1 树模型的演化过程

树模型（Tree-based Models）是一类重要的机器学习方法，经历了从简单的决策树到复杂的集成学习模型的发展过程，主要演化阶段如下：

- **阶段 1：决策树（Decision Tree）**
- **阶段 2：Bagging 方法（如随机森林）**
- **阶段 3：Boosting 方法（如 GBDT, XGBoost, LightGBM）**

1.1 决策树（Decision Tree）

决策树是最基础的树模型，利用特征分裂将数据集划分为多个子集，最终形成树状结构。常见算法包括 ID3、C4.5、CART，主要使用如下标准进行划分：

信息增益（Information Gain）：

$$IG(D, A) = H(D) - \sum_{v \in A} \frac{|D_v|}{|D|} H(D_v)$$

其中，熵的定义为：

$$H(D) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

基尼指数（Gini Index）：

$$Gini(D) = 1 - \sum_{i=1}^{c} p_i^2$$

优缺点：

- 优点：简单易解释，适合小数据集。
- 缺点：容易过拟合，对噪声敏感。

1.2 随机森林（Random Forest）

随机森林是一种 Bagging 方法，通过构建多个独立的决策树并进行投票或平均以提高性能。其预测方式如下：

分类任务：

$$\hat{y} = \arg\max \sum_{b=1}^{B} \mathbb{1}(f_b(x) = y)$$

回归任务：

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} f_b(x)$$

优缺点：

- 优点：减少方差，防止过拟合。
- 缺点：计算复杂度高，解释性较差。

1.3 GBDT（梯度提升决策树）

GBDT 通过迭代训练弱学习器（回归树）来最小化损失函数，公式如下：

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

其中：
- γ_m 为学习率 - $h_m(x)$ 为第 m 棵回归树

—

2 XGBoost 详细解析

2.1 目标函数

XGBoost 的目标函数由损失项与正则化项组成：

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

正则化项：

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j} w_j^2$$

2.2 二阶近似优化

XGBoost 使用二阶导数进行优化：

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[g_i f(x_i) + \frac{1}{2} h_i f^2(x_i) \right] + \Omega(f)$$

其中：

$$g_i = \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i}, \quad h_i = \frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}_i^2}$$

关键特性：

- 分裂查找优化（近似贪心算法）
- 正则化（L_1 和 L_2）
- 并行计算
- 学习率缩减（Shrinkage）

—

3 LightGBM 详细解析

3.1 直方图分桶

LightGBM 使用直方图分桶方法加速特征分裂：

$$O(n \log n) \to O(n)$$

3.2 叶子优先生长

与 XGBoost 的层级生长不同，LightGBM 采用叶子优先策略：

- 增长增益最高的叶节点，减少整体误差。
- 可能导致过拟合，需要正则化处理。

关键特性：

- 自动类别特征处理
- 内存高效存储
- GPU 加速

4 XGBoost vs LightGBM 对比总结

4.1 XGBoost 的损失函数设计

XGBoost 采用加法模型，通过最小化以下正则化损失函数进行训练：

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

其中：

- $l(y_i, \hat{y}_i)$ 表示损失函数，通常为均方误差（MSE）或对数损失（Log Loss）。
- 正则化项：

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_j w_j^2$$

其中 T 是叶节点数量，γ 控制叶子数量的惩罚，λ 控制叶子权重的正则化。

4.1.1 XGBoost 使用的损失函数类型

回归任务（平方误差损失）：

$$l(y, \hat{y}) = (y - \hat{y})^2$$

$$g_i = -2(y_i - \hat{y}_i), \quad h_i = 2$$

分类任务（对数损失）：

$$l(y, \hat{y}) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

$$g_i = \hat{y}_i - y_i, \quad h_i = \hat{y}_i(1 - \hat{y}_i)$$

4.1.2 二阶泰勒展开优化

XGBoost 采用二阶泰勒展开将损失函数近似为：

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[g_i f(x_i) + \frac{1}{2} h_i f^2(x_i) \right] + \Omega(f)$$

$$g_i = \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i}, \quad h_i = \frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}_i^2}$$

4.2 LightGBM 的损失函数设计

LightGBM 的目标函数与 XGBoost 类似：

$$\mathcal{L}(F) = \sum_{i=1}^{n} l(y_i, F_{t-1}(x_i) + f_t(x_i)) + \Omega(f_t)$$

4.2.1 LightGBM 使用的损失函数类型

均方误差（MSE）：

$$l(y, \hat{y}) = (y - \hat{y})^2$$

对数损失（Log Loss）：

$$l(y, \hat{y}) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

Huber 损失：

$$l(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & \text{if } |y - \hat{y}| \leq \delta \\ \delta(|y - \hat{y}| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$

—

4.3 节点分类标准（分裂标准）

4.3.1 XGBoost 的分裂标准

XGBoost 采用增益（Gain）作为节点分裂标准，计算公式如下：

$$Gain = \frac{1}{2}\left[\frac{(\sum_{i\in L} g_i)^2}{\sum_{i\in L} h_i + \lambda} + \frac{(\sum_{i\in R} g_i)^2}{\sum_{i\in R} h_i + \lambda} - \frac{(\sum_{i\in P} g_i)^2}{\sum_{i\in P} h_i + \lambda}\right] - \gamma$$

- L 和 R 分别表示左右子节点。
- P 代表父节点。
- g_i 和 h_i 分别为一阶和二阶导数。
- λ 是正则化参数，γ 控制叶子数目惩罚。

—

4.3.2 LightGBM 的分裂标准

LightGBM 采用与 XGBoost 相似的增益计算方式，但进行了如下优化：

$$Gain = \frac{(G_L)^2}{H_L + \lambda} + \frac{(G_R)^2}{H_R + \lambda} - \frac{(G_P)^2}{H_P + \lambda} - \gamma$$

LightGBM 的改进优化：

- **直方图分桶（Histogram-based Splitting）**通过将连续特征值分桶，大幅减少计算复杂度，将时间复杂度从 $O(n \log n)$ 降低到 $O(n)$。
- **叶子优先分裂（Leaf-wise Splitting）**叶子优先策略优先扩展增益最大的叶子，提高效率，但容易导致过拟合。

—

4.4 XGBoost 与 LightGBM 的对比

特性	XGBoost	LightGBM
损失函数优化	二阶泰勒展开	二阶泰勒展开 + 梯度约束
分裂标准	增益最大化	增益最大化 + 直方图优化
计算优化	并行计算	直方图分桶，降低计算复杂度
分裂策略	层级生长（Level-wise）	叶子优先生长（Leaf-wise）
适用场景	小中型数据集，解释性强	大规模数据集，训练速度更快

表 1: XGBoost 与 LightGBM 的对比

总结：

- XGBoost 适合小中型数据，具有较强的泛化能力和解释性。
- LightGBM 适合大数据集，计算速度快，但可能容易过拟合。

5 GBDT、XGBoost 和 LightGBM 训练过程

5.1 GBDT（梯度提升决策树）

GBDT 通过逐步拟合残差的方式构建回归树。其目标函数为：

$$F_t(x) = F_{t-1}(x) + \gamma_t h_t(x)$$

其中：

- $F_{t-1}(x)$ 为上一轮预测值。
- γ_t 为学习率。
- $h_t(x)$ 为本轮拟合的回归树。

残差计算：

$$r_i^{(t)} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}$$

最终预测结果：

$$\hat{y} = F_T(x)$$

5.1.1 GBDT 代码实现

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

class GBDT:
    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=3):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.trees = []

    def fit(self, X, y):
        F = np.full_like(y, np.mean(y), dtype=np.float64)
        for _ in range(self.n_estimators):
            residuals = y - F
            tree = DecisionTreeRegressor(max_depth=self.max_depth)
            tree.fit(X, residuals)
            self.trees.append(tree)
            F += self.learning_rate * tree.predict(X)

    def predict(self, X):
        F = np.mean(y)
        for tree in self.trees:
            F += self.learning_rate * tree.predict(X)
        return F
```

—

5.2 XGBoost

XGBoost 的目标函数如下：

$$\mathcal{L}(F) = \sum_{i=1}^{n} l(y_i, F_{t-1}(x_i) + f_t(x_i)) + \Omega(f_t)$$

正则化项：

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_j w_j^2$$

使用二阶泰勒展开进行近似：

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

叶子节点的最优权重：

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

增益计算：

$$G = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in j} g_i)^2}{\sum_{i \in j} h_i + \lambda} + \gamma T$$

5.2.1 XGBoost 代码实现

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

class XGBoost:
    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=3, reg_lambda=1):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.reg_lambda = reg_lambda
```

```
        self.trees = []

    def fit(self, X, y):
        F = np.full_like(y, np.mean(y), dtype=np.float64)
        for _ in range(self.n_estimators):
            grad = F - y  # 一阶梯度
            hess = np.ones_like(y)  # 二阶梯度
            tree = DecisionTreeRegressor(max_depth=self.max_depth)
            tree.fit(X, -grad / (hess + self.reg_lambda))
            self.trees.append(tree)
            F += self.learning_rate * tree.predict(X)

    def predict(self, X):
        F = np.mean(y)
        for tree in self.trees:
            F += self.learning_rate * tree.predict(X)
        return F
```

—

5.3 LightGBM

LightGBM 采用直方图分桶策略，目标函数为：

$$\mathcal{L}(F) = \sum_{i=1}^{n} l(y_i, F_{t-1}(x_i) + f_t(x_i)) + \Omega(f_t)$$

直方图分桶复杂度降低：

$$O(n \log n) \to O(n)$$

LightGBM 采用叶子优先策略，生长最大增益的叶子节点，而非层级生长。

5.3.1 LightGBM 代码实现

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

class LightGBM:
    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=3, num_leaves=31):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.num_leaves = num_leaves
        self.trees = []

    def fit(self, X, y):
        F = np.full_like(y, np.mean(y), dtype=np.float64)
        for _ in range(self.n_estimators):
            residuals = y - F
            tree = DecisionTreeRegressor(max_depth=self.max_depth, max_leaf_nodes=self.num_leaves)
            tree.fit(X, residuals)
            self.trees.append(tree)
            F += self.learning_rate * tree.predict(X)

    def predict(self, X):
        F = np.mean(y)
        for tree in self.trees:
            F += self.learning_rate * tree.predict(X)
        return F
```

—

总结：

- GBDT：通过逐步拟合残差，适合中小规模数据。
- XGBoost：使用二阶导数，提供更强的正则化能力，适合大数据。
- LightGBM：基于直方图分桶，训练速度更快，适合超大规模数据。