Introduction to XGBoost

By 吴剑灿

XGBoost: e**X**treme **G**radient **Boost**ing

- 它最初是一个研究项目，由当时在Distributed (Deep) Machine Learning Community (DMLC)组里的**陈天奇**负责。
- 在数据科学方面，有大量的Kaggle选手选用XGBoost进行数据挖掘比赛，是各大数据科学比赛的必杀武器；
  - 在2015年Kaggle上发布的29个获奖算法中，有17个使用了XGBoost
  - 在2015年KDDCup中，前十名中的每个获胜团队都使用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

Bagging vs Boosting

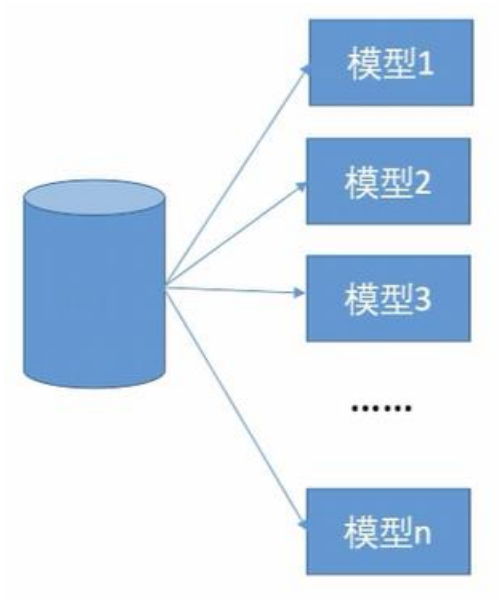
- Bagging

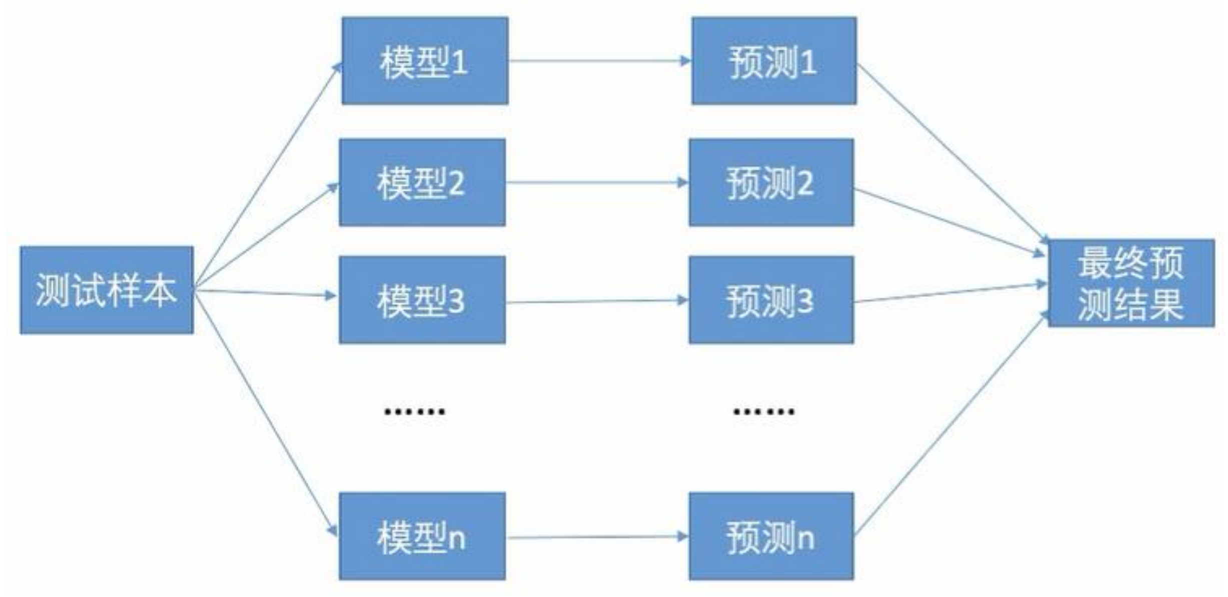
  Leverages unstable base learners that are weak because of overfitting

- Boosting

  Leverage stable base learners that are weak because of underfitting

Bagging的流程





构建多个模型

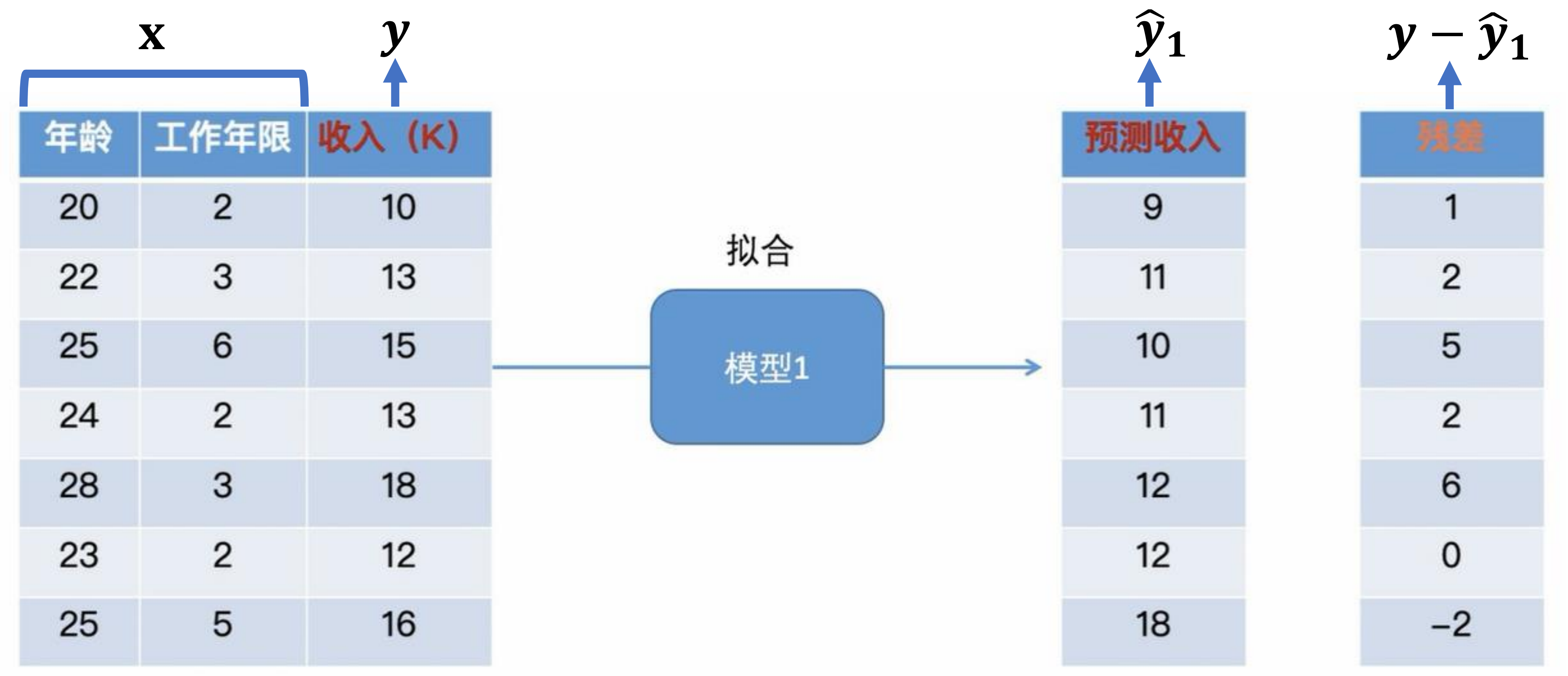
用多个模型预测

提升树

给定一个预测问题，调包侠A在此数据上训练了一个模型，记作Model 1, 但效果不大好，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?

提升树——基于残差的训练

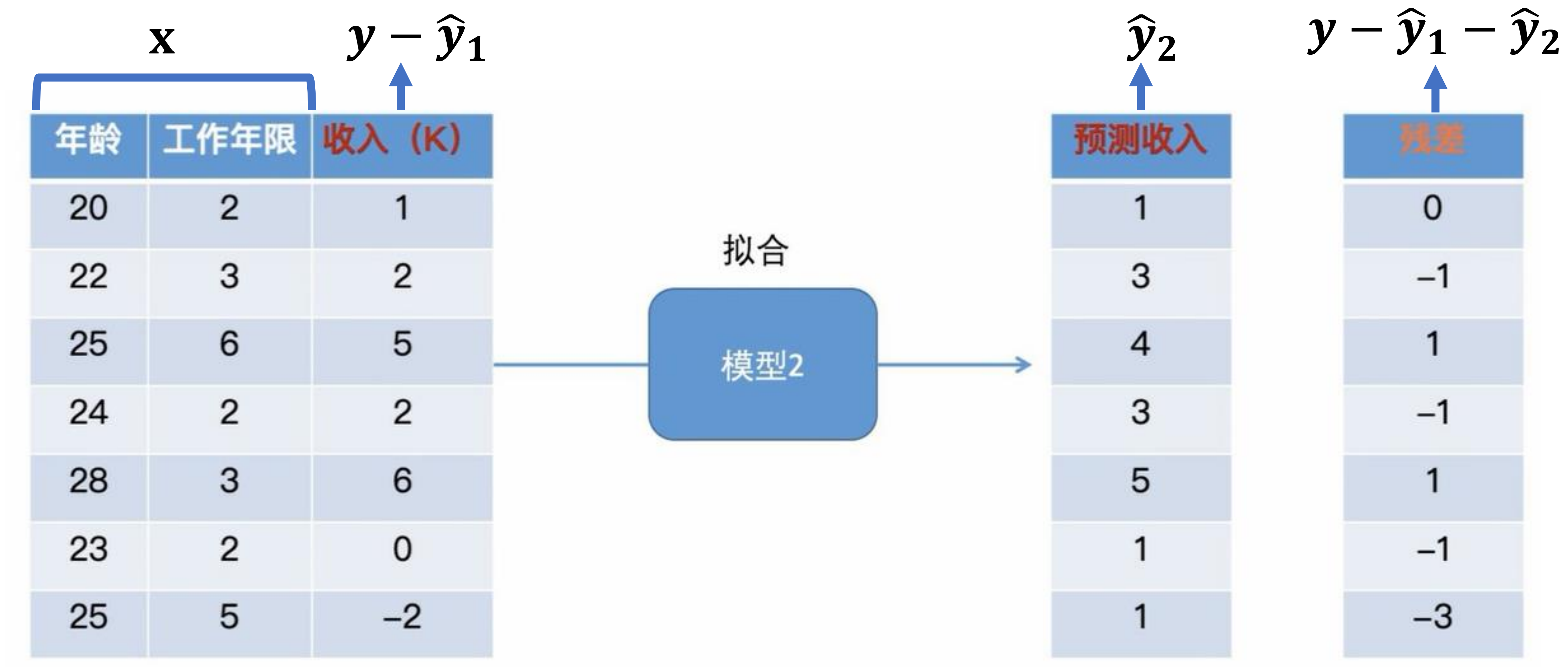


年龄	工作年限	收入（K）
20	2	10
22	3	13
25	6	15
24	2	13
28	3	18
23	2	12
25	5	16

预测收入
9
11
10
11
12
12
18

残差
1
2
5
2
6
0
–2

提升树——基于残差的训练

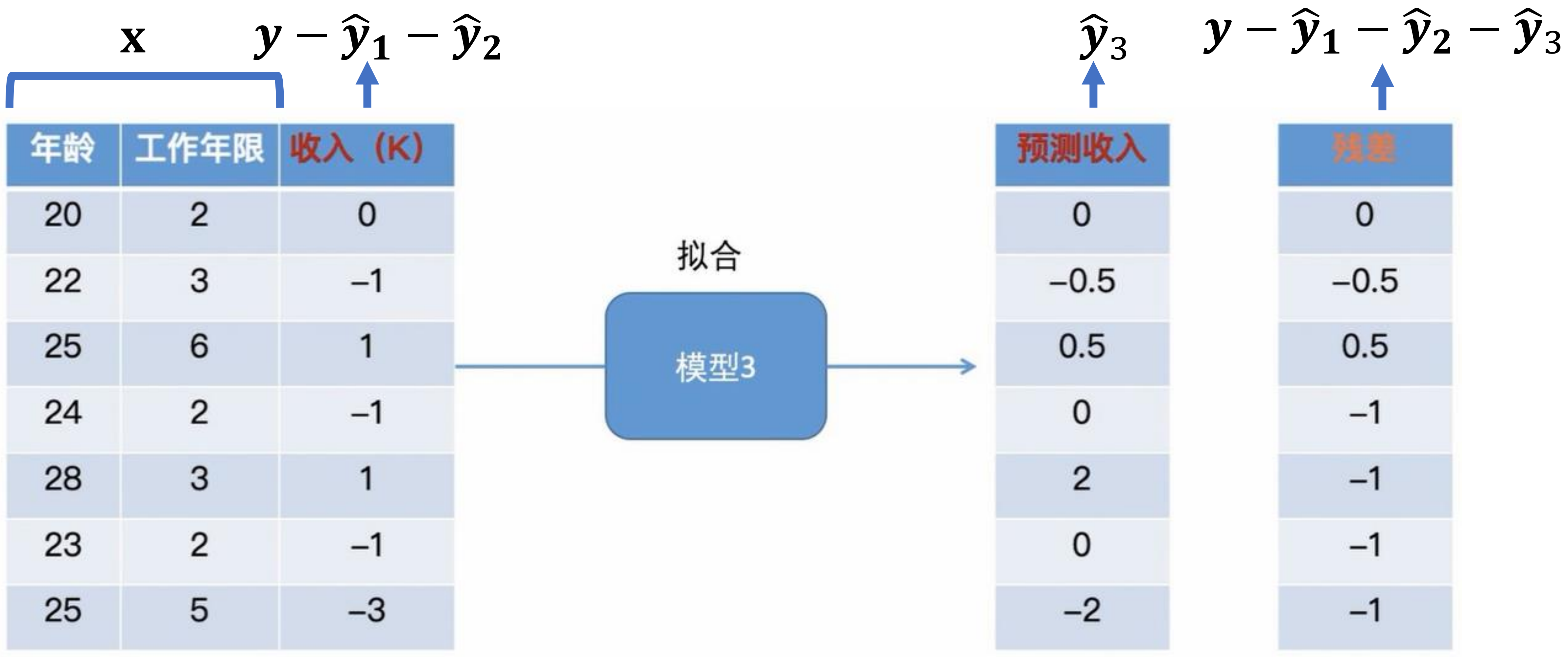


年龄	工作年限	收入（K）
20	2	1
22	3	2
25	6	5
24	2	2
28	3	6
23	2	0
25	5	–2

预测收入
1
3
4
3
5
1
1

残差
0
–1
1
–1
1
–1
–3

提升树——基于残差的训练


x
$y - \hat{y}_1 - \hat{y}_2$
拟合
模型3
$\hat{y}_3$
$y - \hat{y}_1 - \hat{y}_2 - \hat{y}_3$

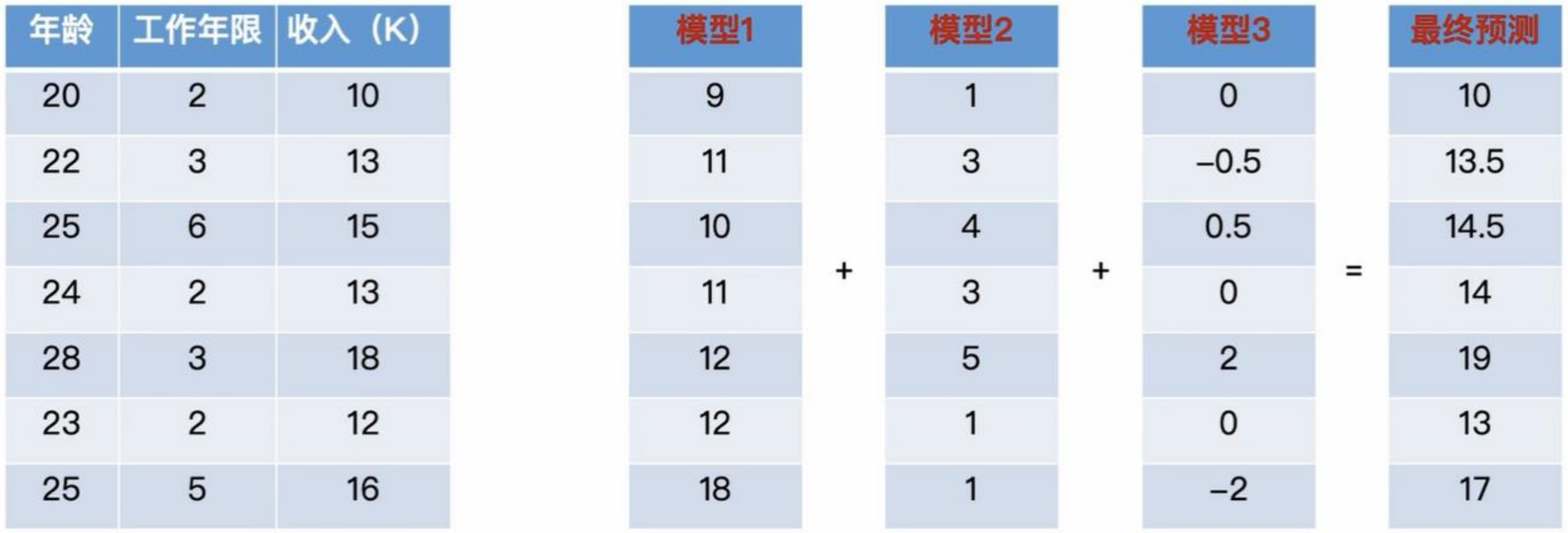
年龄	工作年限	收入（K）
20	2	0
22	3	–1
25	6	1
24	2	–1
28	3	1
23	2	–1
25	5	–3

预测收入
0
–0.5
0.5
0
2
0
–2

残差
0
–0.5
0.5
–1
–1
–1
–1

最终的预测

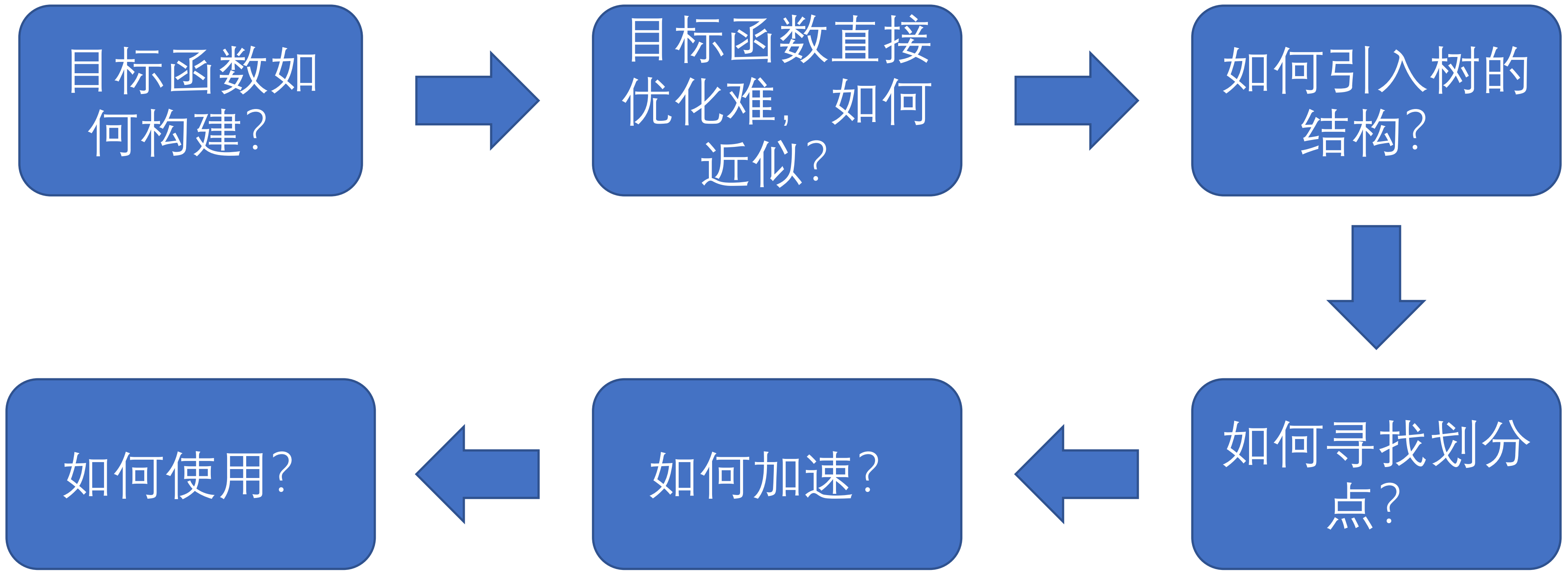
最终预测=模型1的预测+模型2的预测+模型3的预测



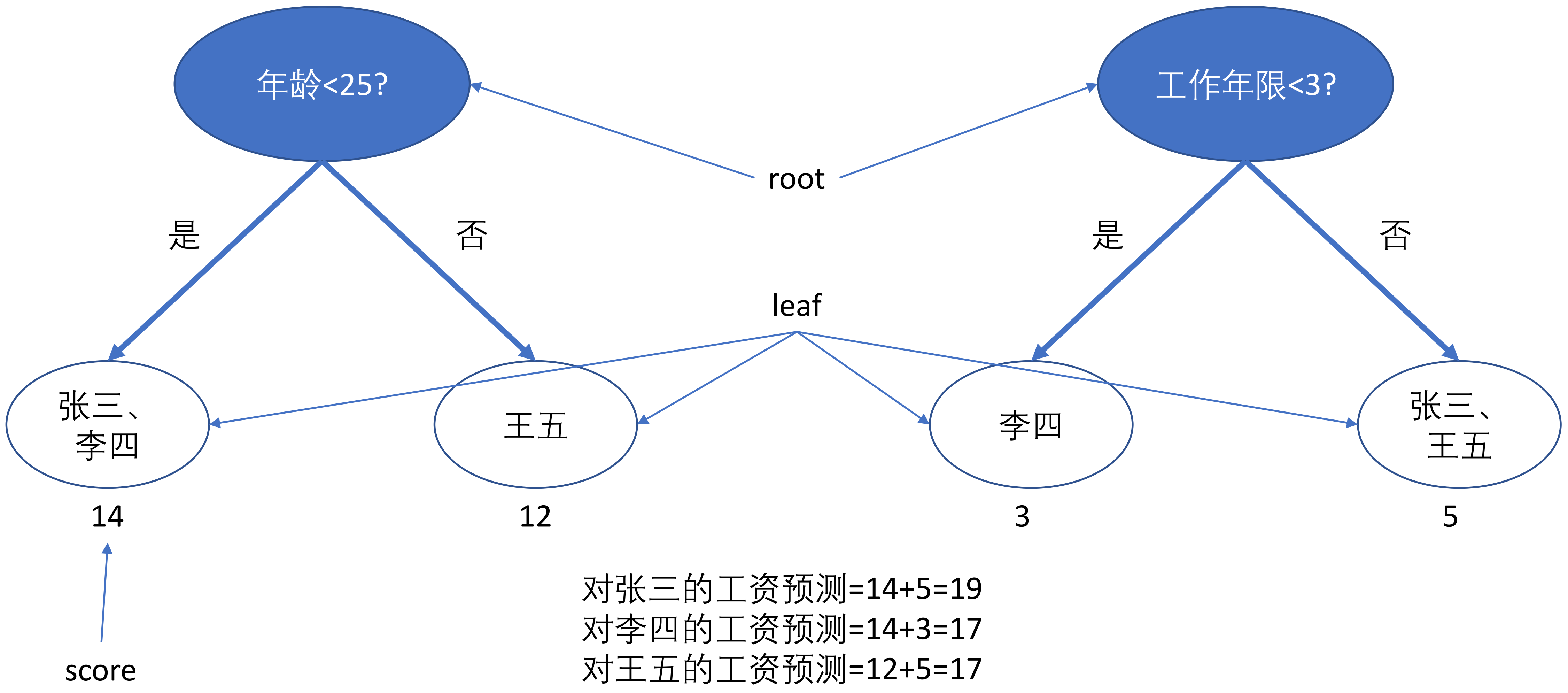
年龄	工作年限	收入（K）
20	2	10
22	3	13
25	6	15
24	2	13
28	3	18
23	2	12
25	5	16

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

接下来。。。



使用多棵树来预测



对张三的工资预测=14+5=19
对李四的工资预测=14+3=17
对王五的工资预测=12+5=17

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

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

Training loss　　Complexity of trees

- 叶节点个数
- 叶节点的评分
- 树的深度
- …

Parameters?

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

Think: regression tree is a function that maps the attributes to the score

- Including structure of each tree, and the score in the leaf
- Or simply use function as parameters

$$\Theta = \{f_1, f_2, \cdots, f_K\}$$

- Instead learning weights in $\mathbf{R}^d$, we are learning functions(trees)

We can not use methods such as SGD, to find f (since they are trees, instead of just numerical vectors)

Addictive training （叠加式训练）

- Start from constant prediction, add a new function each time

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

Model at training round t

Keep functions added in previous round

New function

Addictive training （叠加式训练）

- The prediction at round t is $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

This is what we need to decide in round t

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

Goal: find f_t to minimize this

- Consider square loss

$$Obj^{(t)} = \sum_{i=1}^{n} \left(y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))\right)^2 + \Omega(f_t) + const$$
$$= \sum_{i=1}^{n} \left[\underbrace{2(\hat{y}_i^{(t-1)} - y_i)} f_t(x_i) + f_t(x_i)^2\right] + \Omega(f_t) + const$$

This is usually called residual from previous round

Taylor Expansion Approximation of Loss

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

- Take Taylor expansion of the objective
  - Recall $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$
  - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

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

注：当训练第t棵树的时候，g_i和h_i都是已知的

- *If you are not comfortable with this, think of square loss*

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

定义一棵树

- We define tree by a vector of scores in leaves, and a leaf index mapping function that maps an instance to a leaf

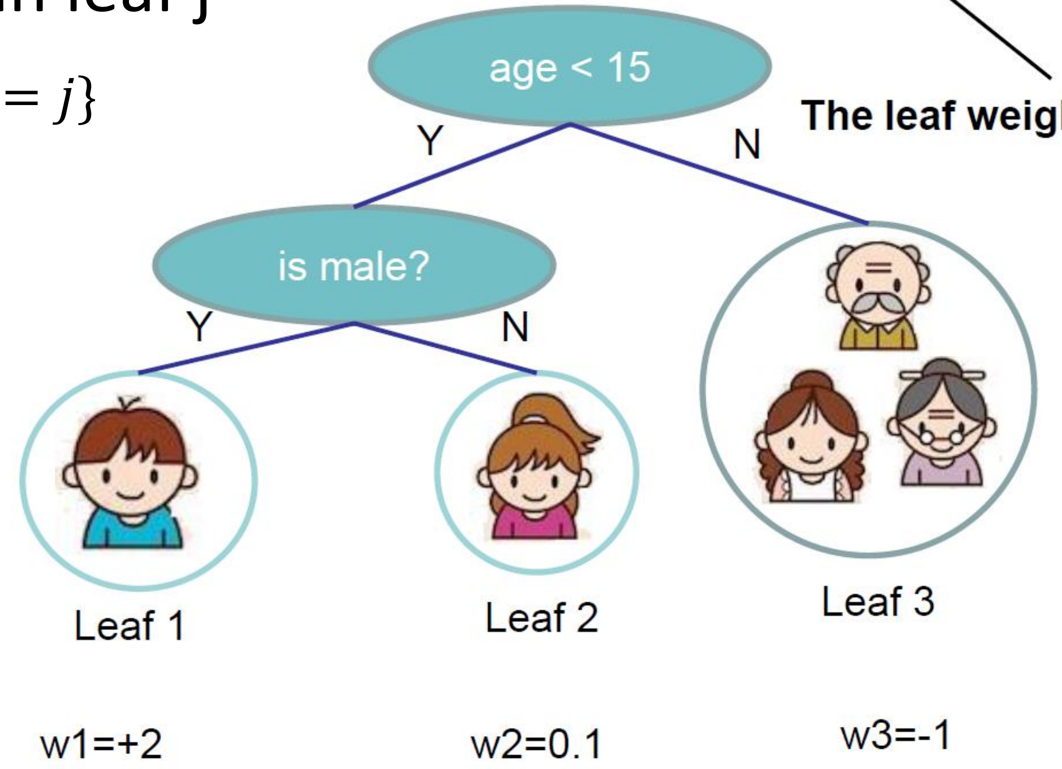
$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \rightarrow \{1, 2, \cdots, T\}$$

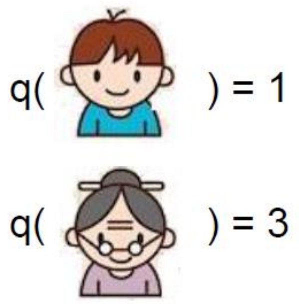
The structure of the tree

The leaf weight of the tree

Instance set in leaf j

$$I_j = \{i | q(i) = j\}$$

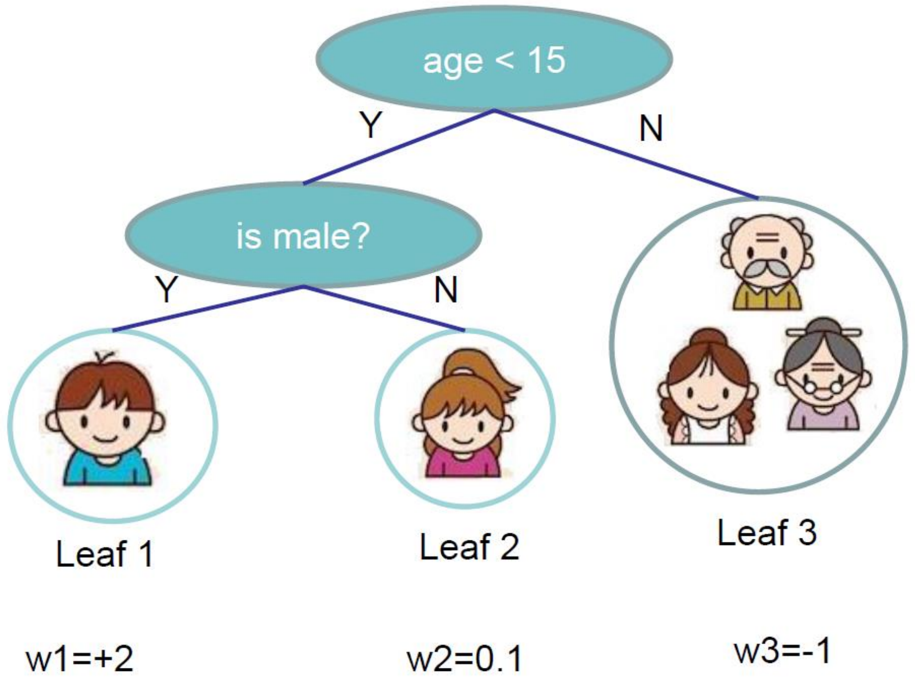




定义树的复杂度

- Define complexity as (this is not the only possible definition)

$$\Omega(f) = \underbrace{\gamma T}_{\text{Number of leaves}} + \underbrace{\frac{1}{2}\lambda\|w\|^2}_{L_2 \text{ norm of leaf scores}}$$



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

新的目标函数

Objective, with constants removed, then regrouping by leaf

$$
\begin{aligned}
Obj^{(t)} &\simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \tfrac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t) \\
&= \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \tfrac{1}{2} h_i w_{q(x_i)}^2\right] + \gamma T + \lambda \tfrac{1}{2} \sum_{j=1}^{T} w_j^2 \\
&= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \tfrac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2\right] + \gamma T
\end{aligned}
$$

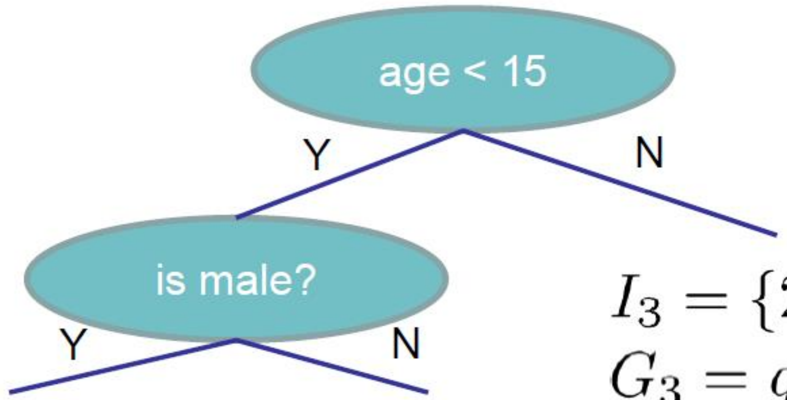
Assume the structure of tree (q(x)) is fixed, the optimal weight in each leaf, and the resulting objective value are

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

<u>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</u>

例子

Instance index		gradient statistics
1		g1, h1
2		g2, h2
3		g3, h3
4		g4, h4
5		g5, h5



$$I_1 = \{1\} \quad G_1 = g_1 \quad H_1 = h_1$$

$$I_2 = \{4\} \quad G_2 = g_4 \quad H_4 = h_4$$

$$I_3 = \{2, 3, 5\} \quad G_3 = g_2 + g_3 + g_5 \quad H_3 = h_2 + h_3 + h_5$$

$$Obj = -\sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

如何寻找树的形状?

- Enumerate the possible tree structures q
- Calculate the structure score for the q, using the scoring eq.

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

- Find the best tree structure, and use the optimal leaf weight

$$w_j^* = -\frac{G_j}{H_j+\lambda}$$

- But... there can be infinite possible tree structures..

寻找最优的split——贪心算法

- In practice, we grow the tree greedily
  - Start from tree with depth 0
  - For each leaf node of the tree, try to add a split. The change of objective after adding the split is

The complexity cost by introducing additional leaf

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

the score of left child

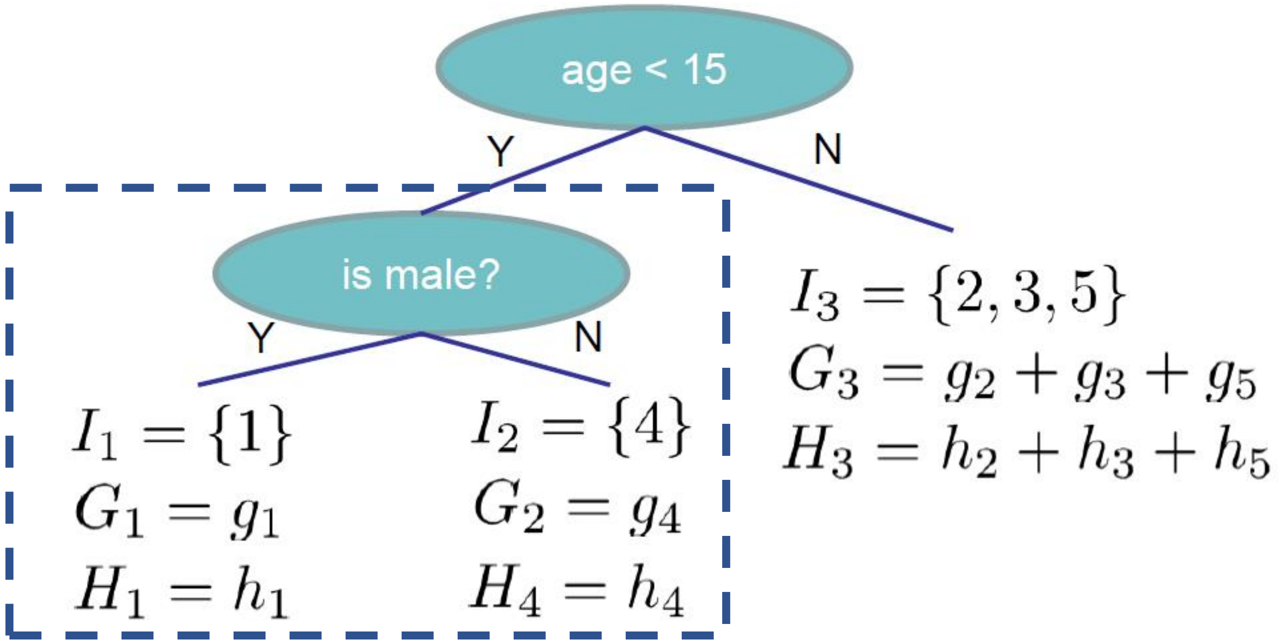
the score of if we do not split

the score of right child

Remaining question: how do we find the best split?

例子

Instance index		gradient statistics
1		g1, h1
2		g2, h2
3		g3, h3
4		g4, h4
5		g5, h5



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

$$I_L = \{1\}, I_R = \{4\}, I = \{1,4\}$$

寻找最优的split——贪心算法

- For each node, enumerate over all features
  - For each feature, sorted the instances by feature value
  - Use a linear scan to decide the best split along that feature
  - Take the best split solution along all the features

- Time Complexity growing a tree of depth K
  - It is O(n d K log n): or each level, need O(n log n) time to sort

    There are d features, and we need to do it for K level

Pruning and Regularization

- Recall the gain of split, it can be negative!

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

  - When **the training loss reduction** is smaller than **regularization**
  - Trade-off between simplicity and predictivness
- Pre-stopping
  - Stop split if the best split have negative gain
  - But maybe a split can benefit future splits..
- Post-Prunning
  - Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain

近似算法

- 贪心算法可能存在的问题：
  - 数据量太大时，无法读入内存进行计算
  - 分布式训练
- 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
  - 问题：如何选择候选分位点？用二阶梯度对样本进行加权？为什么？

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）
  - …

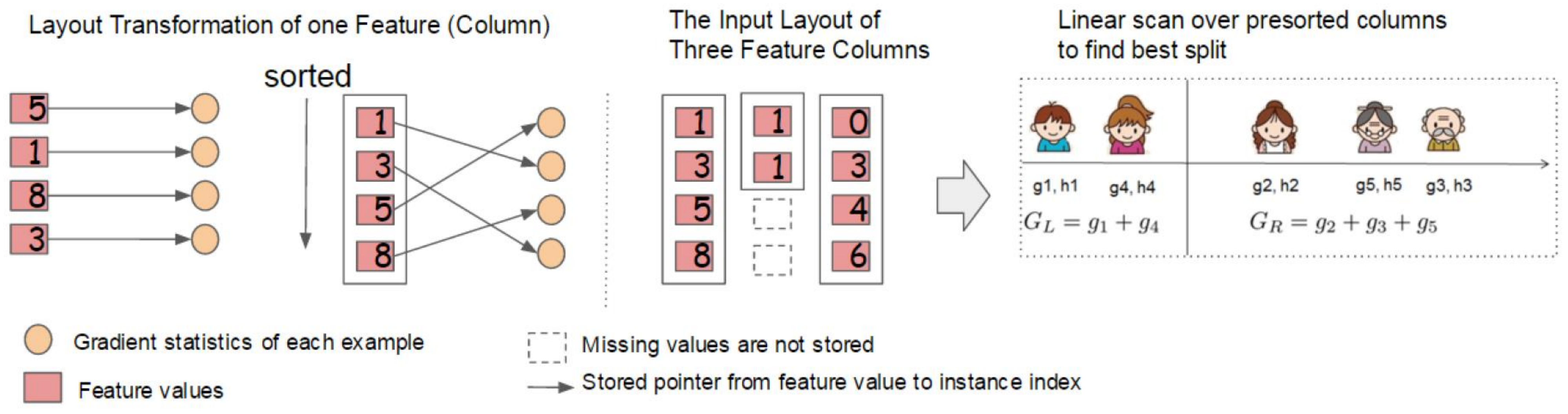
- XGBoost在构建树的节点过程中只考虑非缺失值的数据遍历，而为每个节点增加了一个缺省方向，当样本相应的特征值缺失时，可以被归类到缺省方向上，最优的缺省方向可以从数据中学到

- 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

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



Layout Transformation of one Feature (Column)
sorted
5
1
8
3
1
3
5
8
The Input Layout of Three Feature Columns
1
3
5
8
1
1
0
3
4
6
Linear scan over presorted columns to find best split
g1, h1
g4, h4
g2, h2
g5, h5
g3, h3
$G_L = g_1 + g_4$
$G_R = g_2 + g_3 + g_5$
Gradient statistics of each example
Feature values
Missing values are not stored
Stored pointer from feature value to instance index

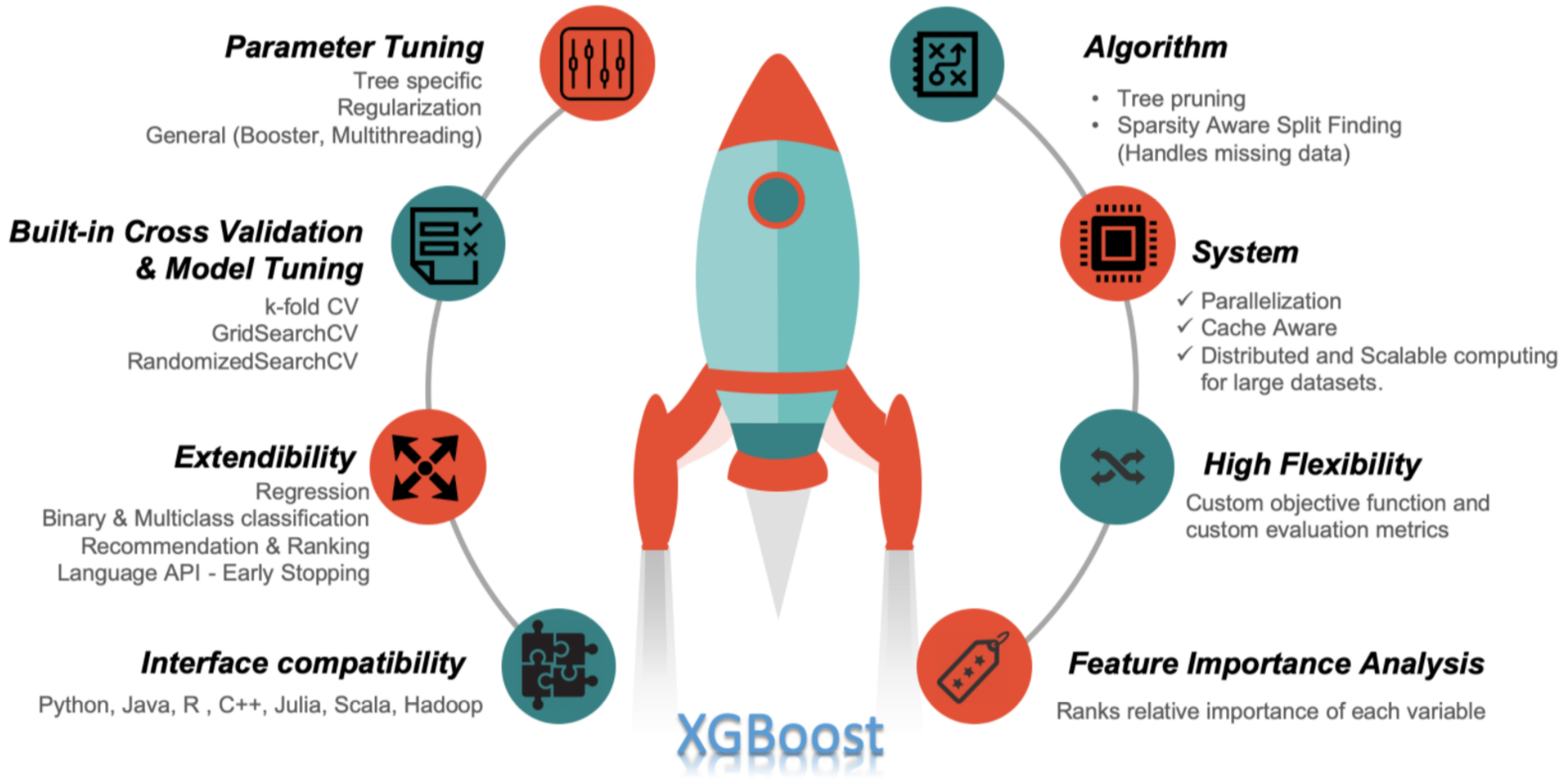
其他优化

- **缓存访问**
  - 为每个线程分配一个连续的缓存区，将需要的梯度信息存放在缓冲区中
  - 实现非连续空间到连续空间的转换，提高了算法效率
- **“核外”块计算**
  - 将数据集分成多个块存放在硬盘中，使用一个独立的线程专门从硬盘读取数据，加载到内存中，这样算法在内存中处理数据就可以和从硬盘读取数据同时进行
  - 块压缩（Block Compression）：对于行索引，只保存第一个索引值，然后用16位的整数保存与该block第一个索引的差值
  - 块分区（Block Sharding ）：将特征block分区存放在不同的硬盘上，以此来增加硬盘IO的吞吐量
- **Shrinkage（缩减）**

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i) \quad \Rightarrow \quad y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$$

  - 相当于学习速率，削弱每棵树的影响，让后面有更大的学习空间
- **列抽样**

优点


Parameter Tuning
Tree specific
Regularization
General (Booster, Multithreading)
Built-in Cross Validation & Model Tuning
k-fold CV
GridSearchCV
RandomizedSearchCV
Extendibility
Regression
Binary & Multiclass classification
Recommendation & Ranking
Language API - Early Stopping
Interface compatibility
Python, Java, R , C++, Julia, Scala, Hadoop
Algorithm
• Tree pruning
• Sparsity Aware Split Finding (Handles missing data)
System
✓ Parallelization
✓ Cache Aware
✓ Distributed and Scalable computing for large datasets.
High Flexibility
Custom objective function and custom evaluation metrics
Feature Importance Analysis
Ranks relative importance of each variable
XGBoost

Thanks