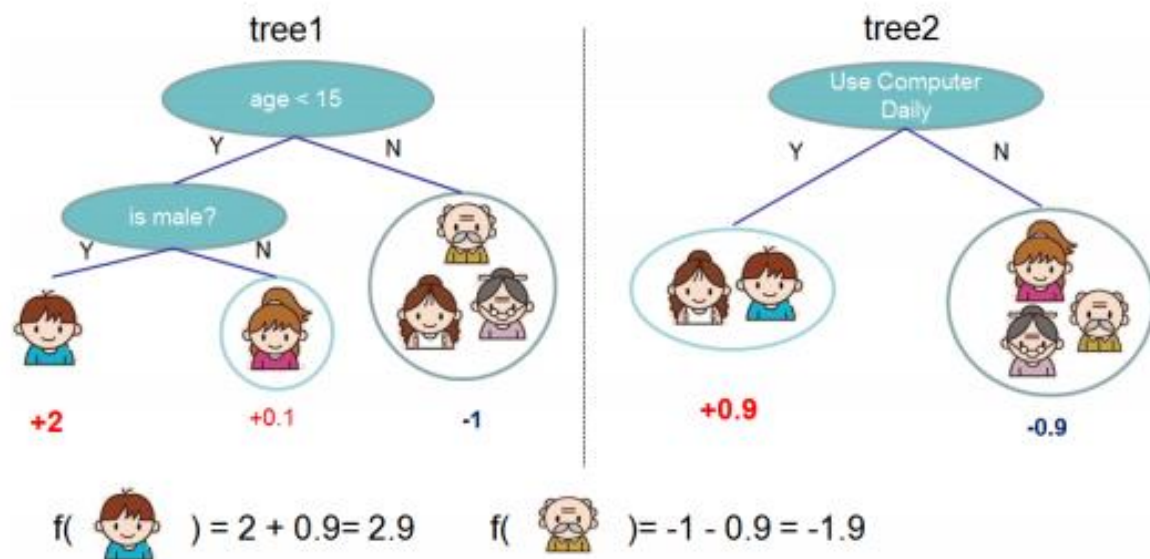


What is XGBoost?

XGBoost是陈天奇等人开发的一个开源机器学习项目，高效地实现了GBDT算法并进行了算法和工程上的许多改进，被广泛应用在Kaggle竞赛及其他许多机器学习竞赛中并取得了不错的成绩。

说到XGBoost，不得不提GBDT(Gradient Boosting Decision Tree)。因为XGBoost本质上还是一个GBDT，但是力争把速度和效率发挥到极致，所以叫X (Extreme) GBoosted。包括前面说过，两者都是boosting方法。



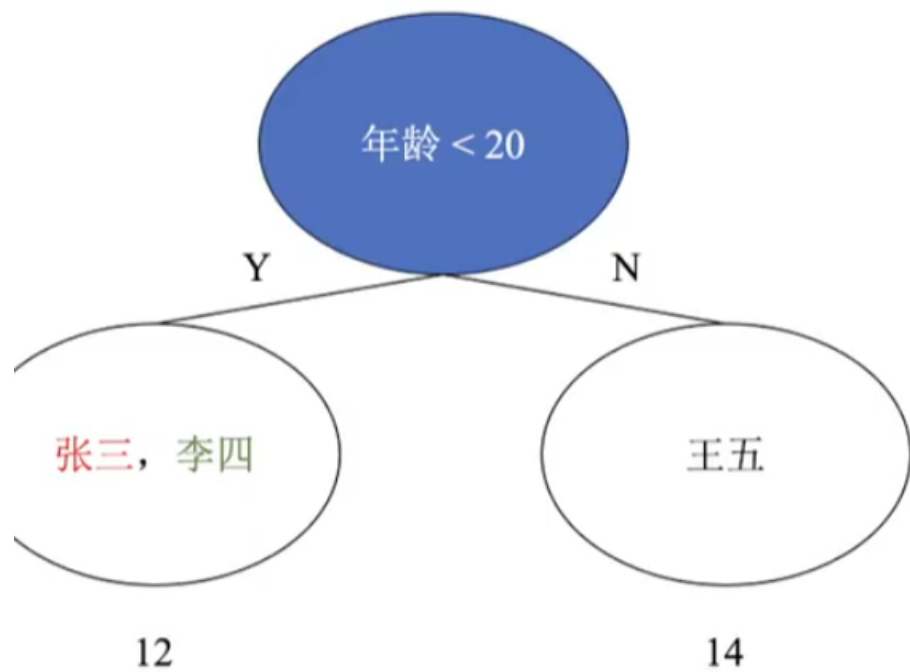
Xgboost推理过程 – boosting流程

最终预测结果 = $\hat{y}_1 + \hat{y}_2 + \hat{y}_3$

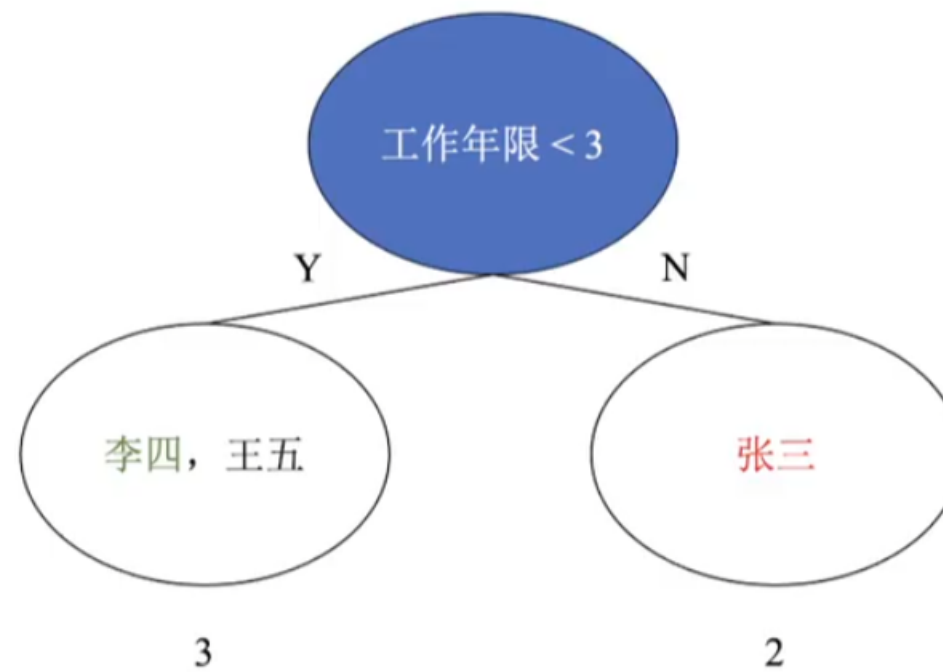
年龄	工作年限	收入(k)
20	2	10
22	3	13
25	6	15
24	2	13
28	3	18
23	2	12
25	5	16

Model ₁	Model ₂	Model ₃	最终预测
9	1	0	10
11	3	-0.5	12.5
10	4	0.5	14.5
11	3	0	14
12	5	2	19
12	1	0	13
18	1	-2	17

使用多棵树预测



对张三薪资预测: $12+2=14$



对李四薪资预测: $12+3=15$

2个好奇

- 1，树结构怎么来的？
- 2，怎么训练这个模型？

必须先看理论。

总览:

1, 推理公式、

2, 目标函数

目标函数的构建

假设有 K 棵树:

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$$

目标函数:

$$obj = \underbrace{\sum_{i=1}^n \mathcal{L}(y_i, \hat{y}_i)}_{\text{损失函数}} + \underbrace{\sum_{k=1}^K \Omega(f_k)}_{\text{控制复杂度}}$$

某棵树的目标函数

目标函数 – 公式分析

Additive Training (叠加式训练)

设第 k 棵树对于样本 x_i 的输出值为 $f_k(x_i)$ ，则训练到第 k 棵树时，样本 x_i 的预测值为：

$$\begin{aligned}\hat{y}_i^{(k)} &= f_1(x_i) + f_2(x_i) + \cdots + f_k(x_i) \\ &= \sum_{j=1}^{k-1} f_j(x_i) + f_k(x_i) \\ &= \hat{y}_i^{(k-1)} + f_k(x_i)\end{aligned}$$

稍微变换一下目标函数

此时，训练的目标函数为： $obj = \sum_{i=1}^n \mathcal{L}(y_i, \hat{y}_i^{(k)}) + \sum_{j=1}^k \Omega(f_j)$

目标函数 - 公式分析

Additive Training (叠加式训练)

$$obj = \sum_{i=1}^n \mathcal{L}(y_i, \hat{y}_i^{(k)}) + \sum_{j=1}^k \Omega(f_j)$$

$$\text{minimize} \quad = \sum_{i=1}^n \mathcal{L}(y_i, \boxed{\hat{y}_i^{(k-1)}} + f_k(x_i)) + \boxed{\sum_{j=1}^{k-1} \Omega(f_j)} + \Omega(f_k)$$

已知 已知

当训练第 k 棵树时

$$\text{minimize} \quad \sum_{i=1}^n \mathcal{L}(y_i, \hat{y}_i^{(k-1)} + f_k(x_i)) + \Omega(f_k)$$

去掉一些与 F_k 无关的常数项

目标函数－公式分析

泰勒展开

$$\begin{aligned} obj &= \sum_{i=1}^n \mathcal{L}(y_i, \hat{y}_i^{(k-1)} + f_k(x_i)) + \Omega(f_k) \\ &\cong \sum_{i=1}^n \left[\mathcal{L}(y_i, \hat{y}_i^{(k-1)}) + \partial_{\hat{y}_i^{(k-1)}} \left(\mathcal{L}(y_i, \hat{y}_i^{(k-1)}) \right) \cdot f_k(x_i) \right. \\ &\quad \left. + \frac{1}{2} \partial_{\hat{y}_i^{(k-1)}}^2 \left(\mathcal{L}(y_i, \hat{y}_i^{(k-1)}) \right) \cdot f_k^2(x_i) \right] + \Omega(f_k) \end{aligned}$$

$$\text{minimize} \quad obj = \sum_{i=1}^n \left[\underline{g_i \cdot f_k(x_i)} + \frac{1}{2} \underline{h_i \cdot f_k^2(x_i)} \right] + \underline{\Omega(f_k)}$$

套用一个泰勒展开的数学公式。

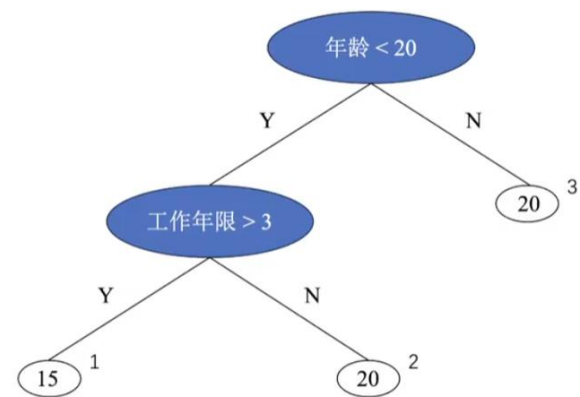
泰勒二阶展开：

$$f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$$

转换后会发现，**g**和**h**是可以计算出来的，每个样本一个

换视角：样本的累计 -> 叶子节点的累计

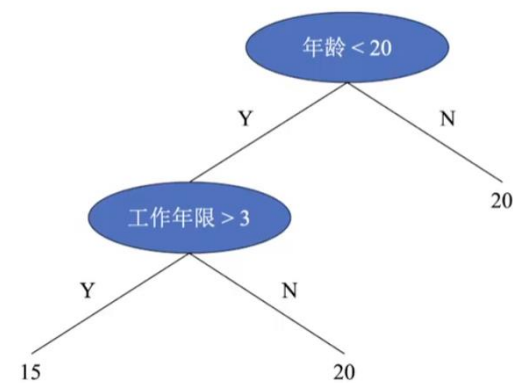
重新定义一棵树



w_j : j 位置叶节点的权重
Leaf value: $w = (w_1, w_2, w_3) = (15, 20, 20)$
 $q(x)$: 样本 x 的位置
 $f_k(x_i) = w_{q(x_i)}$
 $I_j = \{i | q(x_i) = j\}$
eg. $I_3 = \{3, 4, 11, 20, \dots\}$

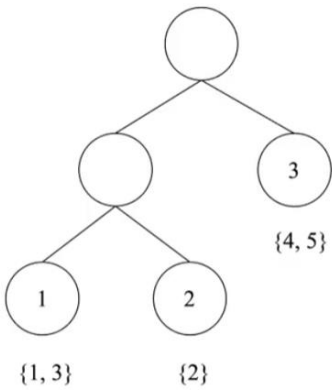
树的复杂度 = 叶节点个数 + 叶节点权重

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



新的目标函数

$$\begin{aligned} obj &= \sum_{i=1}^n \left[g_i \cdot f_k(x_i) + \frac{1}{2} h_i \cdot f_k^2(x_i) \right] + \Omega(f_k) \\ &= \sum_{i=1}^n \left[g_i \cdot w_{q(x_i)} + \frac{1}{2} h_i \cdot w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[\sum_{i \in I_j} g_i \cdot w_j + \frac{1}{2} \sum_{i \in I_j} h_i \cdot w_j^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) \cdot w_j + \frac{1}{2} \left(\left(\sum_{i \in I_j} h_i \right) + \lambda \right) \cdot w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^T \left[G_j \cdot w_j + \frac{1}{2} (H_j + \lambda) \cdot w_j^2 \right] + \gamma T \end{aligned}$$



换个视角看公式

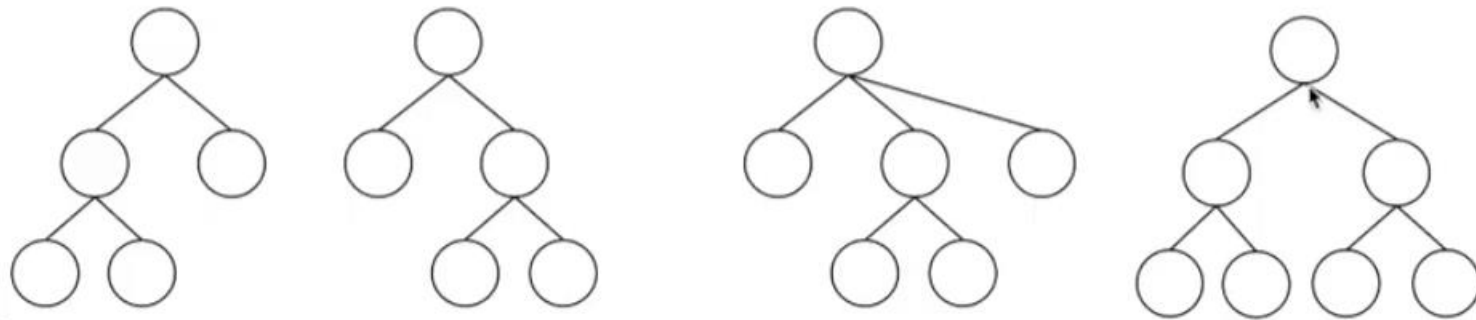
让目标函数达到最小的话， w 取值是可计算的，最小目标函数取值也是可计算的

新的目标函数

$$obj = \sum_{j=1}^T \left[G_j \cdot w_j + \frac{1}{2} (H_j + \lambda) \cdot w_j^2 \right] + \gamma T$$

$$\text{当 } w_j = -\frac{G_j}{(H_j + \lambda)} \text{ 时, } obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{(H_j + \lambda)} + \gamma T$$

obj 代表当我们知道任意一个树的形状时，其损失值最小可以是多少



让目标函数达到最小的 w 取值是明确的

分裂是否值得？看分裂后目标函数变小了没。

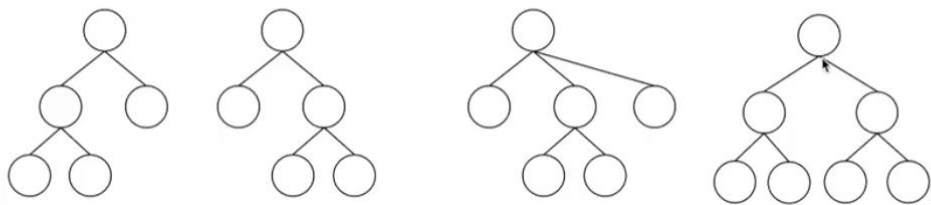
新的目标函数

$$obj = \sum_{j=1}^T \left[G_j \cdot w_j + \frac{1}{2} (H_j + \lambda) \cdot w_j^2 \right] + \gamma T$$

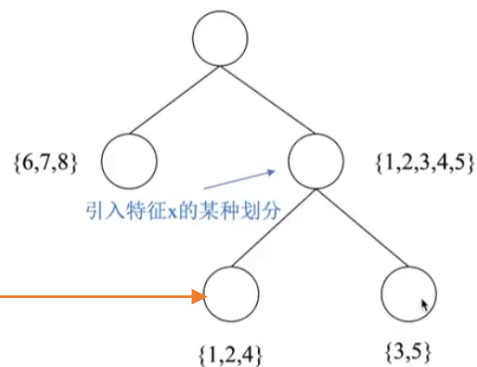
当 $w_j = -\frac{G_j}{(H_j + \lambda)}$ 时,

$$obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{(H_j + \lambda)} + \gamma T$$

obj 代表当我们知道任意一个树的形状时，其损失值最小可以是多少



如何寻找树的形状



$$obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{(H_j + \lambda)} + \gamma T$$

$$obj_{old} = -\frac{1}{2} \left[\frac{G_6^2 + G_7^2 + G_8^2}{H_6 + H_7 + H_8 + \lambda} + \frac{G_1^2 + G_2^2 + G_3^2 + G_4^2 + G_5^2}{H_1 + H_2 + H_3 + H_4 + H_5 + \lambda} \right] + 2\gamma$$

$$obj_{new} = -\frac{1}{2} \left[\frac{G_6^2 + G_7^2 + G_8^2}{H_6 + H_7 + H_8 + \lambda} + \frac{G_1^2 + G_2^2 + G_4^2}{H_1 + H_2 + H_4 + \lambda} + \frac{G_3^2 + G_5^2}{H_3 + H_5 + \lambda} \right] + 3\gamma$$

$$obj_{old} - obj_{new} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G_L^2 + G_R^2}{H_L + H_R + \lambda} \right] - \gamma$$

maximize

左子树分数

右子树分数

不分割可以拿到的分数

加入新叶子结点引入的代价

回归场景：用MSE损失函数

```
class SquaredErrorObjective():  
    # Loss  
    # (y_i-pred_i_minus_1)^2 / 2  
    def loss(self, y, pred):  
        return np.mean(((y - pred)**2)*0.5)  
  
    # Loss关于Pred的一阶导数  
    def gradient(self, y, pred):  
        return pred - y  
  
    # Loss关于Pred的二阶导数  
    def hessian(self, y, pred):  
        return np.ones(len(y))
```

$$MSE_{loss} = \frac{1}{2} \cdot \sum_{i=1}^n (y_i - p_i)^2$$
$$p_i = x_i \cdot w + b$$

$$\frac{\delta loss}{\delta w} = 2 \cdot \frac{1}{2} \cdot \sum_{i=1}^n (y_i - p_i) \cdot (-1) \cdot x_i = \sum_{i=1}^n (p_i - y_i) \cdot x_i$$

$$\frac{\delta loss}{\delta b} = 2 \cdot \frac{1}{2} \cdot \sum_{i=1}^n (y_i - p_i) \cdot (-1) = \sum_{i=1}^n (p_i - y_i)$$

二分类场景：用CrossEntropy损失函数

```
# 补充实现BCE二值交叉熵损失【二分类问题】
class BinaryCrossEntropyObjective():
    # BCE损失函数与导数: https://blog.csdn.net/zzl12880/article/details/128403845
    def loss(self, y, pred):
        return -(y * np.log(1/(1+np.exp(-pred))) + (1 - y) * np.log(1 - np.exp(-pred)))

    # Loss关于Pred的一阶导数
    def gradient(self, y, pred):
        return 1/(1+np.exp(-pred)) - y

    # Loss关于Pred的二阶导数
    def hessian(self, y, pred):
        sigmoid_value= 1/(1+np.exp(-pred))
        return sigmoid_value * (1 - sigmoid_value)
```

2、BCELoss求导

$$\begin{aligned} BEC_{loss} &= - \sum_{i=1}^n [y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)] \\ p_i &= \text{sigmoid}(x_i) = \frac{1}{1 + e^{-x_i}} \\ \frac{\delta p_i}{\delta x_i} &= p_i \cdot (1 - p_i) \\ \frac{\delta loss}{\delta x_i} &= \frac{\delta loss}{\delta p_i} \cdot \frac{\delta p_i}{\delta x_i} = - \sum_{i=1}^n (y_i \cdot \frac{1}{p_i} + (1 - y_i) \cdot \frac{1}{p_i - 1}) \cdot p_i \cdot (1 - p_i) = \sum_{i=1}^n (p_i - y_i) \\ \text{所以: } \frac{\delta loss}{\delta x_i} &= \sum_{i=1}^n (p_i - y_i) \end{aligned}$$