

# 人工智能之机器学习

## XGBoost

主讲人：李老師

## 课程内容

- XGBoost概述
- XGBoost安装
- XGBoost原理讲解
- XGBoost项目案例

## XGBoost概述

- XGBoost是GBDT算法的一种变种，是一种常用的有监督集成学习算法；是一种伸缩性强、便捷的可并行构建模型的Gradient Boosting算法。
- XGBoost官网：
  - <http://xgboost.readthedocs.io>;
- XGBoost Github源码位置
  - <https://github.com/dmlc/xgboost>;
- XGBoost支持开发语言：Python、R、Java、Scala、C++等。

## XGBoost安装

- 安装方式一：
  - 编译Github上的源码，参考<http://xgboost.readthedocs.io/en/latest/build.html>
- 安装方式二：
  - python的whl文件进行安装，要求python版本3.5或者3.6；下载链接：  
<https://www.lfd.uci.edu/~gohlke/pythonlibs/#xgboost>；安装参考命令：pip install f:///xgboost-0.7-cp36-cp36m-win\_amd64.whl
- 安装方式三：
  - 直接pip命令安装：pip install xgboost

**Xgboost**, a distributed gradient boosting (GBDT, GBRT or GBM) library.

[xgboost-0.7-cp35-cp35m-win32.whl](#)

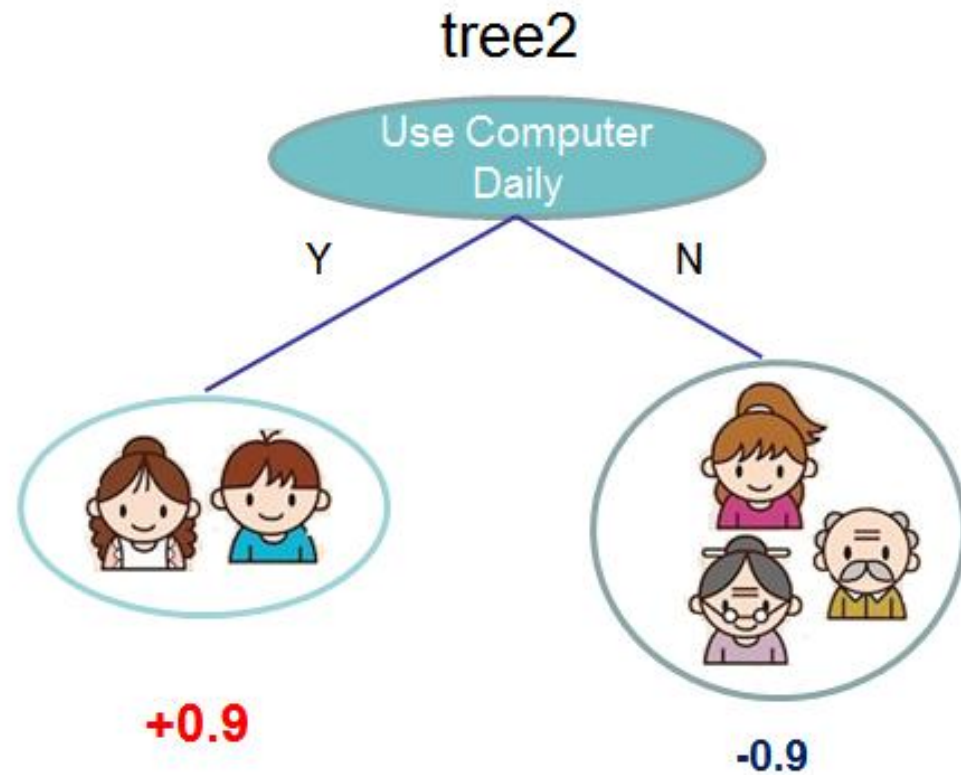
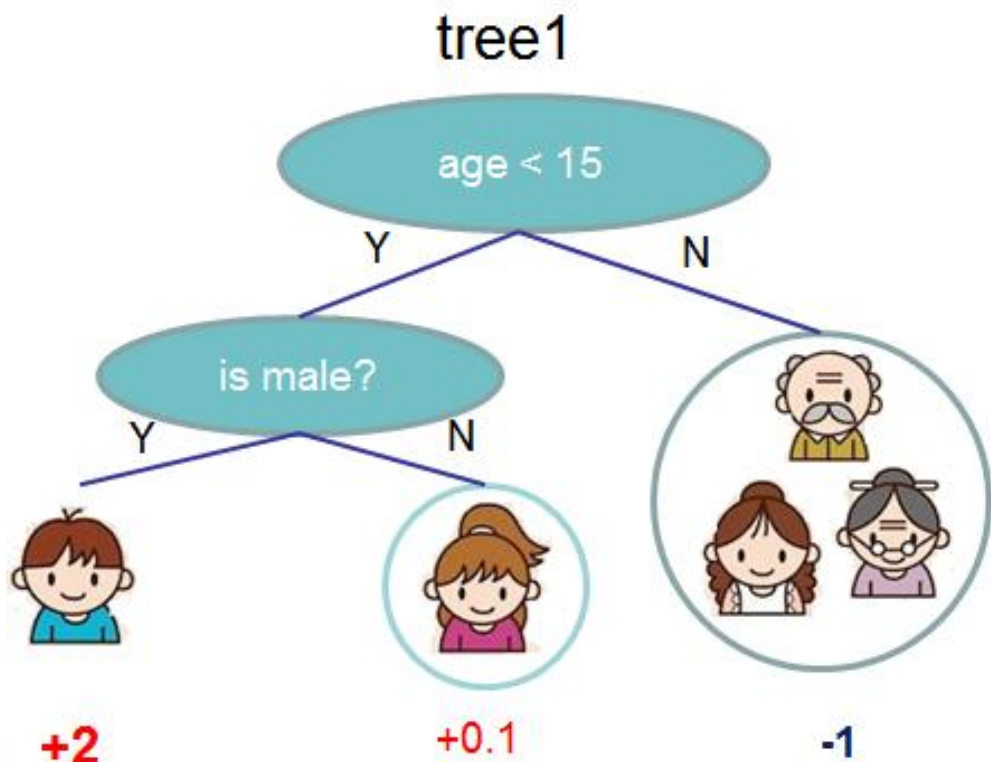
[xgboost-0.7-cp35-cp35m-win\\_amd64.whl](#)

[xgboost-0.7-cp36-cp36m-win32.whl](#)

[xgboost-0.7-cp36-cp36m-win\\_amd64.whl](#)

```
C:\Users\ibf>python
Python 3.6.0 |Anaconda 4.3.1
Type "help", "copyright", "c
>>> import xgboost as xgb
>>>
```

# CART, GBDT



$$f(\text{young man}) = 2 + 0.9 = 2.9$$

$$f(\text{elderly man}) = -1 - 0.9 = -1.9$$

# 模型

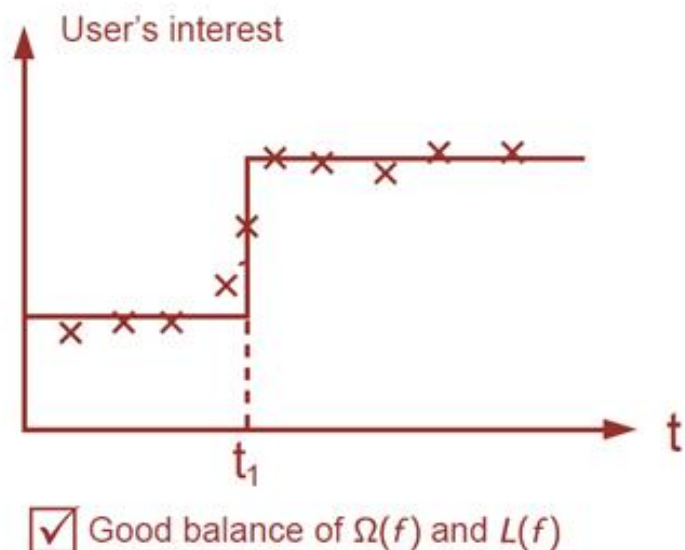
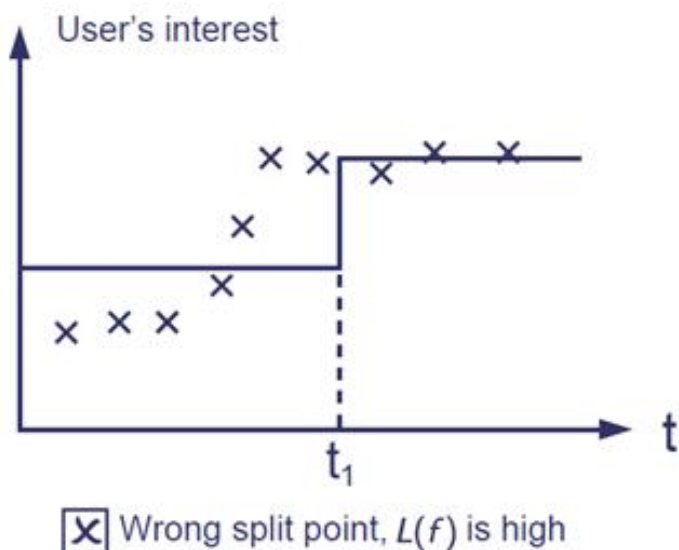
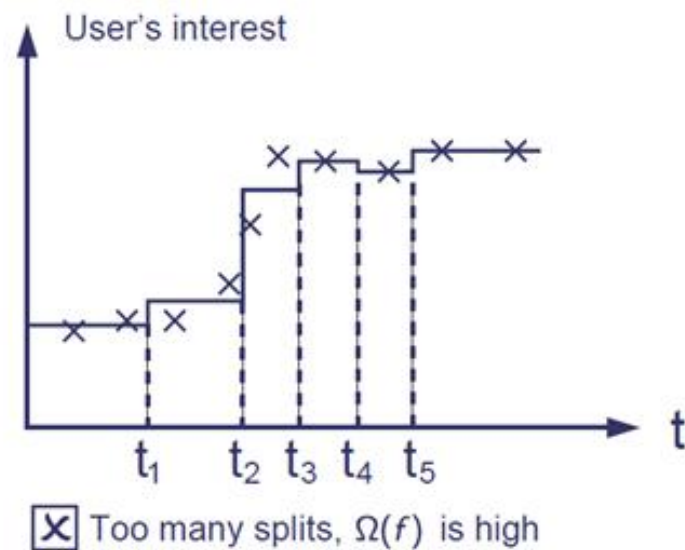
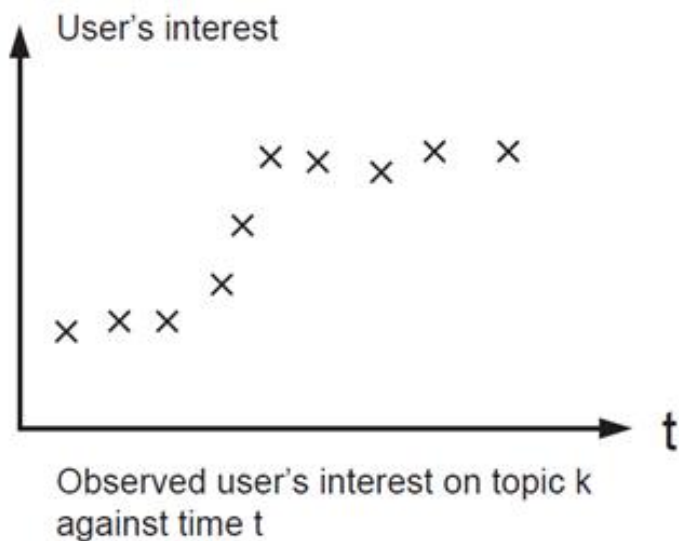
- 目标函数

$$Obj(\theta) = L(\theta) + \Omega(\theta)$$

误差函数：体现的是  
模型有多拟合数据

正则化项：惩罚复杂模型  
的参数，用于解决过拟合

# 模型



- GBDT的目标函数

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = \hat{y}_i^{(1)} + f_2(x_i)$$

.....

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

$$obj = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)})$$



# XGBoost

- XGBoost的目标函数

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = \hat{y}_i^{(1)} + f_2(x_i)$$

.....

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

$$obj = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i)$$

$$f_t(x) = w_{q(x)}$$

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

## XGBoost公式推导

- 第t次迭代后，模型的预测等于前t-1次的模型加上第t棵树的预测：

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

- 目标函数可以写成：

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

## XGBoost公式推导

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

- 将误差函数中的  $\hat{y}_i^{(t-1)} + f_t(x_i)$  看成一个整体，求解这个整体取值在  $\hat{y}_i^{(t-1)}$  处进行二阶泰勒展开：

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

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

$$h_i = \partial_{\hat{y}_i^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$$

## XGBoost公式推导

- 将函数中的所有常数项全部去掉，可以得到以下公式：

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

- 将函数和正则项带入公式中得到以下公式：

$$loss \approx \sum_{i=1}^n \left[ g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^T w_j^2$$

## XGBoost公式推导

$$loss \approx \sum_{i=1}^n \left[ g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^T w_j^2$$

- 定义每个叶子节点j上的样本集合为:  $I_j$

$$I_j = \{i \mid q(x_i) == j\}$$

- 将样本累加操作转换为叶节点的操作

$$loss \approx \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i \right) w_j^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^T w_j^2$$

$$= \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T = \sum_{j=1}^T \left[ G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T$$

## XGBoost公式推导

- 最终的目标函数:






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

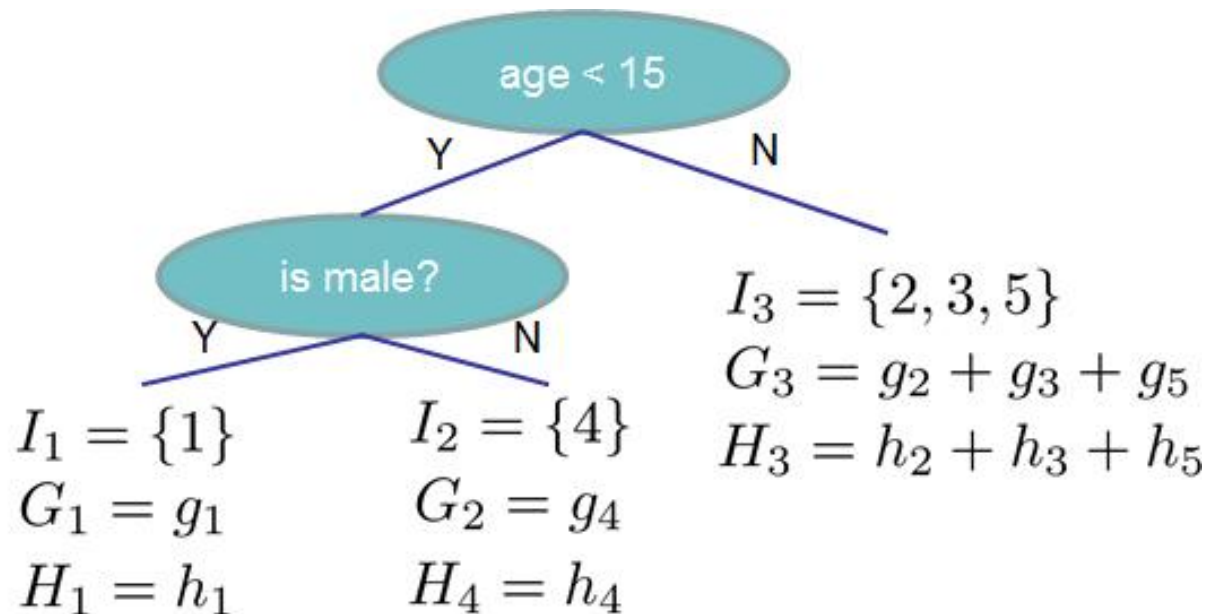
- 如果树的结构确定(q函数确定), 为了使目标函数最小, 可以令导数为0, 可以求得最优的w, 将w带入目标函数, 可以得到最终的损失为:

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

# XGBoost公式推导

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		$g_5, h_5$



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

The smaller the score is, the better the structure is

## XGBoost的学习策略

- 当树的结构确定的时候，我们可以得到最优的叶子点分数以及对应的最小损失值，问题在于如何确定树结构？
  - 暴力穷举所有可能的结构，选择损失值最小的；(很难求解)
  - 贪心法，每次尝试选择一个分裂点进行分裂，计算操作前后的增益，选择增益最大的方式进行分裂。
- 决策树相关算法计算指标：
  - ID3算法：信息增益
  - C4.5算法：信息增益率
  - CART算法：Gini系数



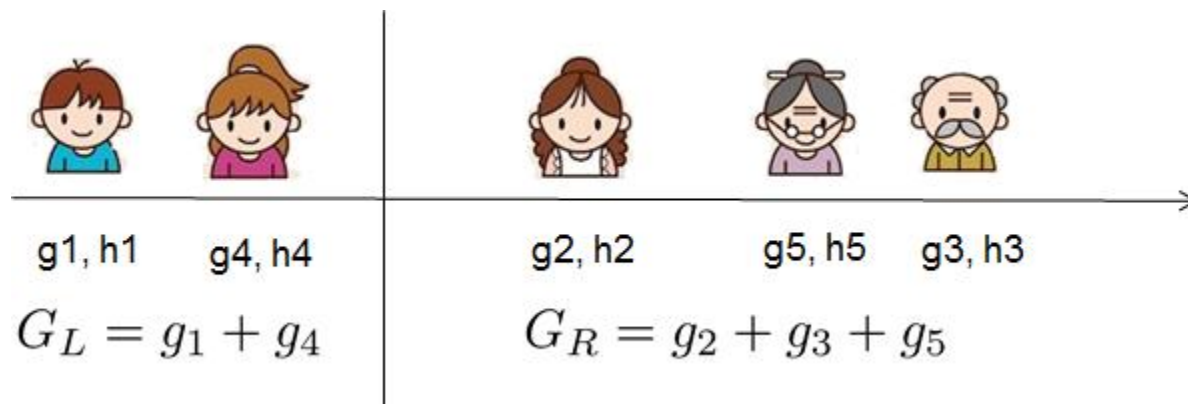
## XGBoost的学习策略

- XGBoost目标函数:  $loss^* = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$
- 从目标函数中, 我们希望损失函数越小越好, 那就是  $\frac{G^2}{H + \lambda}$  越大越好; 从而, 对于一个叶子节点的分裂的分裂, 分裂前后的信息增益定义为:

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

- Gain值越大, 分裂后减少的损失值越大。所以对于一个叶子节点分割时, 计算所有候选的(feature,value)对应的gain, 选择gain最大特征进行分割。

# XGBoost的学习策略



## 树节点分裂方法

- 精确算法：遍历所有特征的所有可能的分割点，计算gain值，选择最大的gain值对应的(feature,value)进行分割

---

### Algorithm 1: Exact Greedy Algorithm for Split Finding

---

**Input:**  $I$ , instance set of current node

**Input:**  $d$ , feature dimension

$gain \leftarrow 0$

$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$

**for**  $k = 1$  **to**  $m$  **do**

$G_L \leftarrow 0, H_L \leftarrow 0$

**for**  $j$  in sorted( $I$ , by  $\mathbf{x}_{jk}$ ) **do**

$G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$

$G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$

$score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

**end**

**end**

**Output:** Split with max score

---

## 树节点分裂方法

- 近似算法：对于每个特征，只考虑分位点，减少计算复杂度

---

### Algorithm 2: Approximate Algorithm for Split Finding

---

```
for  $k = 1$  to  $m$  do
    | Propose  $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$  by percentiles on feature  $k$ .
    | Proposal can be done per tree (global), or per split(local).
end
for  $k = 1$  to  $m$  do
    |  $G_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$ 
    |  $H_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$ 
end
```

Follow same step as in previous section to find max score only among proposed splits.

---

## 树节点分裂方法

- 近似算法案例：三分位数

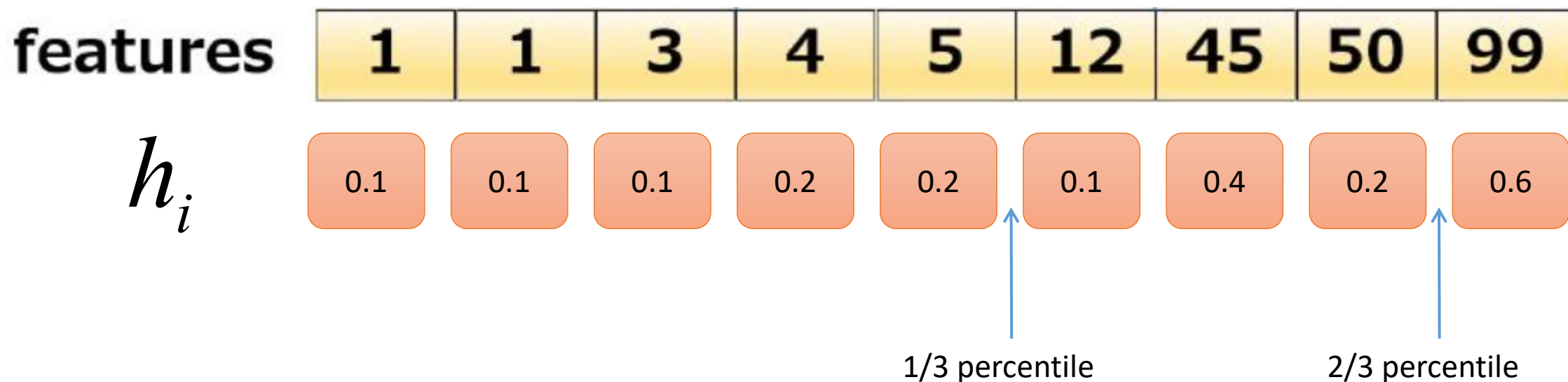
features	1	1	3	4	5	12	45	50	99
labels	1	0	0	1	1	0	0	0	0
$g_i$	$g_1$	$g_2$	$g_3$	$g_4$	$g_5$	$g_6$	$g_7$	$g_8$	$g_9$
$h_i$	$h_1$	$h_2$	$h_3$	$h_4$	$h_5$	$h_6$	$h_7$	$h_8$	$h_9$
	$G_1, H_1$			$G_2, H_2$			$G_3, H_3$		

$$Gain = \max \left\{ \frac{1}{2} \left[ \frac{G_1^2}{H_1 + \lambda} + \frac{G_{23}^2}{H_{23} + \lambda} - \frac{(G_{123})^2}{(H_{123}) + \lambda} \right] - \gamma, \right.$$

$$\left. \frac{1}{2} \left[ \frac{G_3^2}{H_3 + \lambda} + \frac{G_{12}^2}{H_{12} + \lambda} - \frac{(G_{123})^2}{(H_{123}) + \lambda} \right] - \gamma \right\}$$

## XGBoost树节点划分方法

- XGBoost不是简单的按照样本个数进行分位的，而是按照上一轮的预测误差函数的二阶导数值作为权重来进行划分的：



## XGBoost的其它特性

- 列采样(column subsampling): 借鉴随机森林的做法, 支持列抽样, 不仅可以降低过拟合, 还可以减少计算量;
- 支持对缺失值的自动处理。对于特征的值有缺失的样本, XGBoost可以自动学习分裂方向;
- XGBoost支持并行。XGBoost的并行是特征粒度上的, 在计算特征的Gain的时候, 会并行执行, 但是在树的构建过程中, 还是串行构建的;
- XGBoost算法中加入正则项, 用于控制模型的复杂度, 最终模型更加不容易过拟合;
- XGBoost基学习器支持CART、线性回归、逻辑回归;
- XGBoost支持自定义损失函数(要求损失函数二阶可导)。



# XGBoost相关参数

参数	XGBClassifier	XGBRegressor
max_depth	给定树的深度，默认为3	
learning_rate	每个迭代产生的模型的权重/学习率，默认为0.1	
n_estimators	子模型的数量，默认为100	
objective	给定损失函数，默认为"binary:logistic"	给定损失函数，默认为"reg:linear"
booster	给定模型的求解方式，默认为: gbtree； 可选参数: gbtree、gblinear、dart	
n_jobs	使用多少个线程并行构建XGBoost模型，默认为1	
reg_alpha	L1正则项的权重，默认为0	
reg_lambda	L2正则项的权重，默认为1	

- 参考链接:

- [https://xgboost.readthedocs.io/en/latest/python/python\\_api.html](https://xgboost.readthedocs.io/en/latest/python/python_api.html)
- <https://xgboost.readthedocs.io/en/latest/parameter.html>



