

# XGBoost实验

时间： 2020/12/15

# 算法简介

## XGBoost : eXtreme Gradient Boosting

- 最初是一个研究项目，由当时在Distributed (Deep) Machine Learning Community (DMLC) 组中的**陈天奇**负责。
- 在数据科学方面，有大量的kaggle比赛选用XGBoost进行数据挖掘比赛；
  - 2015年的kaggle上发布的29个获奖算法中，有17个使用了XGBoost
  - 2015年的KDDcup中，前十名的获胜团队均使用了XGBoost

KDD 2016

## **XGBoost: A Scalable Tree Boosting System**

Tianqi Chen  
University of Washington  
tqchen@cs.washington.edu

Carlos Guestrin  
University of Washington  
guestrin@cs.washington.edu

# 算法简介

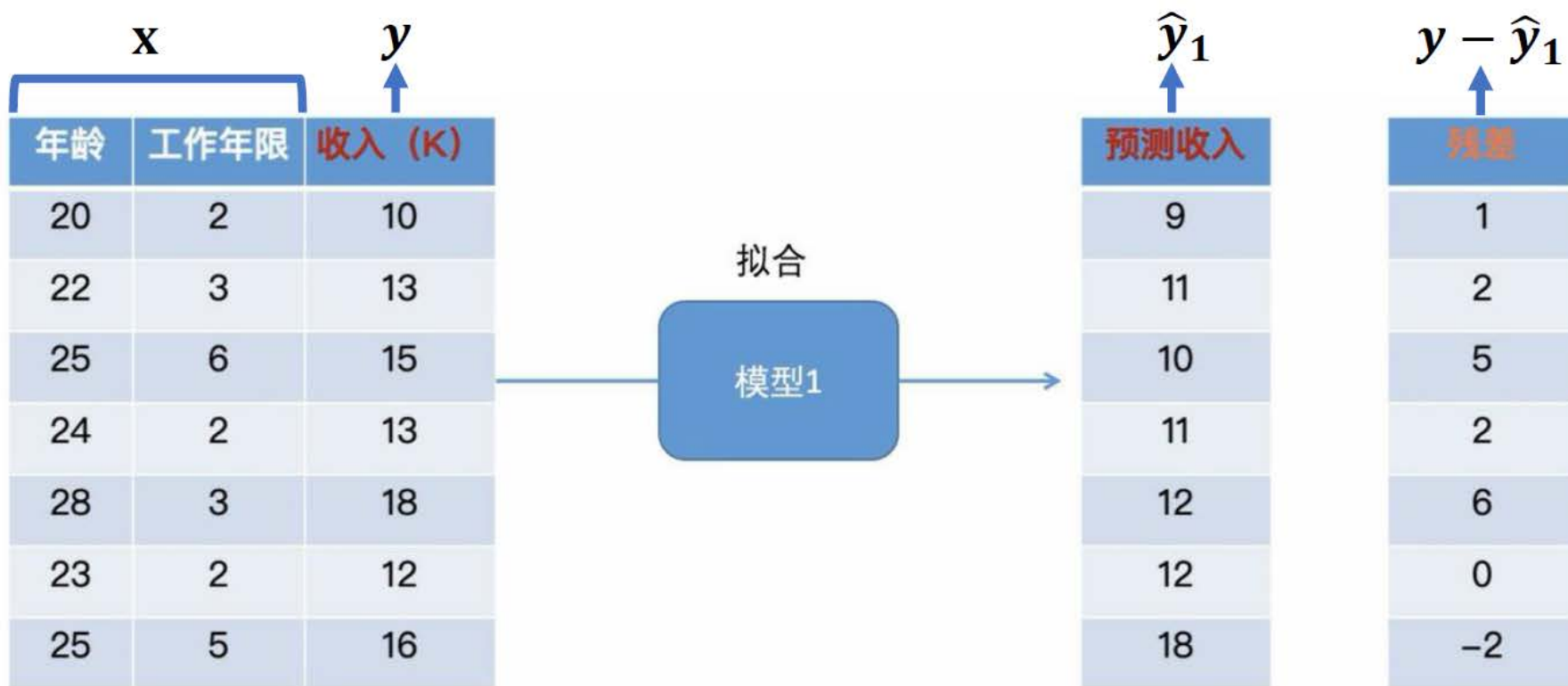
## 提升树

给定一个预测问题，我们在此数据上训练了一个模型，记作Model 1，但是效果不好，误差比较大。

**问题：**如果我们想在这个模型的基础上继续做下去，并且不去改变Model 1，那该怎么做？

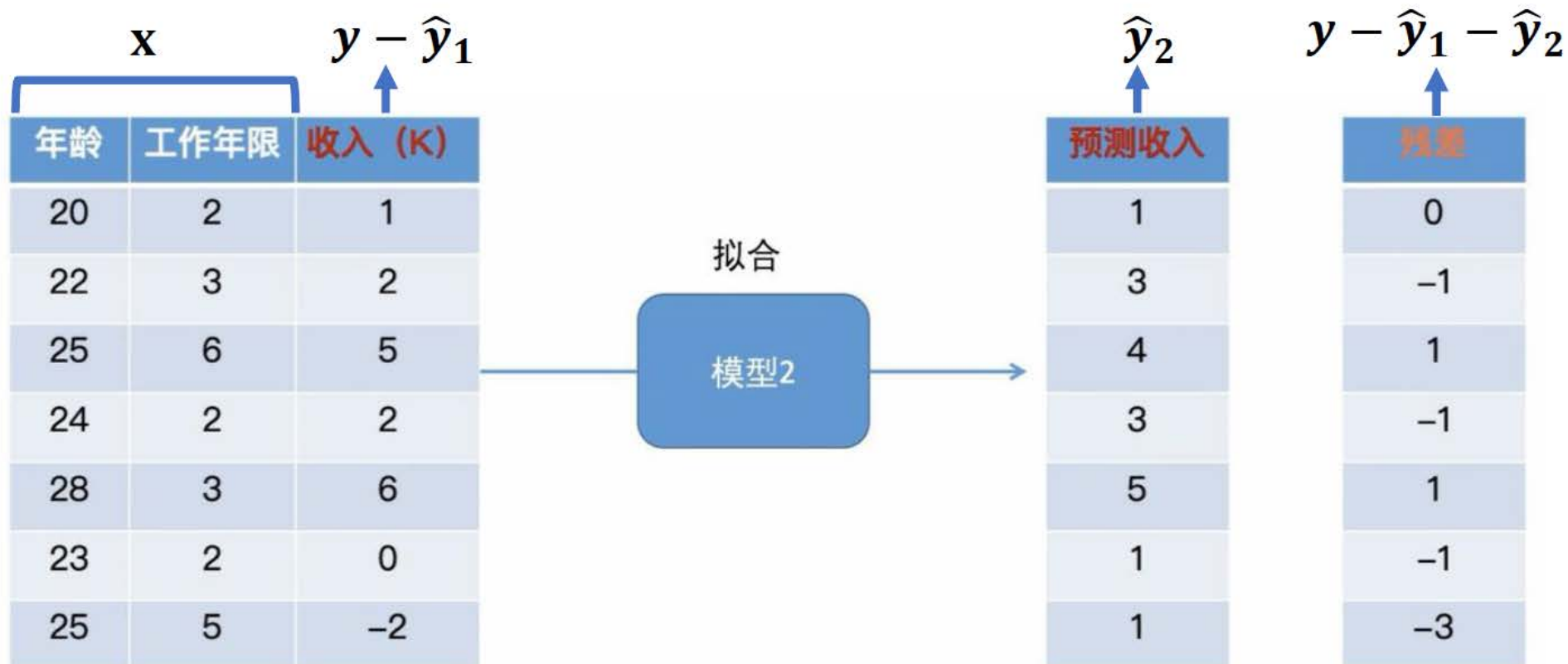
# 算法简介

## 提升树——基于残差的训练



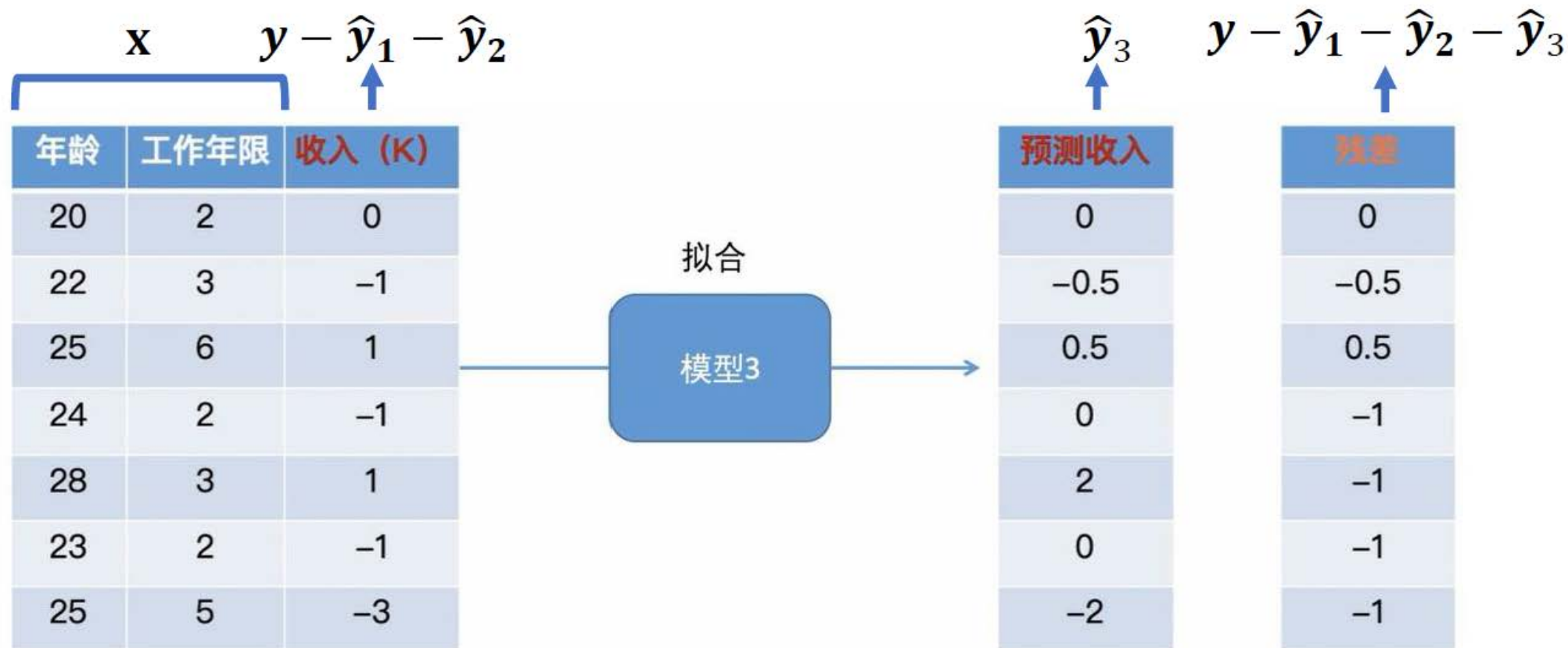
# 算法简介

## 提升树——基于残差的训练



# 算法简介

## 提升树——基于残差的训练



# 算法简介

## 最终的预测

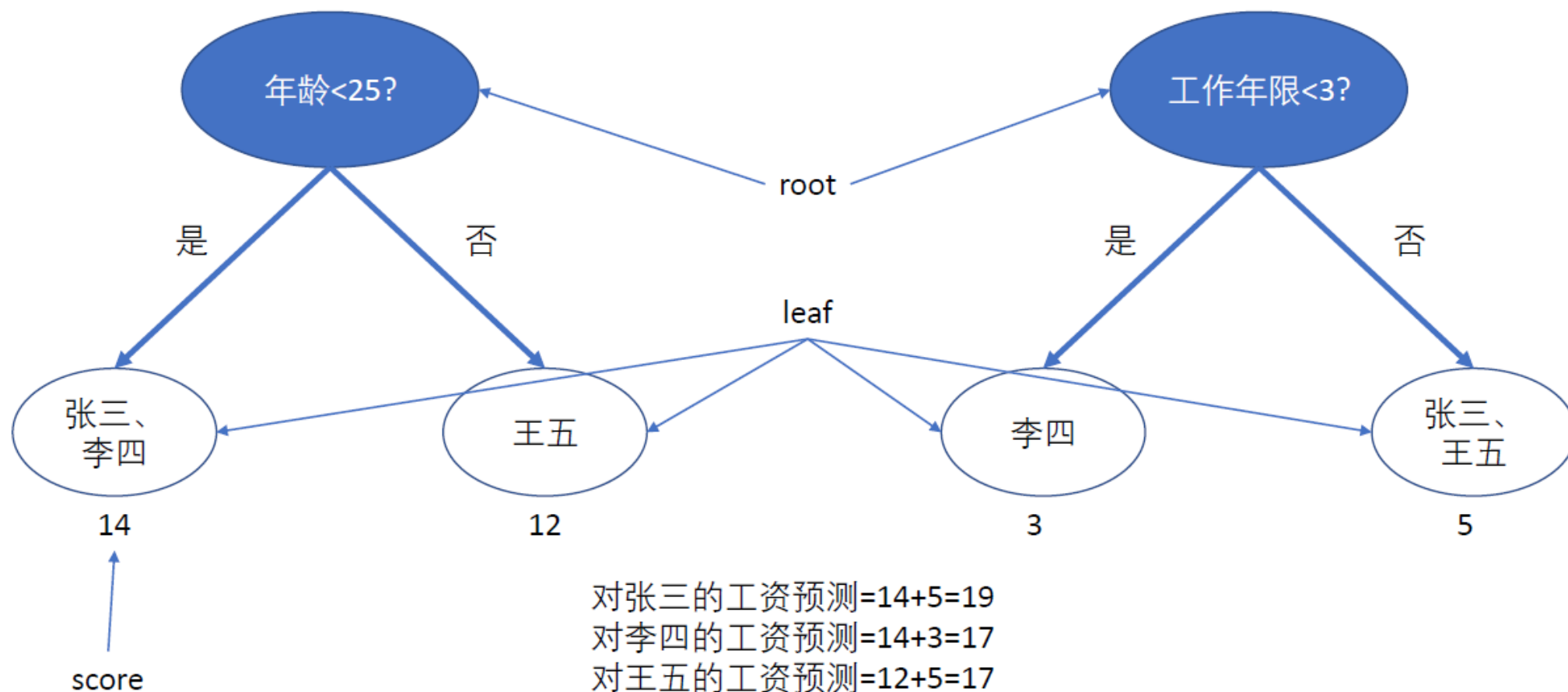
最终预测=模型1的预测+模型2的预测+模型3的预测

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

模型1		模型2		模型3		最终预测
9		1		0		10
11		3		-0.5		13.5
10		4		0.5		14.5
11	+	3	+	0	=	14
12		5		2		19
12		1		0		13
18		1		-2		17

# 算法简介

使用多颗树来进行预测，每个model均为一棵树





# 目标函数

假设已经训练了K颗树，对于第i个样本的预测值为：

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

## 目标函数

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

训练误差


正则化项

- 叶节点个数
- 叶节点评分
- 树的深度
- ...


# 叠加式训练

每一轮都是由上一轮已经固定的预测值，加上本轮新加入的函数共同预测


$$\begin{aligned}\hat{y}_i^{(0)} &= 0 \\ \hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\ \hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\ &\dots \\ \hat{y}_i^{(t)} &= \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)\end{aligned}$$



第t轮的模型预测值



第t-1轮的模型预测值



新的函数

# 叠加式训练

第t轮的预测为:  $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

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

目标: 找到 $f_t$ 使得整个式子最小



# 泰勒展开

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

对 $f_t(x_i)$ 进行二阶泰勒展开:

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

$$\text{令 } g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

则:

$$Obj^{(t)} = \left( \sum_{i=1}^n \left[ l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] \right) + \Omega(f_t) + constant$$

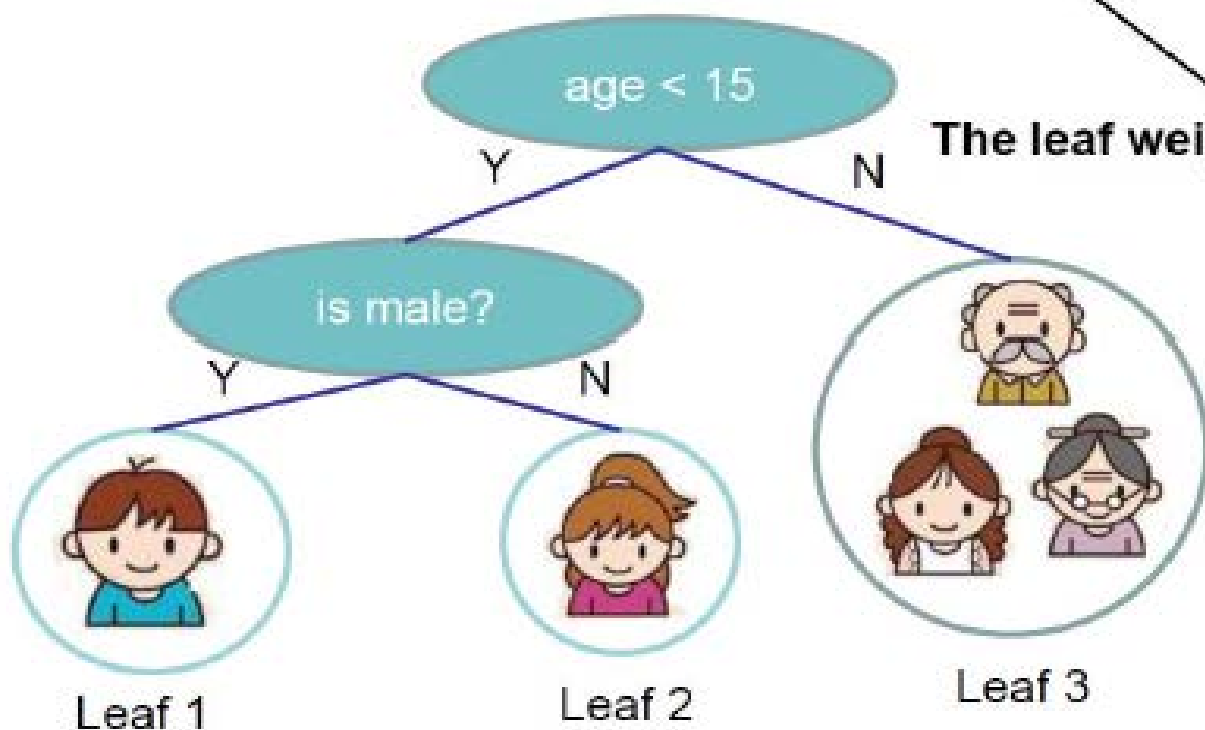
当训练第 $t$ 棵树的时候,  $g_i$ 和 $h_i$ 都是已知的

# 定义一棵树

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q: \mathbf{R}^d \rightarrow \{1, 2, \dots, T\}$$

The structure of the tree

The leaf weight of the tree



$w_1 = +2$

$w_2 = 0.1$

$w_3 = -1$

$$q(\text{boy}) = 1$$

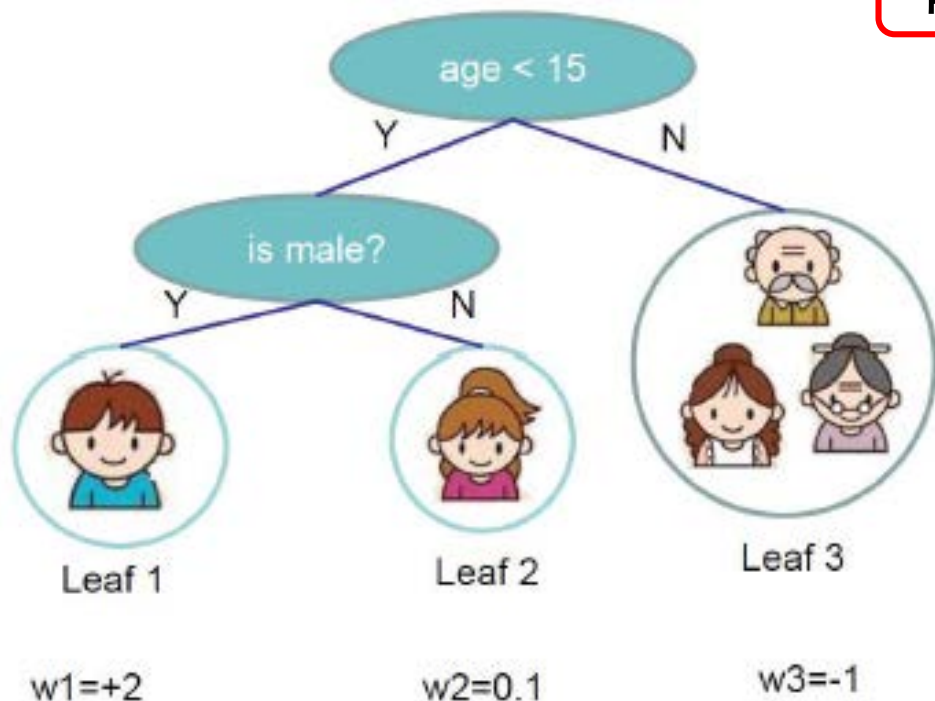
$$q(\text{woman}) = 3$$

# 树的复杂度

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

叶子节点的数量

叶子节点值的L2正则



$$\Omega = \gamma 3 + \frac{1}{2} \lambda (4 + 0.01 + 1)$$

## 新的目标函数

$$\begin{aligned} Obj^{(t)} &= \left( \sum_{i=1}^n \left[ l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] \right) + \Omega(f_t) \\ &= \sum_{i=1}^n \left[ g_i \omega_q(x_i) + \frac{1}{2} h_i \omega_q^2(x_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T \omega_j^2 \\ &= \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) \omega_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i + \lambda \right) \omega_j^2 \right] + \gamma T \end{aligned}$$

则可求得最优值为：

$$w_j^* = -\frac{G_j}{H_j + \lambda} \qquad Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

知道树的结构就可以算出叶节点的score以及当前的目标函数

# 寻找最优的split—贪心算法

在实际中：

树从0开始增长

对于每一个叶子节点，尝试去进行split, 则分裂后的值记为：






$$Gain = \frac{1}{2} \left[ \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

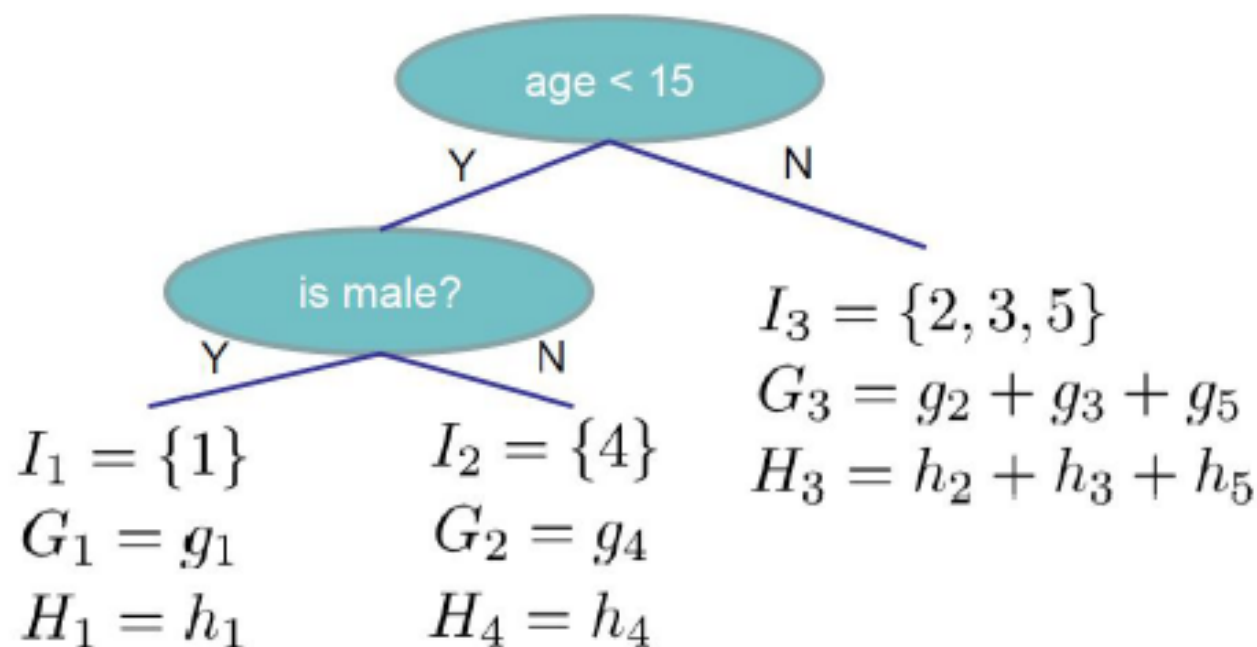
the score of left child      the score of right child      the score of if we do not split      The complexity cost by introducing additional leaf



# 寻找最优的split—贪心算法

Instance index      gradient statistics

1		$g_1, h_1$
2		$g_2, h_2$
3		$g_3, h_3$
4		$g_4, h_4$
5		$g_5, h_5$



$$\mathcal{L}_{split} = \frac{1}{2} \left[ \frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$

# 寻找最优的split—贪心算法

对于每一个节点，枚举出所有的特征值

- 对于每一个特征，将实例按照特征值进行排序
- 对于一个特征，计算出最佳的split点
- 计算所有特征的最佳split点

树的早停：当最佳分裂的增益小于一定的阈值时，停止分裂

# 扩展阅读

贪心算法问题：

- 数据量太大时，无法读入内存进行运算

近似算法

- 对于每个特征，只考察分位点 (quantile) 可以减少计算复杂度
- Global：学习每棵树前就提出候选切分点，并在每次分裂时都采用这种分割
- Local：每次分裂前将重新提出候选切分点
- 一般来说Local策略需要更多的计算步骤，而Global策略因为节点已有划分所以需要更多的候选点

# 扩展阅读

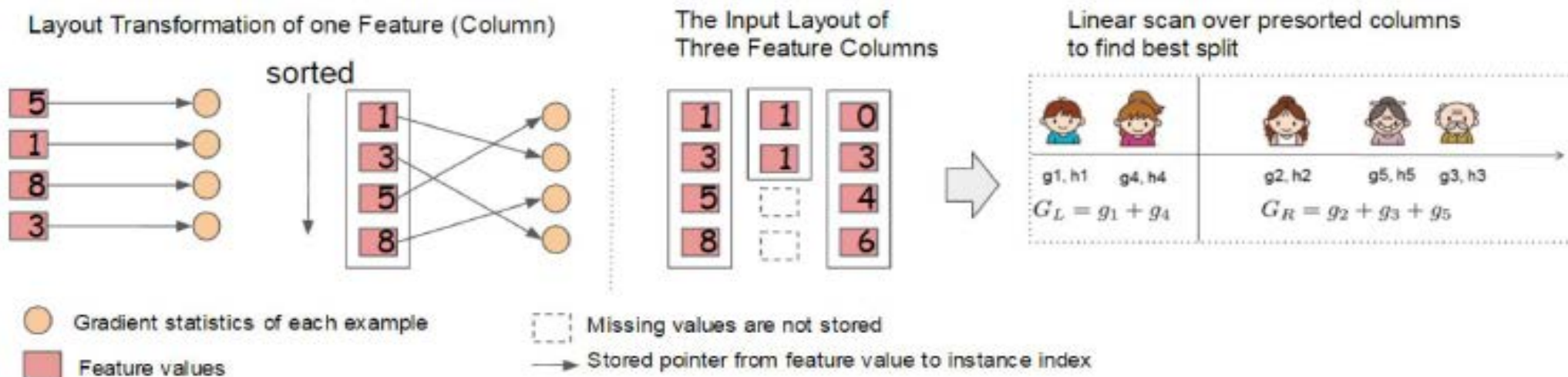
## 稀疏感知

- 实际工程中一般会出现输入值稀疏的情况
  - 数据缺失,
  - 数据本身的分布,
  - 特征工程（如one-hot编码）
  - ...
- XGBoost在构建树的节点过程中只考虑非缺失值的数据遍历，而为每个节点增加了一个缺省方向，当样本相应的特征值缺失时，可以被归类到缺省方向上，最优的缺省方向可以从数据中学到
- 作者通过在Allstate-10K数据集上进行了实验，从结果可以看到稀疏算法比普通算法在处理数据上快了超过50倍

# 扩展阅读

## 列块并行计算

- 在树生成过程中，**最耗时的一个步骤就是在每次寻找最佳分裂点时都需要对特征的值进行排序**
- XGBoost 在训练之前会根据特征对数据进行排序，然后保存到块结构中，并在每个块结构中都采用了稀疏矩阵存储格式（Compressed Sparse Columns Format, CSC）进行存储，后面的训练过程中会重复地使用块结构



# 实验要求

Python实现XGBoost模型

要求：

自己实现xgboost子树结构

按照xgboost的方式对节点进行分裂和计算

数据：

皮马印第安人糖尿病数据，8维特征，二分类任务

Baseline：

测试集精度0.7