPick(sorted[topN:len(D)],randN)
topSet = sorted[1:topN]; usedSet = topSet + randSet;
$W_t[randSet]$ *= $fact$;
$newModel = \ell(D[usedSet], -\epsilon_t[usedSet], W_t[usedSet])$;
models.append(newModel)

# LightGBM——EFB: Greedy Bundling

**Input:** $F$: features; $K$: max conflict count; $G$: Construct graph
**Procedure:**

*SearchOrder* = G.sortByDegree();
*bundles* $= \{\}$, *bundlesConflict* $= \{\}$;
**for** $i$ in *searchOrder* **do**
*needNew* = True;
**for** $j$ in $len(bundles)$ **do**
$cnt$ = ConflictCnt($bundles[j]$,$F[i]$);
**if** $cnt + bundlesConflict[i] \leq$ K **then**
$bundles[j]$.add($F[i]$);
*needNew* = False;
**if** *needNew* **then**
Add $F[i]$ as a new bundle to *bundles*;

**Output:** *bundles*

# LightGBM——EFB: Merge Exclusive

**Input:** *numData*: number of data;
*F*: One bundle of exclusive features;
**Procedure:**
Initialize $binRanges = \{0\}, totalBin = 0$;
**for** $f$ in $F$ **do**
$totalBin += f.numBin$;
$binRanges$.append($totalBin$);
newBin = new Bin($numData$);
**for** $i = 1$ to $F$ **do**
newBin[i]=0
**for** $j = 1$ to $len(F)$:
**if** $F[j].bin[i] \neq 0$:
$newBin[i] = F[j].bin[i] + binRanges[j]$;
**Output:** $newBins, binRanges$

# LightGBM——Histogram Based Algorithm

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

**Procedure:**

```
for i = 1 to d do
    for node in NodeSet do
        usedRows = RowSet[node];
        for k = 1 to m do
            H = new Histogram(); Build histogram;
            for j in usedRows do
                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 with best split points.
    ...
```

# LightGBM——PV Tree Find Best Split

LightGBM 基学习器的最佳分裂规则是

$$V_{A_k|D}(d) = \frac{1}{n}[\frac{(\sum_{x_i \in LargeG_L} g_i + \frac{1-a}{b} \sum_{x_i \in SmallG_L} g_i)^2}{n_L} + \frac{(\sum_{x_i \in LargeG_R} g_i + \frac{1-a}{b} \sum_{x_i \in SmallG_R} g_i)^2}{n_R}]$$

传统的分布式计算最佳分裂点是基于 Local Worker 平行计算之后汇总 Global 后找出 Global BSP 再去 Local Worker 进行分裂；而 LightGBM 以 PV Tree FindBestSplit 策略加速寻找最优属性分裂。

# LightGBM——PV: Find Best Split

**Input:** Data $D = \{(x_i, y_i);\ i = 1, \ldots, n\}$; Attributes $A = \{A_i;\ i = 1, \ldots, m\}$;

**Procedure:**

```
for all A_i do
    Construct Histogram(); H = new Histogram();
    for all a in A_i do
        H.binAt(x.bin).Put(x.label);
    leftSum = new HistogramSum();
    for all bin in H do
        leftSum = leftSum + H.binAt(bin)
        rightSum = H.AllSum - leftSum
        split.gain = CalSplitGain(leftSum, rightSum)
        bestSplit.gain = ChoiceBetterOne(split, bestSplit)
```

**Output:** bestSplit

# LightGBM——PV Tree

**Procedure:**

localHist = ConstructHistogramS(D);
**Local Voting**
splits = []
**for** all H in localHist **do**
    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, localHist);
bestSplit = globalHistograms.FindBestSplit()

# LightGBM–DART

**Procedure:**

Let N be the total number of trees
$S_1 = \{x, -L_x^{'}(0)\}$
M = $\{T_i\}$; $T_1$ be a tree trained on the dataset $S_1$
**for** $t = 1\, to\, N$ **do**
  D = the subset of M s.t. $T \in M$ is in D with prob $p_{drop}$
  **if** D = $\emptyset$ **then**;
    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** $T \in D$ **do**
    Multiply T in M by a factor of $\frac{|D|}{|D|+1}$;

**Output:** M

# CatBoost– Algorithm

**Input:** Training Data $D = \{(x_i, y_i);\ i = 1, \ldots, n\}$; Iterations T;
Number of Permutations S; Train Mode $M$; $\ell$,$\alpha$;

$\sigma_i$ = random permutation of $\{1, \ldots, n\}$ for $i = 1, \ldots, s$;

$S_r(i) = 0,\ r = 1, \ldots, s$ for $i = 1, \ldots, n$;

$S'_{r,j}(i) = 0,\ r = 1, \ldots, s$ for $i = 1, \ldots, n, j = 1, \ldots, [log_2 n]$;

**for** i = 1 to T **do**

$grad = ClacGrad(L, S, y)$; $grad' = ClacGrad(L, S', y)$;

$r = random(1, s)$; $T_t = BuildTree(M, grad_r, grad'_r, \sigma_r, \mathbf{x})$;

$leaf_{r,i} = GetLeaf(\mathbf{x}_i, T_t, \sigma_r)$ for $r = 0, \ldots, s;\ i = 1, \ldots, n$;

**foreach** *leaf* $R_j^t$ in $T_t$ **do**

$b_j^t = -avg(grad_0(i)\ for\ i : leaf_{r,i} = j)$;

$S, S' = UpdModel(M, leaf, T_t, \{b_j^t\}_j, S, S', grad, grad', \sigma_{r\,r=1}^{s})$;

**Output:** $F(x) = \sum_{t=1}^{T} \sum_j \alpha b_j^t I(GetLeaf(x, T_t, ApplyMode) == j)$

# CatBoost– BuildTree

**Input:** $M, grad_r, grad'_r, \sigma_r, \mathbf{x}$; T = empty tree;
  **foreach** step of topdown procedure **do**;
    Form a set $C$ of candidate splits;
    **for each** $c \in C$ **do**
      $T_c$ = add split to T; $leaf_i$ = GetLeaf($\mathbf{x}_i, T_c, \sigma$);
      **for each** *leaf* $j$ *in* $T_c$ **do**
        **for each** $i : \ leaf_i = j$ **do**
          **if** M == Plain **then**
            $\Delta(i) = avg(grad(p) : \ leaf_p = j)$ **else**
            $avg(grad'_{[log_2\sigma_r(i)]}(p) : leaf_p = j, \sigma_p < \sigma_i)$;
        $loss(T_c) = \|\Delta - grad\|_2$
      $T = argmin_{T_c}(loss(T_c))$
**Output:** T

# CatBoost– UpdModel

**Input:** $Mode, leaf, T_t, \{b_j^t\}_j, S, S', grad, grad', \sigma_r{}_{r=1}^{s}$

**foreach** *leaf j in T* **do**

**foreach** $i\, s.t.\, leaf_{r,j} = j$ **do**

$S_0(i) = S_0(i) + \alpha b_j$;

**for** r = 1 to s **do**

**foreach** $i\, s.t.\, leaf_{r,j} = j$ **do**

**if** Mode == Plain **then**

$S_r(i) = S_r(i) - \alpha avg(grad_r(p)\, for\, p :\ leaf_{r,p} = j)$;

**else: for** l = 1 to $[log_2 n]$ **do**

$S'_{r,l}(i) = S'_{r,l}(i) - \alpha avg(grad_r(p) :\ leaf_{r,p} = j), \sigma_r(p) < 2^l$;

$S_r(i) = S'_{r,[log_2 \sigma_r(i)]}(i)$; **Output:** $S, S'$

# CatBoost– Categorical Feature

(1) Greedy TBS

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：

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

拉普拉斯平滑处理：

$$\hat{x}_i^k = \frac{\sum_{x_j \in D_k}^{n} y_i I(x_j^k = x_i^k) + aP}{\sum_{x_j \in D_k}^{n} I(x_j^k = x_i^k) + a}$$

(2) Holdout TBS: Where $D = D_1$

(3) Leave one out TBS

(4) Orderd TBS