集成学习-提升方法

七月算法

主要内容

- 集成学习算法的思路
- AdaBoost算法
- Gradient boosting算法

集成学习算法思路

- 假如有T个朋友，每个人拥有一个预测A股的函数gi(x)，如何更好的预测股价升降
  - validation：选择一个炒股成绩最好的gi(x)
  - uniformly：平均每个人预测结果，投票的方式
  - non-uniformly: 加权求和每个人的结果，某些人的权重更大
  - conditionally: 有条件的加权求和每个的结果，满足条件下某些人的权重更大

- 存在函数 $g_1\ g_2 \cdots g_T$，每个 $g_i(x)$ 函数表示股价升降预测
  - validation：选择一个炒股成绩最好的

$$G(x)=g_{t\cdot}(x),\ \mathrm{argmin}_{t\cdot\in\{1..T\}}\mathrm{E}_{val}(g_t)$$

  - uniformly：平均每个人预测结果

$$G(x)=\mathrm{sign}\left(\sum_{i=1}^{T}1\cdot g_i(x)\right)$$

  - non-uniformly: 加权求和每个人的结果，某些人的权重更大

$$G(x)=\mathrm{sign}\left(\sum_{i=1}^{T}\alpha_i\cdot g_i(x)\right)\ with\ \alpha_i\geq 0$$

  - conditionally: 有条件的加权求和每个的结果，满足条件下某些人的权重更大

$$G(x)=\mathrm{sign}\left(\sum_{i=1}^{T}q_i(x)\cdot g_i(x)\right)\ with\ q_i(x)\geq 0$$

- Uniform blending
  - 利用函数的多样化，实现G(x)比单个g(x)好
  - 分类应用
    - $G(x)=\text{argmax}_{k=1,\ldots K}\sum_{i}^{T}[g_i(x)=k]$
  - 回归应用

$$G(x)=\frac{1}{T}\sum_{i}^{T}g_i(x)$$

有效的提高稳定性，降低variance

- Linear blending
  - 选择一组权重，满足

$$G(x)=\mathrm{sign}\left(\sum_{i=1}^{T}\alpha_i\cdot g_i(x)\right)\ \textit{with}\ \alpha_i\geq 0 \qquad \min_{\alpha_i\geq 0} Error(G(x,\alpha))$$

  - 回归问题，等价于新的特征下的线性回归，唯一的约束条件是alpha非负

linear blending for regression

$$\min_{\alpha_t\geq 0}\frac{1}{N}\sum_{n=1}^{N}\left(y_n-\sum_{t=1}^{T}\alpha_t g_t(\mathbf{x}_n)\right)^2$$

LinReg + transformation

$$\min_{w_i}\frac{1}{N}\sum_{n=1}^{N}\left(y_n-\sum_{i=1}^{\tilde{d}}w_i\phi_i(\mathbf{x}_n)\right)^2$$

  - 验证集，训练集

- 如果构造多样性的 $g_1\ g_2 \cdots g_T$
  - 基于模型
    - 不同的算法
    - 不同的参数
  - 基于数据
    - 随机采样
    - 设置权重，改变概率
    - 随机选择特征

自适应提升算法

- Adaptive Boosting (AdaBoost)
  - 对样本赋予权重，采用迭代方式构造
  - 线性加权得到最后结果

- 利用权重构造不同的函数
  - 对同样的算法，相同的训练集，如果样本的权重不一样，能够得到不同的函数

$$argmin_{g_t} \sum_{i}^{N} u_i^t [g_t(x_i) \neq y_i] \text{ ---> } g_t \quad \text{argmin}_{g_t} \sum_{i}^{N} u_i^{t-1} [g_{t-1}(x_i) \neq y_i] \text{ ---> } g_{t-1}$$

  - 反过来，已知一个函数，可以通过设置权重，使得这个函数看起来像随机的

$$\frac{\sum_{i}^{N} u_i^t \left[g_t(x_i) \neq y_i \right]}{\sum_{i}^{N} u_i^t} = \frac{1}{2}$$

- 权重－>函数，函数－>权重，因此构造出一个迭代的方式，通过这个迭代的方法可以制造出一系列的 g(x)

want: $$\frac{\sum_{n=1}^{N} u_n^{(t+1)} [\![y_n \neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^{N} u_n^{(t+1)}} = \frac{\blacksquare_{t+1}}{\blacksquare_{t+1} + \bullet_{t+1}} = \frac{1}{2},$$ where

$$\blacksquare_{t+1} = \sum_{n=1}^{N} u_n^{(t+1)} [\![y_n \neq g_t(\mathbf{x}_n)]\!], \bullet_{t+1} = \sum_{n=1}^{N} u_n^{(t+1)} [\![y_n = g_t(\mathbf{x}_n)]\!]$$

- need: $\underbrace{(\text{total } u_n^{(t+1)} \text{ of incorrect})}_{\blacksquare_{t+1}} = \underbrace{(\text{total } u_n^{(t+1)} \text{ of correct})}_{\bullet_{t+1}}$

- 即在已有的权重下，迭代构造新的权重（对应新的判别函数）
  - 把分错样本的权重放大，分对样本的权重缩小即可

'optimal' re-weighting: let $\epsilon_t = \frac{\sum_{n=1}^{N} u_n^{(t)} [\![y_n \neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^{N} u_n^{(t)}}$,

multiply incorrect $\propto (1 - \epsilon_t)$; multiply correct $\propto \epsilon_t$

define scaling factor $\blacklozenge_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$

$$\begin{aligned} \text{incorrect} &\leftarrow \text{incorrect} \cdot \blacklozenge_t \\ \text{correct} &\leftarrow \text{correct} \,/\, \blacklozenge_t \end{aligned}$$

- 基于权重构造系列函数的方法
  - u1 = [1/N, … 1/N];
  - for t = 1… T
    - 在ut权重下选择gt，使得权重分类误差最小
    - 根据gt的结果，重新生产ut+1
  - 最后得到一组gt(x)函数（实际上可以认为g0(x)为随机猜测函数）

- 得到了g1(x)… gt(x)一系列函数，那么如何构造G(x)呢?
- AdaBoost 是一个在迭代中直接构造权重alpha的算法

$$\alpha_t=\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \quad \varepsilon_t=\frac{\sum_i^N u_i^t[g_t(x_i)\neq y_i]}{\sum_i^N u_i^t}$$

$$\alpha_t = \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \quad \varepsilon_t = \frac{\sum_i^N u_i^t [g_t(x_i) \neq y_i]}{\sum_i^N u_i^t}$$

- 当加权错误率接近0的时候，a变得非常大
- 当加权错误率接近1/2的时候，a接近0

- 完整的AdaBoost算法

  (1) 初始化权重 $U^1=\left[\frac{1}{N}\dots\frac{1}{N}\right]$

  (2) 构造迭代过程，t=1...T

  * 选择最优的函数

  $$g_t=\text{argmin}_g \sum_{i=1}^{N} u_i^{t-1}[(g(x_i)\text{-}y_i)]$$

  * 计算错误率

  $$\varepsilon_t=\frac{\sum_i^N u_i^t\left[g_t\left(x_i\right)\neq y_i\right]}{\sum_i^N u_i^t}$$

  * 根据当前的结果更新权重

  * xi 正确分类: $u_i^{t+1}=u_i^t\sqrt{\frac{1\text{-}\varepsilon}{\varepsilon}}$

  * xi 错误分类: $u_i^{t+1}=u_i^t\sqrt{\frac{\varepsilon}{1\text{-}\varepsilon}}$

  * 记录权重 $\alpha_t=\frac{1}{2}\ln\left(\frac{1\text{-}\varepsilon_t}{\varepsilon_t}\right)$

- 输出判别函数 $G(x)=\text{sign}\left(\sum_t^T \alpha_t g_t(x)\right)$

- AdaBoost函数也可以看作是前向分步算法的一种实现
  - 集成模型为加法模型
  - 损失函数为指数函数

- 前向分步算法

□ 考虑加法模型

$$f(x)=\sum_{m=1}^{M}\beta_m b(x;\gamma_m)$$

□ 其中：

- 基函数：$b(x;\gamma_m)$
- 基函数的参数 γ_m
- 基函数的系数：β_m

• 前向分步算法含义

□ 在给定训练数据及损失函数L(y,f(x))的条件下，学习加法模型f(x)成为**经验风险极小化**即**损失函数极小化**问题：

$$\min_{\beta_m,\gamma_m}\sum_{i=1}^{N}L\left(y_i,\sum_{m=1}^{M}\beta_m b(x_i;\gamma_m)\right)$$

□ 算法简化：如果能够从前向后，每一步只学习一个基函数及其系数，逐步逼近上式，即：每步只优化损失函数：$\min_{\beta,\gamma}\sum_{i=1}^{N}L(y_i,\beta b(x_i;\gamma))$

- 前向分步算法的算法框架

- 输入：
  - 训练数据集T={(x1,y1), (x2,y2)…(xN,yN)}
  - 损失函数L(y,f(x))
  - 基函数集{b(x; γ)}
- 输出：
  - 加法模型f(x)
- 算法步骤：

- 初始化$f_0(x) = 0$
- 对于m=1,2,..M
  - 极小化损失函数 $(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$
    - 得到参数 β_m γ_m
  - 更新当前模型：

$$f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$$

- 得到加法模型 $f(x) = f_M(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$

- 前向分步算法与AdaBoost

- AdaBoost算法是前向分步算法的特例，这时，模型是基本分类器组成的加法模型，损失函数是指数函数。
- 损失函数取：

$$L(y, f(x)) = \exp(-yf(x))$$

- 证明

- 假设经过m-1轮迭代，前向分步算法已经得到$f_{m-1}(x)$：

$$f_{m-1}(x) = f_{m-2}(x) + \alpha_{m-1}G_{m-1}(x) = \alpha_1 G_1(x) + \cdots + \alpha_{m-1}G_{m-1}(x)$$

- 在第m轮迭代得到 α_m，$G_m(x)$和 $f_m(x)$
- 目标是使前向分步算法得到的 α_m 和$G_m(x)$使 $f_m(x)$ 在训练数据集T上的指数损失最小，即

$$(\alpha_m, G_m(x)) = \arg\min_{\alpha, G} \sum_{i=1}^{N} \exp(-y_i(f_{m-1}(x_i) + \alpha G(x_i)))$$

- 证明

- 进一步：

$$\left(\alpha_m, G_m(x)\right) = \arg\min_{\alpha, G} \sum_{i=1}^{N} \overline{w}_{mi} \exp\left(-y_i \alpha G(x_i)\right)$$

- 其中： $\overline{w}_{mi} = \exp\left(-y_i f_{m-1}(x_i)\right)$
- $\overline{w}_{mi}$ 既不依赖 α 也不依赖G，所以与最小化无关。但$\overline{w}_{mi}$依赖于$f_{m-1}(x)$，所以，每轮迭代会发生变化。

- 证明

□ 首先求分类器G*(x)

□ 对于任意$\alpha>0$，是上式最小的G(x)由下式得到：

$$G_m^*(x)=\arg\min_G\sum_{i=1}^N\overline{w}_{mi}I(y_i\neq G(x_i))$$

□ 其中， $\overline{w}_{mi}=\exp(-y_if_{m-1}(x_i))$

- 证明

□ **分类错误率为：**

$$e_m = \frac{\sum_{i=1}^{N} \overline{w}_{mi} I\left(y_i \neq G(x_i)\right)}{\sum_{i=1}^{N} \overline{w}_{mi}} = \sum_{i=1}^{N} \overline{w}_{mi} I\left(y_i \neq G(x_i)\right)$$

- 证明

- 求权值：

$$\sum_{i=1}^{N}\overline{w}_{mi}\exp(-y_i\alpha G(x_i))$$

$$=\sum_{y_i=G_m(x_i)}\overline{w}_{mi}e^{-\alpha}+\sum_{y_i\neq G_m(x_i)}\overline{w}_{mi}e^{\alpha}$$

$$=\left(e^{\alpha}-e^{-\alpha}\right)\sum_{i=1}^{N}\overline{w}_{mi}I(y_i\neq G(x_i))+e^{-\alpha}\sum_{i=1}^{N}\overline{w}_{mi}$$

- 将G*(x)带入：$G_m^*(x)=\arg\min_{G}\sum_{i=1}^{N}\overline{w}_{mi}I(y_i\neq G(x_i))$
- 求导，得到

$$\alpha_m=\frac{1}{2}\log\frac{1-e_m}{e_m}$$

- 证明

权值的更新

- 由模型

$$f_m(x) = f_{m-1}(x) + \alpha_m G_m(x)$$

- 以及权值

$$\overline{w}_{mi} = \exp(-y_i f_{m-1}(x_i))$$

- 可以方便的得到：

$$\overline{w}_{m+1,i} = \overline{w}_{m,i} \exp(-y_i \alpha_m G_m(x))$$

Theoretical Guarantee of AdaBoost

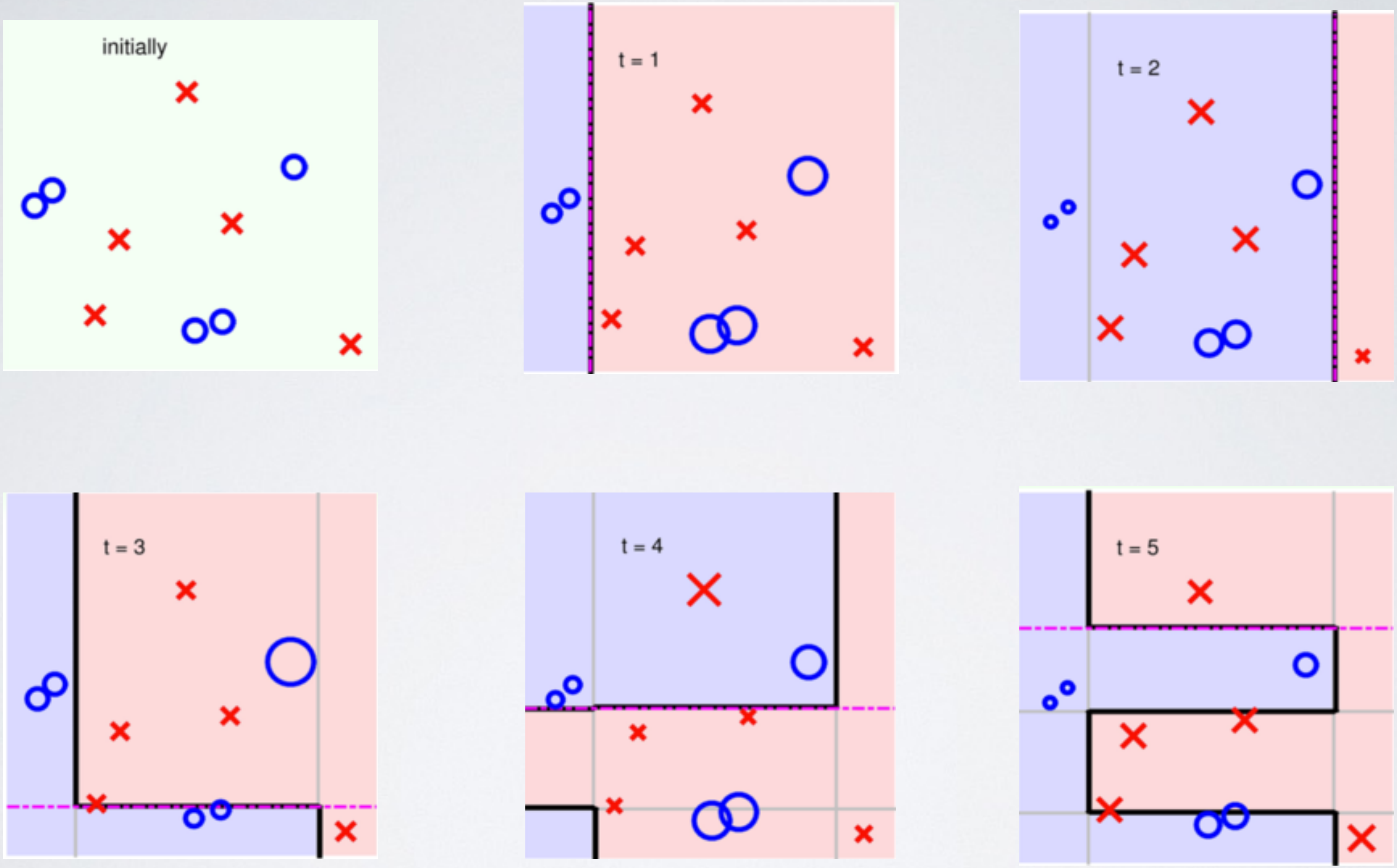
- From VC bound

$$E_{\text{out}}(G) \le E_{\text{in}}(G) + O\left(\sqrt{\underbrace{O(d_{\text{VC}}(\mathcal{H}) \cdot T \log T)}_{d_{\text{VC}} \text{ of all possible } G} \cdot \frac{\log N}{N}}\right)$$

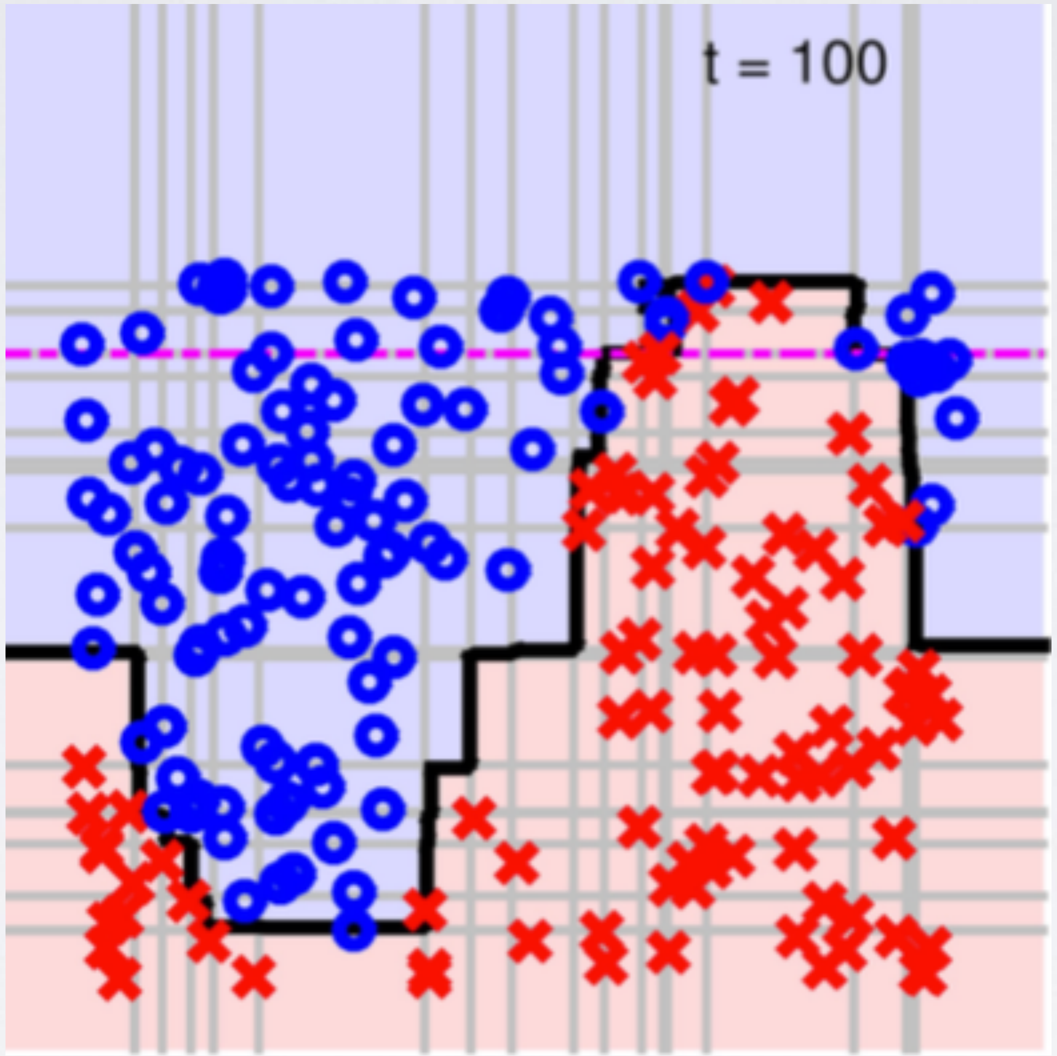
- **first term can be small**:
  $E_{\text{in}}(G) = 0$ after $T = O(\log N)$ iterations if $\epsilon_t \le \epsilon < \frac{1}{2}$ always
- **second term can be small**:
  overall d_{VC} grows "slowly" with T

boosting view of AdaBoost:
if $\mathcal{A}$ is weak but always **slightly better than random** ($\epsilon_t \le \epsilon < \frac{1}{2}$),
then (AdaBoost+$\mathcal{A}$) can be strong ($E_{\text{in}} = 0$ and E_{out} small)

- 图示



- 图示



- 著名案例

AdaBoost-Stump in Application

original picture by F.U.S.I.A. assistant and derivative work by Sylenius via Wikimedia Commons

The World's First 'Real-Time' Face Detection Program

- AdaBoost-Stump as core model: linear aggregation of key patches selected out of 162,336 possibilities in 24x24 images —**feature selection** achieved through AdaBoost-Stump
- modified linear aggregation *G* to rule out non-face earlier —**efficiency** achieved through modified linear aggregation

提升树

- GBRT (gradient boosted regression tree)
- GBDT(gradient boosted decision tree)
- LambdaMART
- etc

- 集成树方法（Tree Ensemble methods)
  - 有许多的决策树构成，继承了树的优点：
    - Invariant to scaling of inputs
    - Learn higher order interaction between features
  - 集成的方法
    - 提高整体的泛化能力，克服单一树的过拟合

- 树集成模型
  - 模型

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

Space of functions containing all Regression trees

每一个回归树，就是把样本映射到一个score，使用加性模型

  - 参数

每棵树的结构，每个非叶节点对应一个feature, 一个split value

  - 目标

学习出函数的集合，每个函数实际构成又是个单一的树

- 假设我们拥有k个树

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

- 目标函数

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

Training loss　　　　Complexity of the Trees

- 如何设计复杂度函数？损失函数？

- CART树回顾
  - 信息增益 —> 用gini指数作为损失函数
  - 树的减枝操作 —> 用节点数目作为正则函数
  - 最大深度 -> 树空间的约束
  - 回归树叶节点score L2 Norm->正则函数

- 模型可以应用到回归，分类等多种问题
  - 尽管我们使用回归树作为基函数
  - 只需要通过设计合适的目标函数，可以应用到分类问题
    - 回归问题：$l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
    - 二分类问题：$l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$

- 如何实现函数的学习?
  - 我们寻找的对象是一棵棵的回归树，不再是向量空间，因此无法使用SGD算法
  - 采用加性提升训练方法：Additive Training
    - 从最简单起点开始，一步步增加

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

New function

Model at training round t

Keep functions added in previous round

- 通过最优化目标函数，来确定选择什么样的$f_t(x)$加入到模型当中

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

This is what we need to decide in round t

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

Goal: find f_t to minimize this

- 如采用L2损失函数

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

前一次迭代的残差

- 对目标函数进行Talay展开，保留两阶

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

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

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

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

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

- 如采用L2损失函数

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

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

- 去掉常数项目，观察新的目标函数

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

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

很明显 g_i h_i 都是可以计算，已知的

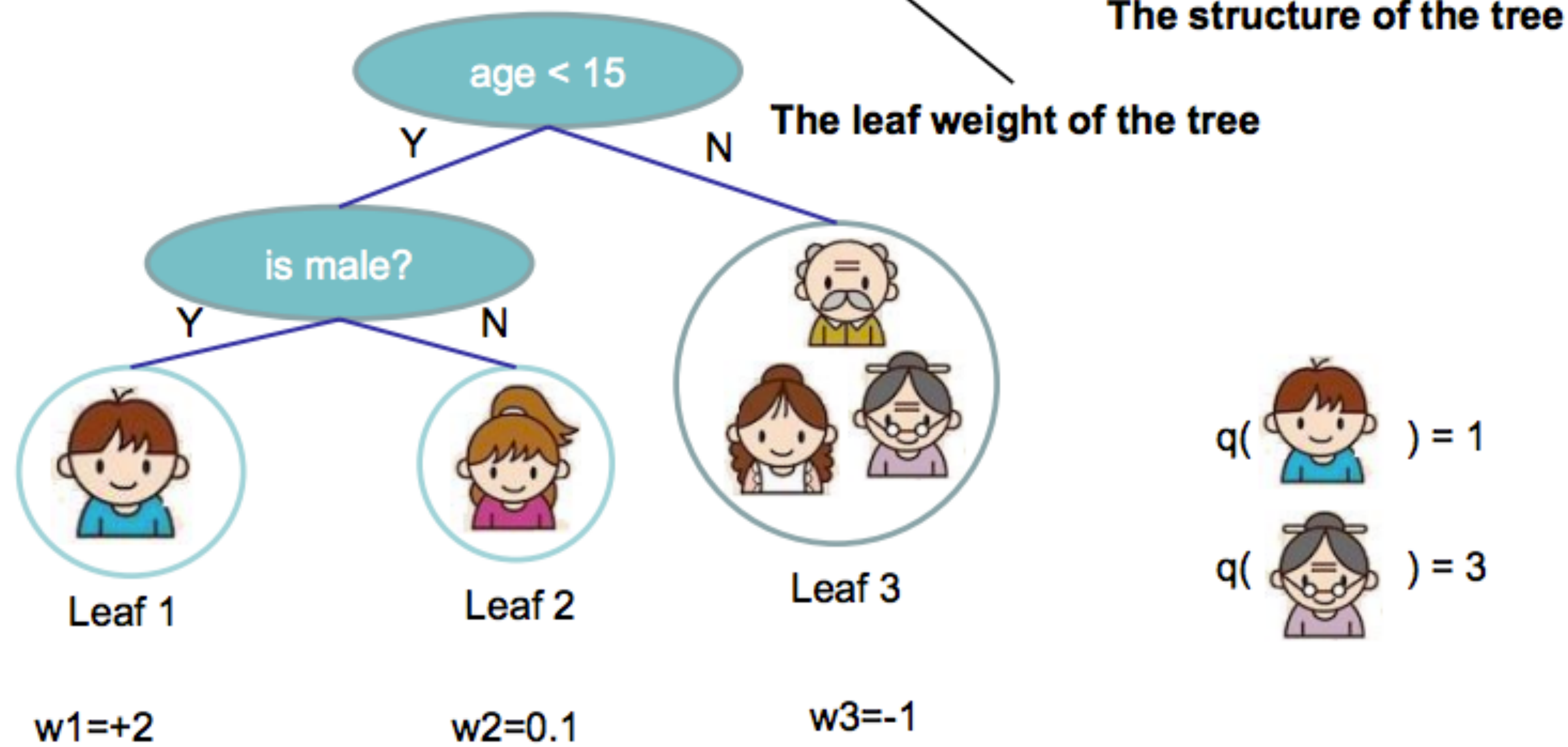
- 接下来仔细考察 $f_t(x)$，即一棵独立的回归树

- 考察 $f_t(x)$

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

The structure of the tree

The leaf weight of the tree

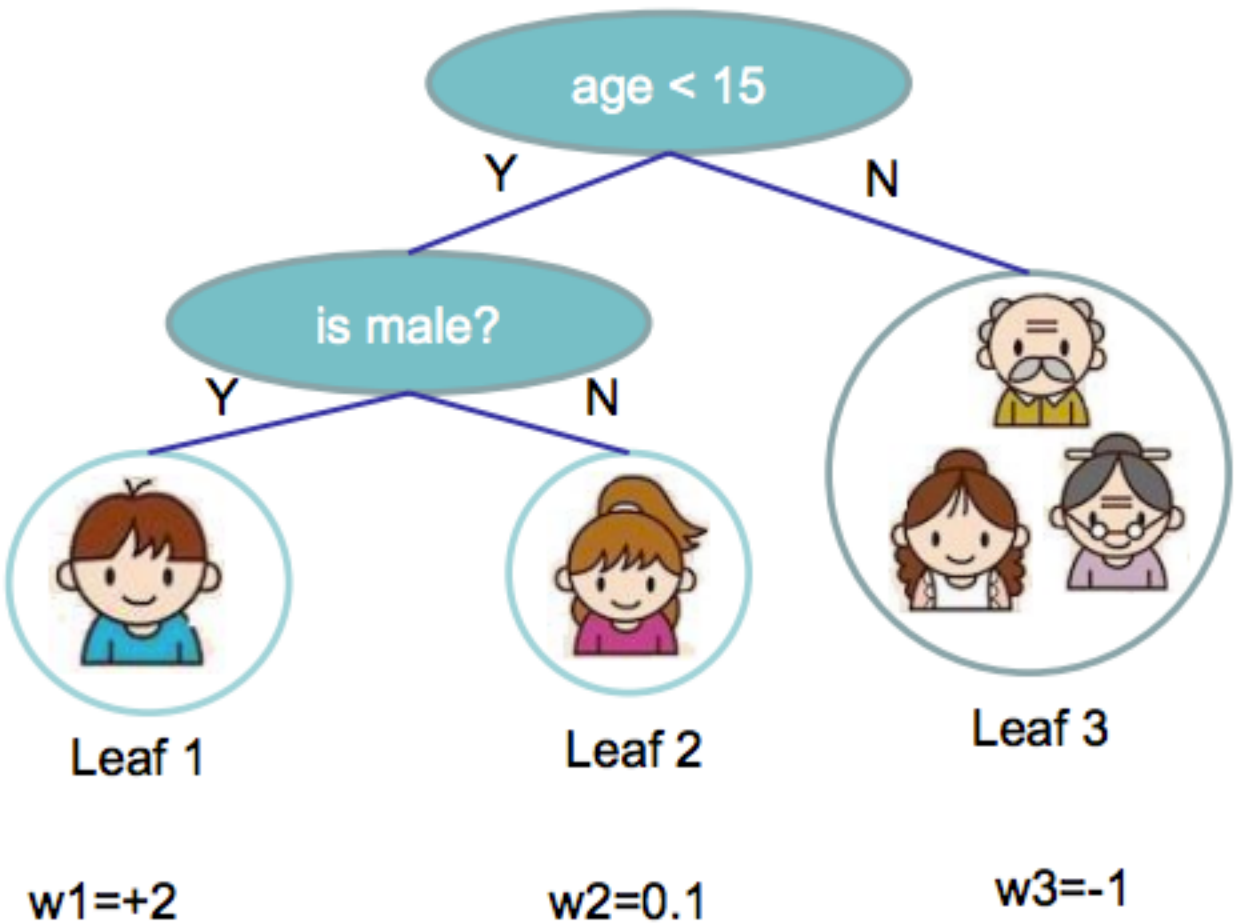


- 每个叶节点输出标量，所有叶子节点输出构成T维向量W
- q 为样本到叶节点的映射

- 指定一个复杂度函数

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

Number of leaves **L2 norm of leaf scores**



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

注意不是唯一的方法

- 对某一个$f_t(x)$，遍历所有叶节点 $I_j = \{i|q(x_i) = j\}$ 首先定义叶节点训练样本集合

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

T个变量的二次函数

- 树的评价计算

- Two facts about single variable quadratic function

$$argmin_x\ Gx + \frac{1}{2}Hx^2 = -\frac{G}{H},\ H > 0 \qquad \min_x\ Gx + \frac{1}{2}Hx^2 = -\frac{1}{2}\frac{G^2}{H}$$

- Let us define $G_j = \sum_{i\in I_j} g_i \quad H_j = \sum_{i\in I_j} h_i$

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

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

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

This measures how good a tree structure is!

- 知道如何评价树，如何搜索出最佳树?
  - 枚举所有树结构
    - 计算树的评价函数

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

  - 选择最优树，计算对应的W

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

但是，树结构太多了

- 不是搜索所有树结构，而是从根节点开始，一层层成长出一颗最佳树

- 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

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

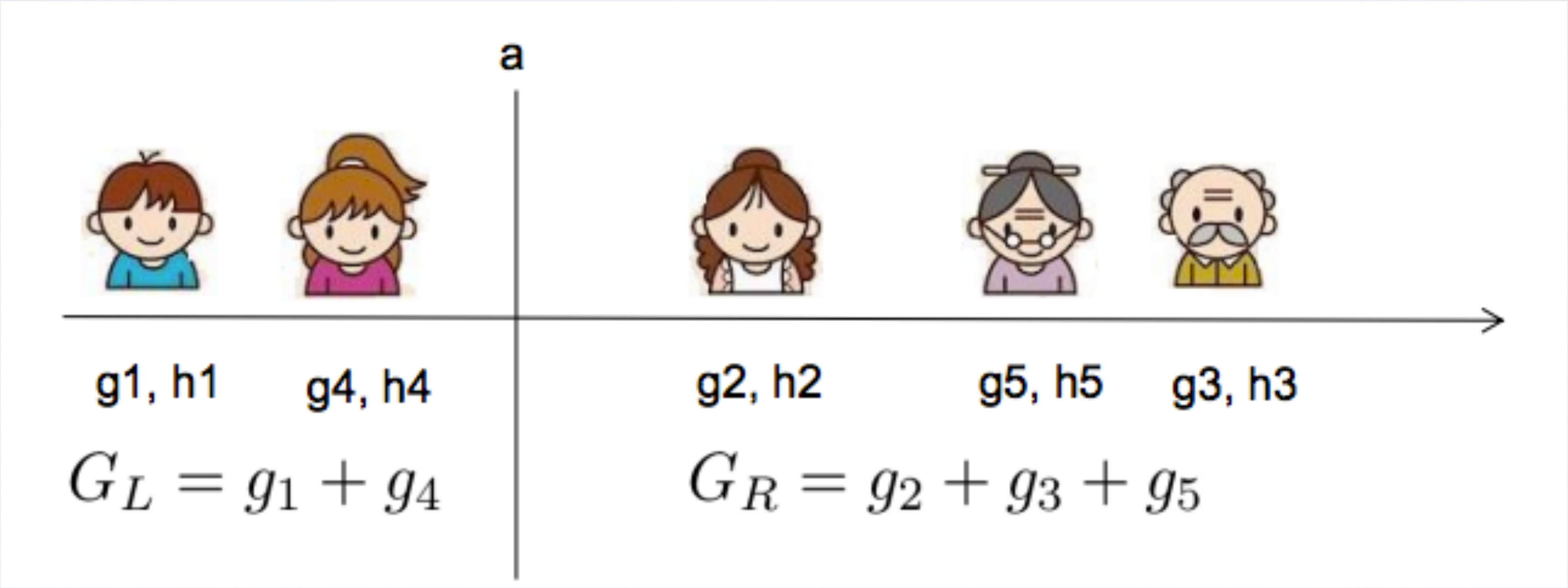
the score of left child

the score of if we do not split

the score of right child

注意不再计算gini指数

- 如何发现最佳的分支点
  - 遍历所有特征维度
    - 在选中的特征维度上，按样本值排序，选择对应的split value
    - 计算目标函数



- 完整的算法

- Add a new tree in each iteration
- Beginning of each iteration, calculate

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

- Use the statistics to greedily grow a tree $f_t(x)$

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

- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
  - Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
  - ϵ is called step-size or shrinkage, usually set around 0.1
  - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting

- 随机森林 vs AdaBoost vs Gradient boosting
    - **随机 vs 权重 vs 梯度**
- 梯度提升算法实现包，刷Kaggle数据的神器
    - https://github.com/dmlc/xgboost