

法律声明

□ 本课件包括演示文稿、示例、代码、题库、视频和声音等内容，小象学院和主讲老师拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意及内容，我们保留一切通过法律手段追究违反者的权利。

□ 课程详情请咨询

■ 微信公众号：小象

■ 新浪微博：ChinaHadoop



提升



小象学院
ChinaHadoop.cn

邹博

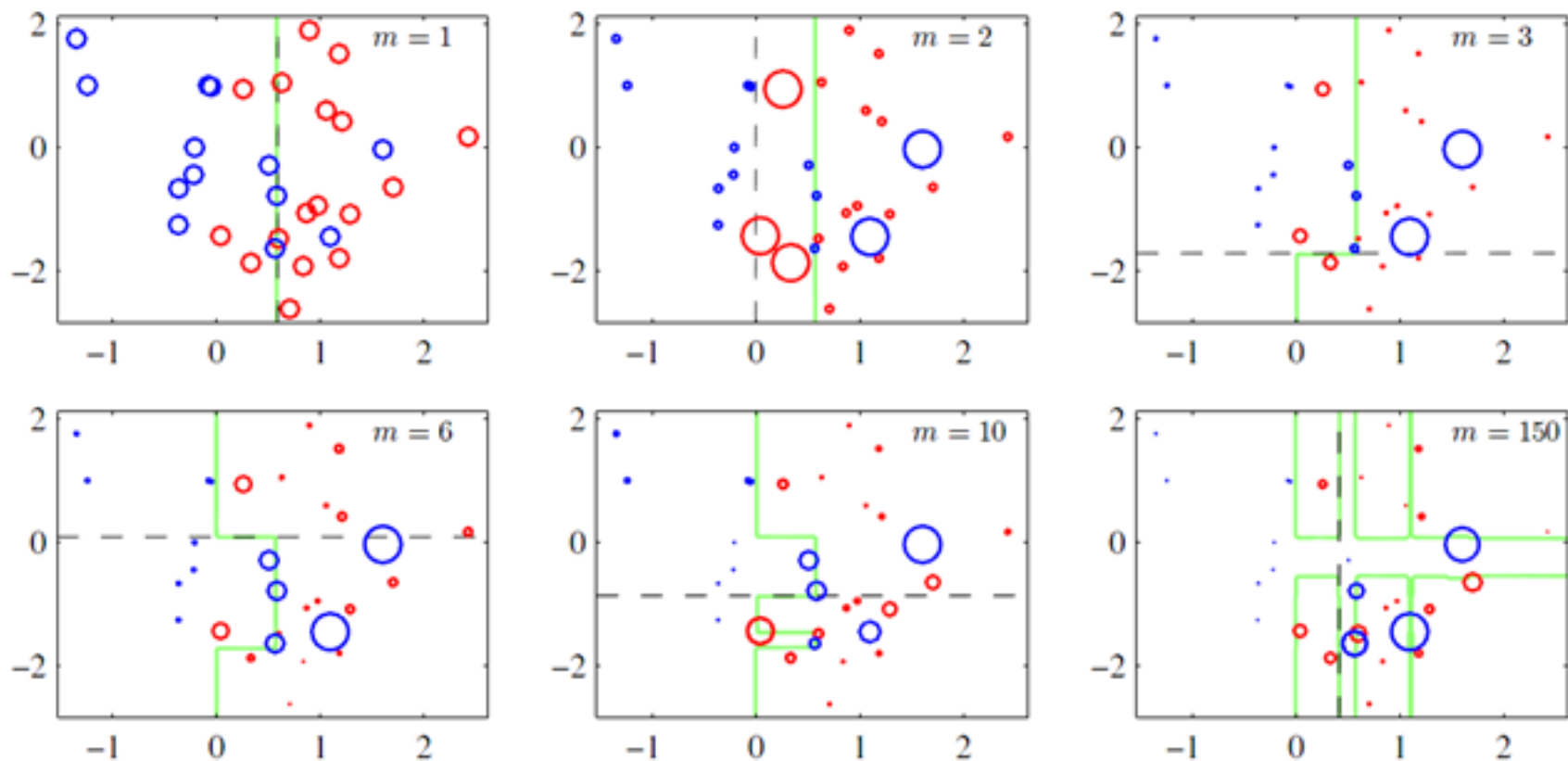
主要内容

- 分析随机森林的特点
- 由弱分类器得到强分类器
 - 样本加权、分类器加权
- Adaboost 算法
 - 算法描述
 - 前向分步算法+指数损失函数
- 由 Adaboost/GBDT 改造随机森林

由决策树和随机森林的关系的思考

- 随机森林的决策树分别采样建立，相对独立。
- 思考：
 - 假定当前一定得到了 $m-1$ 颗决策树，是否可以通过现有样本和决策树的信息，对第 m 颗决策树的建立产生有益的影响呢？
 - 各个决策树组成随机森林后，最后的投票过程可否在建立决策树时即确定呢？

样本加权



提升的概念

- 提升是一个机器学习技术，可以用于回归和分类问题，它每一步产生一个弱预测模型(如决策树)，并加权累加到总模型中；如果每一步的弱预测模型生成都是依据损失函数的梯度方向，则称之为梯度提升(Gradient boosting)。
- 梯度提升算法首先给定一个目标损失函数，它的定义域是所有可行的弱函数集合(基函数)；提升算法通过迭代的选择一个负梯度方向上的基函数来逐渐逼近局部极小值。这种在函数域的梯度提升观点对机器学习的很多领域有深刻影响。
- 提升的理论意义：如果一个问题存在弱分类器，则可以通过提升的办法得到强分类器。

提升算法

□ 给定输入向量 \mathbf{x} 和输出变量 y 组成的若干训练样本 $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$, 目标是找到近似函数 $\hat{F}(\vec{x})$, 使得损失函数 $L(y, F(\mathbf{x}))$ 的损失值最小

□ $L(y, F(\mathbf{x}))$ 的典型定义为 $L(y, F(\vec{x})) = \frac{1}{2}(y - F(\vec{x}))^2$

$$L(y, F(\vec{x})) = |y - F(\vec{x})|$$

□ 假定最优函数为 $F^*(\vec{x})$, 即:

$$F^*(\vec{x}) = \arg \min_F E_{(x,y)} [L(y, F(\vec{x}))]$$

□ 假定 $F(\mathbf{x})$ 是一族基函数 $f_i(\mathbf{x})$ 的加权和

$$F(\vec{x}) = \sum_{i=1}^M \gamma_i f_i(x) + const$$

提升算法推导

- 梯度提升方法寻找最优解 $F(x)$ ，使得损失函数在训练集上的期望最小。方法如下：
- 首先，给定常函数 $F_0(x)$ ：

$$F_0(\vec{x}) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$$

- 以贪心的思路扩展得到 $F_m(x)$ ：

$$F_m(\vec{x}) = F_{m-1}(\vec{x}) + \arg \min_{f \in H} \sum_{i=1}^n L(y_i, F_{m-1}(\vec{x}_i) + f(\vec{x}_i))$$

梯度近似 $F_m(\vec{x}) = F_{m-1}(\vec{x}) + \arg \min_{f \in H} \sum_{i=1}^n L(y_i, F_{m-1}(\vec{x}_i) + f(\vec{x}_i))$

□ 贪心法在每次选择最优基函数 f 时仍然困难

■ 使用梯度下降的方法近似计算

■ 将样本带入基函数 f 得到 $f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_n)$, 从而 L 退化为向量 $L(y_1, f(\mathbf{x}_1)), L(y_2, f(\mathbf{x}_2)), \dots, L(y_n, f(\mathbf{x}_n))$

$$F_m(\vec{x}) = F_{m-1}(\vec{x}) - \gamma_m \sum_{i=1}^n \nabla_f L(y_i, F_{m-1}(\vec{x}_i))$$

□ 上式中的权值 γ 为梯度下降的步长, 使用线性搜索求最优步长:

$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, F_{m-1}(\vec{x}_i) - \gamma \cdot \nabla_f L(y_i, F_{m-1}(\vec{x}_i)))$$

提升算法

□ 初始给定模型为常数 $F_0(\vec{x}) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$

□ 对于 $m=1$ 到 M

■ 计算伪残差 $r_{im} = \left[\frac{\partial L(y_i, F(\vec{x}_i))}{\partial F(\vec{x}_i)} \right]_{F(\vec{x})=F_{m-1}(\vec{x})} \quad i = 1, 2, \dots, n$

□ pseudo residuals

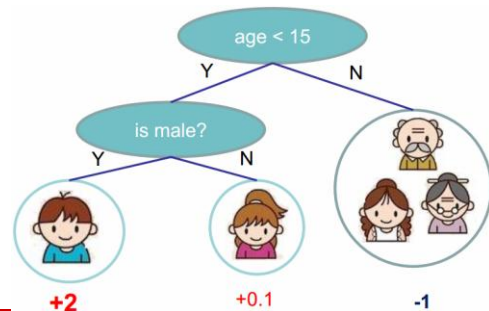
■ 使用数据 $\{(\vec{x}_i, r_{im})\}_{i=1}^n$ 计算拟合残差的基函数 $f_m(x)$

■ 计算步长 $\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, F_{m-1}(\vec{x}_i) - \gamma \cdot f_m(\vec{x}_i))$

□ 一维优化问题

■ 更新模型 $F_m(\vec{x}) = F_{m-1}(\vec{x}) - \gamma_m f_m(\vec{x}_i)$

梯度提升决策树GBDT



- 梯度提升的典型基函数即**决策树**(尤其是**CART**)
- 在第 m 步的梯度提升是根据伪残差数据**计算决策树** $t_m(\mathbf{x})$ 。令树 $t_m(\mathbf{x})$ 的**叶节点**数目为 J ，即树 $t_m(\mathbf{x})$ 将输入空间划分为 J 个**不相交区域** $R_{1m}, R_{2m}, \dots, R_{Jm}$ ，并且决策树 $t_m(\mathbf{x})$ 可以在每个区域中给出某个类型的确定性预测。使用指示记号 $I(\mathbf{x})$ ，对于输入 \mathbf{x} ， $t_m(\mathbf{x})$ 为：

$$t_m(\vec{x}) = \sum_{j=1}^J b_{jm} I(\vec{x} \in R_{jm})$$

- 其中， b_{jm} 是样本 \mathbf{x} 在区域 R_{jm} 的预测值。

$$\text{GBDT } t_m(\vec{x}) = \sum_{j=1}^J b_{jm} I(\vec{x} \in R_{jm})$$

□ 使用线性搜索计算学习率，最小化损失函数

$$F_m(\vec{x}) = F_{m-1}(\vec{x}) + \gamma_m \cdot t_m(\vec{x}_i)$$

$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, F_{m-1}(\vec{x}_i) + \gamma \cdot t_m(\vec{x}_i))$$

□ 进一步：对树的每个区域分别计算步长，从而系数 b_{jm} 被合并到步长中，从而：

$$F_m(\vec{x}) = F_{m-1}(\vec{x}) + \sum_{j=1}^J \gamma_{jm} I(\vec{x} \in R_{jm})$$

$$\gamma_{jm} = \arg \min_{\gamma} \sum_{\vec{x}_i \in R_{jm}} L(y_i, F_{m-1}(\vec{x}_i) + \gamma \cdot t_m(\vec{x}_i))$$

参数设置和正则化

- 对训练集拟合过高会降低模型的泛化能力，需要使用**正则化**技术来降低过拟合。
 - 对复杂模型增加惩罚项，如：模型复杂度正比于叶结点数目或者叶结点预测值的平方和等。
 - 用于决策树剪枝
- 叶结点数目控制了树的层数，一般选择 $4 \leq J \leq 8$ 。
- 叶结点包含的最少样本数目
 - 防止出现过小的叶结点，降低预测方差
- 梯度提升迭代次数M：
 - 增加M可降低训练集的损失值，但有过拟合风险
 - 交叉验证

衰减因子、降采样

- 衰减Shrinkage $F_m(\vec{x}) = F_{m-1}(\vec{x}) + \nu \cdot \gamma_m f_m(\vec{x}_i)$, $0 < \nu \leq 1$
 - 称 ν 为学习率
 - $\nu=1$ 即为原始模型；推荐选择 $\nu < 0.1$ 的小学习率。过小的学习率会造成计算次数增多。
- 随机梯度提升Stochastic gradient boosting
 - 每次迭代都对伪残差样本采用无放回的降采样，用部分样本训练基函数的参数。令训练样本数占有伪残差样本的比例为 f ； $f=1$ 即为原始模型；推荐 $0.5 \leq f \leq 0.8$ 。
 - 较小的 f 能够增强随机性，防止过拟合，并且收敛的快。
 - 降采样的额外好处是能够使用剩余样本做模型验证。

$$F^*(\vec{x}) = \arg \min_F E_{(x,y)} [L(y, F(\vec{x}))]$$

GBDT总结

$$F(\vec{x}) = \sum_{i=1}^M \gamma_i f_i(x) + const$$

- ❑ 函数估计本来被认为是在函数空间而非参数空间的数值优化问题，而阶段性的加性扩展和梯度下降手段将函数估计转换成参数估计。
- ❑ 损失函数是最小平方误差、绝对值误差等，则为回归问题；而误差函数换成多类别Logistic似然函数，则成为分类问题。
- ❑ 对目标函数分解成若干基函数的加权和，是常见的技术手段：神经网络、径向基函数、傅立叶/小波变换、SVM都可以看到它的影子。
- ❑ 思考：如果对基函数的学习中，不止考虑函数的参数和权值，而是对样本本身也加权，会得到什么结果呢？

考虑使用二阶导信息

□ 目标函数: $J(f_t) = \sum_{i=1}^n L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + C$

□ 根据Taylor展式:

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

□ 令,

$$g_i \stackrel{\text{def}}{=} \frac{\partial L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}, \quad h_i \stackrel{\text{def}}{=} \frac{\partial^2 L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)^2}}$$

□ 得:

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

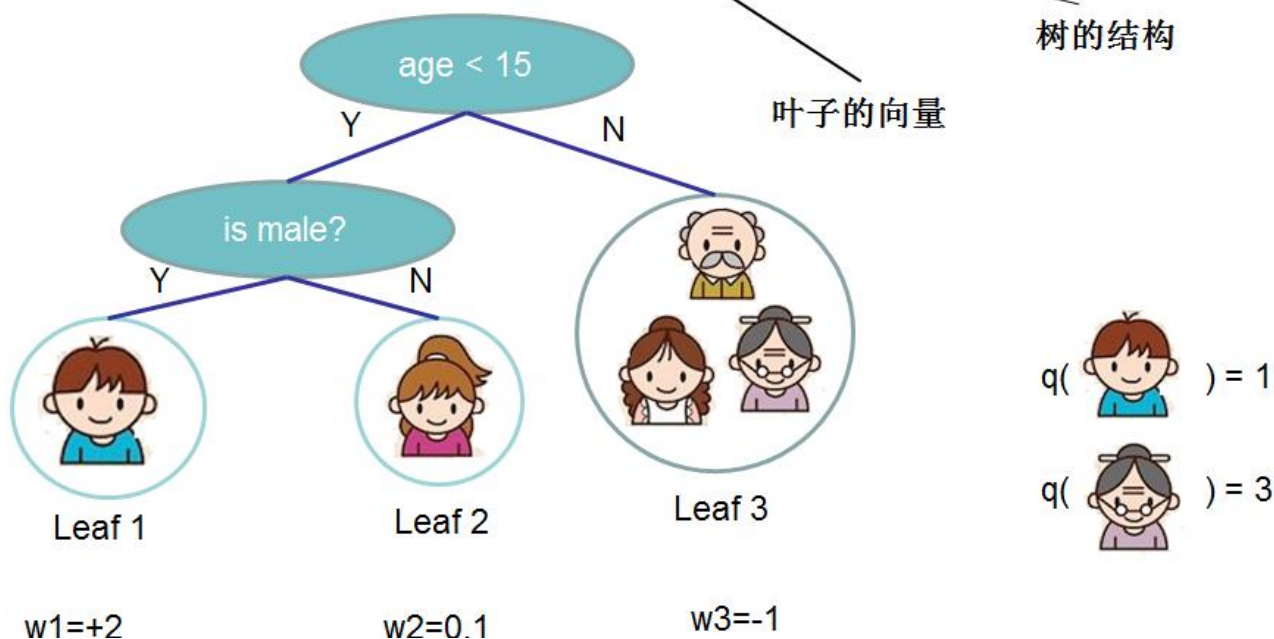
决策树的描述

- 使用决策树对样本做分类(回归), 是从根结点到叶节点的细化过程; 落在相同叶节点的样本的预测值是相同的。
- 假定某决策树的叶结点数目为 T , 每个叶结点的权值为 $\vec{w}=(w_1, w_2 \cdots w_T)$, 决策树的学习过程, 就是构造如何使用特征得到划分, 从而得到这些权值的过程。
- 样本 x 落在叶结点 q 中, 定义 f 为: $f_t(x) = w_{q(x)}$
 - 一个决策树的核心即“树结构”和“叶权值”。

举例

□ 是否喜欢计算机游戏的例子：

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



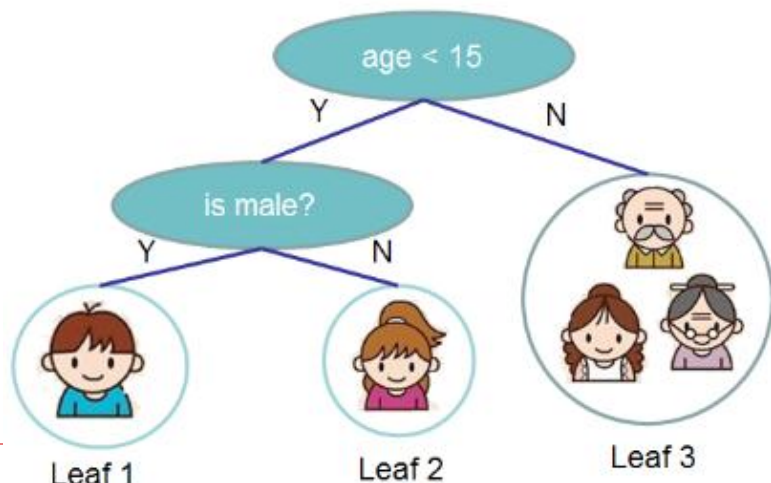
正则项的定义

□ 决策树的复杂度可考虑叶结点数和叶权值，如使用叶结点总数和叶权值平方和的加权。

■ 不是唯一的定义方式

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

γ 叶子的个数 λ w的L2模平方



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

目标函数计算

$$\begin{aligned} J(f_t) &\approx \sum_{i=1}^n \left[L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + C \\ &= \sum_{i=1}^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + C \\ &= \sum_{i=1}^n \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma \cdot T + \lambda \cdot \frac{1}{2} \sum_{j=1}^T w_j^2 + C \\ &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i \right) w_j^2 \right] + \gamma \cdot T + \lambda \cdot \frac{1}{2} \sum_{j=1}^T w_j^2 + C \\ &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma \cdot T + C \end{aligned}$$

目标函数继续化简

□ 对于：
$$J(f_t) = \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma \cdot T + C$$

□ 定义 $G_j \stackrel{\text{def}}{=} \sum_{i \in I_j} g_i$, $H_j \stackrel{\text{def}}{=} \sum_{i \in I_j} h_i$






□ 从而，
$$J(f_t) = \sum_{j=1}^T \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma \cdot T + C$$

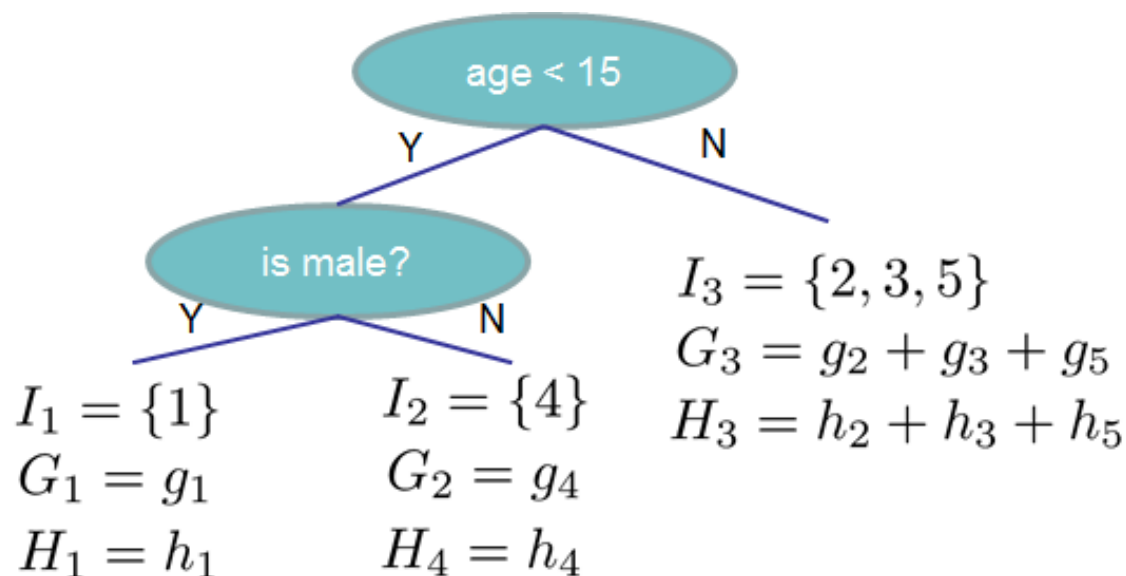
□ 对 w 求偏导，得

$$\frac{\partial J(f_t)}{\partial w_j} = G_j + (H_j + \lambda) w_j \stackrel{\text{令}}{=} 0 \Rightarrow w_j = -\frac{G_j}{H_j + \lambda}$$

□ 回代入目标函数，得
$$J(f_t) = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma \cdot T$$

举例：刚才的公式在做啥？

样本号		梯度数据
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



$$Obj = -\frac{1}{2} \sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

这个分数越小，代表这个树的结构越好

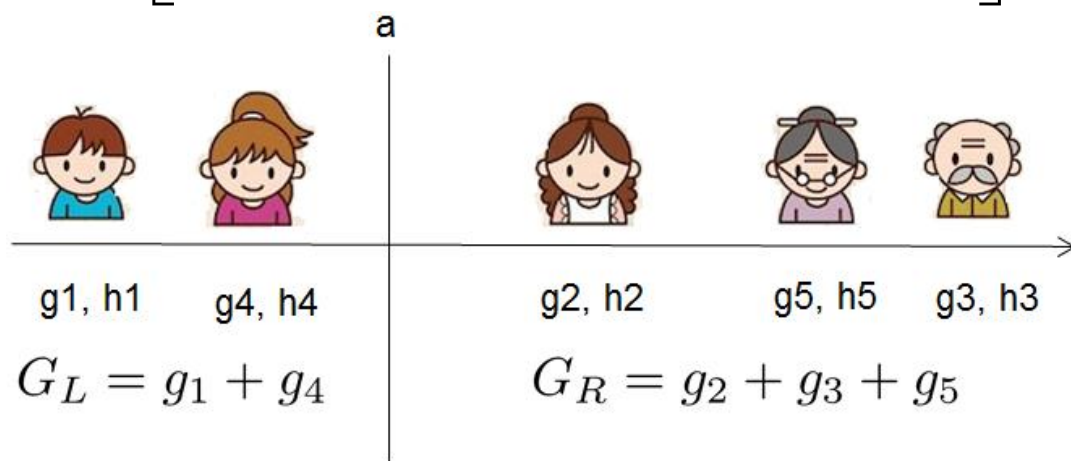
构造决策树的结构 $J(f_t) = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma \cdot T$

- 问：对于当前结点，应该如何进行子树划分？
- 借鉴ID3/C4.5/CART的做法，使用贪心法：
 - 对于某可行划分，计算划分后的 $J(f)$ ；
 - 对于所有可行划分，选择 $J(f)$ 降低最小的分割点。

构造决策树的结构 $J(f_t) = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma \cdot T$

□ 枚举可行的分割点，选择增益最大的划分，继续同样的操作，直到满足某阈值或得到纯结点。

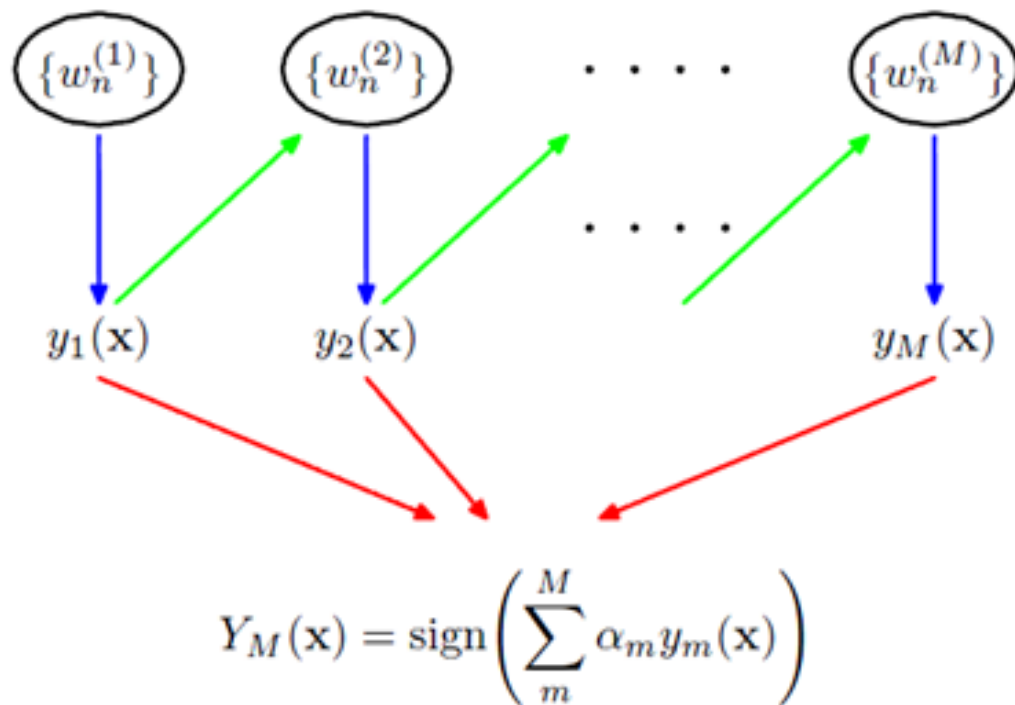
$$Gain(\phi) = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$



XGBoost小结

- 以上即为XGBoost使用的核心推导过程。
- 相对于传统的GBDT，XGBoost使用了二阶信息，可以更快的在训练集上收敛。
 - 由于“随机森林族”本身具备过拟合的优势，因此XGBoost仍然一定程度的具有该特性。
 - XGBoost的实现中使用了并行/多核计算，因此训练速度快；同时它的原生语言为C/C++，这是它速度快的实践原因。

boosting的思想



Adaboost

- 设训练数据集 $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$
- 初始化训练数据的权值分布

$$D_1 = (w_{11}, w_{12}, \dots, w_{1i}, \dots, w_{1N}), \quad w_{1i} = \frac{1}{N}, \quad i = 1, 2, \dots, N$$

Adaboost: 对于 $m=1,2,\dots,M$

- 使用具有权值分布 D_m 的训练数据集学习，得到基本分类器

$$G_m(x): \mathcal{X} \rightarrow \{-1, +1\}$$

- 计算 $G_m(x)$ 在训练数据集上的分类误差率

$$e_m = P(G_m(x_i) \neq y_i) = \sum_{i=1}^N w_{mi} I(G_m(x_i) \neq y_i)$$

- 计算 $G_m(x)$ 的系数

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

Adaboost: 对于 $m=1,2,\dots,M$

□ 更新训练数据集的权值分布

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2} \cdots w_{m+1,i} \cdots, w_{m+1,N}),$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, \dots, N$$

□ 这里, Z_m 是规范化因子

$$Z_m = \sum_{i=1}^N w_{mi} \exp(-\alpha_m y_i G_m(x_i))$$

■ 它的目的仅仅是使 D_{m+1} 成为一个概率分布

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)) \Rightarrow Z_m w_{m+1,i} = w_{mi} \exp(-\alpha_m y_i G_m(x_i)) \Rightarrow Z_1 w_{2,i} = w_{1i} \exp(-\alpha_1 y_i G_1(x_i))$$

Adaboost

□ 构建基本分类器的线性组合

$$f(x) = \sum_{m=1}^M \alpha_m G_m(x)$$

□ 得到最终分类器

$$G(x) = \text{sign}(f(x)) = \text{sign}\left(\sum_{m=1}^M \alpha_m G_m(x)\right)$$

举例

□ 给定下列训练样本，试用AdaBoost算法学习一个强分类器。

序号	1	2	3	4	5	6	7	8	9	X
X	0	1	2	3	4	5	6	7	8	9
Y	1	1	1	-1	-1	-1	1	1	1	-1

解

□ 初始化训练数据的权值分布

$$D_1 = (w_{11}, w_{12} \cdots w_{1i} \cdots, w_{1N}), w_{1i} = \frac{1}{N}, i = 1, 2, \cdots, N$$

□ $w_{1i} = 0.1$

m=1	序号	1	2	3	4	5	6	7	8	9	X
	X	0	1	2	3	4	5	6	7	8	9
	Y	1	1	1	-1	-1	-1	1	1	1	-1

□ 对于m=1

□ 在权值分布为D1的训练数据上，阈值v取2.5时误差率最低，故基本分类器为：

$$G_1(x) = \begin{cases} 1, & x < 2.5 \\ -1, & x > 2.5 \end{cases}$$

m=1

□ $G_1(x)$ 在训练数据集上的误差率 $e_1 = P(G_1(x_i) \neq y_i) = 0.3$

□ 计算 G_1 的系数:

$$\alpha_1 = \frac{1}{2} \log \frac{1-e_1}{e_1} = 0.4236$$

□ $f_1(x) = 0.4236 * G_1(x)$

□ 分类器 $\text{sign}(f_1(x))$ 在训练数据集上有3个误分类点。

m=1

□ 更新训练数据的权值分布：

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2} \cdots w_{m+1,i} \cdots, w_{m+1,N}),$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, \dots, N$$

□ $D_2 = (0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.1666, 0.1666, 0.1666, 0.0715)$

■ 计算 D_2 ，是为下一个基本分类器使用

□ $f_1(x) = 0.4236 * G_1(x)$

□ 分类器 $\text{sign}(f_1(x))$ 在训练数据集上有3个误分类点。

m=2	X	0	1	2	3	4	5	6	7	8	9
	Y	1	1	1	-1	-1	-1	1	1	1	-1
	w	0.0715	0.0715	0.0715	0.0715	0.0715	0.0715	0.1666	0.1666	0.1666	0.0715

□ 对于m=2

□ 在权值分布为D2的训练数据上，阈值v取8.5时误差率最低，故基本分类器为：

$$G_2(x) = \begin{cases} 1, & x < 8.5 \\ -1, & x > 8.5 \end{cases}$$

m=2

□ G2(x)在训练数据集上的误差率

$$e_2 = P(G_2(x_i) \neq y_i) = 0.2143(0.0715 * 3)$$

□ 计算G2的系数:

$$\alpha_2 = \frac{1}{2} \log \frac{1 - e_2}{e_2} = 0.6496$$

m=2

□ 更新训练数据的权值分布：

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2} \cdots w_{m+1,i} \cdots, w_{m+1,N}),$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, \dots, N$$

□ $D_3 = (0.0455, 0.0455, 0.0455, 0.1667, 0.1667, 0.01667, 0.1060, 0.1060, 0.1060, 0.0455)$

□ $f_2(x) = 0.4236G_1(x) + 0.6496G_2(x)$

□ 分类器 $\text{sign}(f_2(x))$ 在训练数据集上有3个误分类点。

m=3	X	0	1	2	3	4	5	6	7	8	9
	Y	1	1	1	-1	-1	-1	1	1	1	-1
	w	0.0455	0.0455	0.0455	0.1667	0.1667	0.1667	0.1060	0.1060	0.1060	0.0455

□ 对于m=3

□ 在权值分布为D3的训练数据上，阈值v取5.5时误差率最低，故基本分类器为：

$$G_3(x) = \begin{cases} 1, & x > 5.5 \\ -1, & x < 5.5 \end{cases}$$

m=3

□ G3(x)在训练数据集上的误差率

$$e_3 = P(G_3(x_i) \neq y_i) = 0.1820(0.0455 * 4)$$

□ 计算G3的系数:

$$\alpha_3 = \frac{1}{2} \log \frac{1 - e_3}{e_3} = 0.7514$$

m=3

□ 更新训练数据的权值分布：

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2} \cdots w_{m+1,i} \cdots, w_{m+1,N}),$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, \dots, N$$

□ $D_4 = (0.125, 0.125, 0.125, 0.102, 0.102, 0.102, 0.065, 0.065, 0.065, 0.125)$

□ $f_3(x) = 0.4236G_1(x) + 0.6496G_2(x) + 0.7514G_3(x)$

□ 分类器 $\text{sign}(f_3(x))$ 在训练数据集上有0个误分类点。

Adaboost误差上限 $\frac{1}{N} \sum_{i=1}^N I(G(x_i) \neq y_i) \leq \frac{1}{N} \sum_i \exp(-y_i f(x_i)) = \prod_m Z_m$

□ $G(x_i) \neq y_i$ 时, $y_i * f(x_i) < 0$, $\exp(-y_i f(x_i)) \geq 1$ 前半部分得证。

□ 后半部分:

$$\begin{aligned} \frac{1}{N} \sum_i \exp(-y_i f(x_i)) &= \sum_i \frac{1}{N} \exp\left(-\sum_{m=1}^M \alpha_m y_i G_m(x_i)\right) \\ &= \sum_i w_{1i} \exp\left(-\sum_{m=1}^M \alpha_m y_i G_m(x_i)\right) = \sum_i w_{1i} \prod_{m=1}^M \exp(-\alpha_m y_i G_m(x_i)) \\ &= \sum_i w_{1i} \exp(-\alpha_1 y_i G_1(x_i)) \prod_{m=2}^M \exp(-\alpha_m y_i G_m(x_i)) \\ &= \sum_i Z_1 w_{2i} \prod_{m=2}^M \exp(-\alpha_m y_i G_m(x_i)) = Z_1 \sum_i w_{2i} \prod_{m=2}^M \exp(-\alpha_m y_i G_m(x_i)) \\ &= Z_1 Z_2 \sum_i w_{3i} \prod_{m=3}^M \exp(-\alpha_m y_i G_m(x_i)) \\ &= Z_1 Z_2 \cdots Z_{M-1} \sum_i w_{Mi} \exp(-\alpha_M y_i G_M(x_i)) \\ &= \prod_{m=1}^M Z_m \end{aligned}$$

后半部分

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i))$$

$$\Rightarrow Z_m w_{m+1,i} = w_{mi} \exp(-\alpha_m y_i G_m(x_i))$$

$$\Rightarrow Z_1 w_{2,i} = w_{1i} \exp(-\alpha_1 y_i G_1(x_i))$$

$$\begin{aligned} \frac{1}{N} \sum_i \exp(-y_i f(x_i)) &= \sum_i \frac{1}{N} \exp\left(-\sum_{m=1}^M \alpha_m y_i G_m(x_i)\right) \\ &= \sum_i w_{1i} \exp\left(-\sum_{m=1}^M \alpha_m y_i G_m(x_i)\right) = \sum_i w_{1i} \prod_{m=1}^M \exp(-\alpha_m y_i G_m(x_i)) \\ &= \sum_i w_{1i} \exp(-\alpha_1 y_i G_1(x_i)) \prod_{m=2}^M \exp(-\alpha_m y_i G_m(x_i)) \\ &= \sum_i Z_1 w_{2i} \prod_{m=2}^M \exp(-\alpha_m y_i G_m(x_i)) = \sum_i Z_1 w_{2i} \prod_{m=2}^M \exp(-\alpha_m y_i G_m(x_i)) \\ &= Z_1 Z_2 \sum_i w_{3i} \prod_{m=3}^M \exp(-\alpha_m y_i G_m(x_i)) \\ &= Z_1 Z_2 \cdots Z_{M-1} \sum_i w_{Mi} \exp(-\alpha_M y_i G_M(x_i)) \\ &= \prod_{m=1}^M Z_m \end{aligned}$$

训练误差界

$$\prod_{m=1}^M Z_m = \prod_{m=1}^M (2\sqrt{e_m(1-e_m)}) = \prod_{m=1}^M \sqrt{1-4\gamma_m^2} \leq \exp\left(-2\sum_{m=1}^M \gamma_m^2\right) \quad \text{其中, } \gamma_m = \frac{1}{2} - e_m$$

$$Z_m = \sum_{i=1}^N w_{mi} \exp(-\alpha_m y_i G_m(x_i))$$

$$= \sum_{y_i = G_m(x_i)} w_{mi} e^{-\alpha_m} + \sum_{y_i \neq G_m(x_i)} w_{mi} e^{\alpha_m}$$

$$= (1 - e_m) e^{-\alpha_m} + e_m e^{\alpha_m}$$

$$= 2\sqrt{e_m(1-e_m)}$$

$$= \sqrt{1-4\gamma_m^2}$$

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

$$\gamma_m = \frac{1}{2} - e_m$$

取 $\gamma_1, \gamma_2 \dots$ 的最小值, 记做 γ

$$\frac{1}{N} \sum_{i=1}^N I(G(x_i) \neq y_i) \leq \exp(-2M\gamma^2)$$

附：Adaboost算法解释

- AdaBoost算法是模型为加法模型、损失函数为指数函数、学习算法为前向分步算法时的二类学习方法。

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

前向分步算法的含义

- 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 $T = \{(x_1, y_1), (x_2, y_2) \dots (x_N, y_N)\}$
- 损失函数 $L(y, f(x))$
- 基函数集 $\{b(x; \gamma)\}$

□ 输出：

- 加法模型 $f(x)$

□ 算法步骤：

前向分步算法的算法框架

□ 初始化 $f_0(x) = 0$

□ 对于 $m=1, 2, \dots, M$

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

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

□ 其中： $\bar{w}_{mi} = \exp(-y_i f_{m-1}(x_i))$

□ \bar{w}_{mi} 既不依赖 α 也不依赖 G ，所以与最小化无关。但 \bar{w}_{mi} 依赖于 $f_{m-1}(x)$ ，所以，每轮迭代会发生变化。

基本分类器 $G^*(x)$

□ 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:

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

□ 其中, $\bar{w}_{mi} = \exp(-y_i f_{m-1}(x_i))$

权值的计算

□ 求权值：

$$\begin{aligned} & \sum_{i=1}^N \bar{w}_{mi} \exp(-y_i \alpha G(x_i)) \\ &= \sum_{y_i = G_m(x_i)} \bar{w}_{mi} e^{-\alpha} + \sum_{y_i \neq G_m(x_i)} \bar{w}_{mi} e^{\alpha} \\ &= (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^N \bar{w}_{mi} I(y_i \neq G(x_i)) + e^{-\alpha} \sum_{i=1}^N \bar{w}_{mi} \end{aligned}$$

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

□ 求导，得到

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

分类错误率

□ 分类错误率为：

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

权值的更新

□ 由模型

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

□ 以及权值

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

□ 可以方便的得到：

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

权值和错误率的关键解释

- 事实上，根据Adaboost的构造过程，权值调整公式为：

$$w_{m+1,i} = \begin{cases} \frac{w_{mi}}{Z_m} e^{-\alpha_m} & G_m(x_i) = y_i \\ \frac{w_{mi}}{Z_m} e^{\alpha_m} & G_m(x_i) \neq y_i \end{cases}$$

- 二者做除，得到 $e^{2\alpha_m} = \frac{e_m}{1-e_m}$

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

AdaBoost总结

- AdaBoost算法可以看做是采用指数损失函数的提升方法，其每个基函数的学习算法为前向分步算法；
- AdaBoost的训练误差是以指数速率下降的；
- AdaBoost算法不需要事先知道下界 γ ，具有自适应性(Adaptive)，它能自适应弱分类器的训练误差率。

Code

```
max_level = 5          # 每棵树最大深度
granularity = 10       # 枚举粒度
rf = True              # 是否使用随机森林
alpha = 0.67          # 采样百分数
tree_number = 50       # 建立多少棵树
tree_weight = [0] * tree_number # 每棵树的权值
data = []              # 样本数据
```

```
if __name__ == "__main__":
    fileData = file("data.txt")
    for line in fileData:
        d = map(float, line.split(' '))
        data.append(d)
    fileData.close()

    forest = [] # 森林
    sample_number = len(data) # 样本数目
    # 初始化样本权值
    sample_weight = [1.0/float(sample_number)] * sample_number
    for i in range(tree_number):
        # 根据样本权值生成决策树tree
        tree = decision_tree(sample_weight)
        # 计算tree的错误率, 同时更新样本权值
        tree_weight[i] = predict_tree2(tree, sample_weight)
        # 将tree添加到森林中
        forest.append(tree)
    show_data(forest)
```

```
class TreeNode:
    def __init__(self):
        self.sample = [] # 该结点拥有哪些样本
        self.weight = [] # 每个样本的权值
        self.feature = -1 # 用几号特征划分
        self.value = 0 # 该特征的取值
        self.type = 0 # 该结点的类型
        self.left = -1 # 该结点的左孩子
        self.right = -1 # 该结点的右孩子
        self.gini = 0 # 该结点的gini系数
```

Code

```
def decision_tree(sample_weight):
    m = len(data)
    tree = []
    root = TreeNode()
    if rf:
        root.sample = random_select(alpha) # 随机选择样本
    else:
        root.sample = [x for x in range(m)] # 选择所有样本
    root.assign_weight(sample_weight)
    root.gini_coefficient()
    tree.append(root)
    first = 0
    last = 1
    for level in range(max_level):
        for node in range(first, last):
            tree[node].split(tree)
            first = last
            last = len(tree)
            print level+1, len(tree)
    return tree
```

Code

```
def predict_tree(d, tree):
    node = tree[0]
    while node.left != -1 and node.right != -1:
        if d[node.feature] < node.value:
            node = tree[node.left]
        else:
            node = tree[node.right]
    return node.type

def predict_tree2(tree, sample_weight):
    err = 0.0
    m = len(data)
    predict_y = [1] * m    # 0表示错误, 1表示正确
    for i in range(m):
        if data[i][-1] != predict_tree(data[i], tree):
            err += sample_weight[i]
            predict_y[i] = 0
    if err == 0.0:
        tree_w = 100
    else:
        tree_w = math.log((1-err)/err) / 2    # tree的系数
    a = math.exp(tree_w)
    for i in range(m):
        if predict_y[i] == 0:
            sample_weight[i] *= a
        else:
            sample_weight[i] /= a
    normalize(sample_weight)
    return tree_w
```

Gini系数

将样本分配到直方柱中

```
for i in range(m):
    d = data[self.sample[i]][-1] * self.weight[i]
    j = int((d - m1) * n / (m2 - m1))
    if j < 0:
        j = 0
    elif j >= n:
        j = n-1
    hist[j] += 1
hist.sort()
```

将频数转换成累积概率

```
for j in range(1,n):
    hist[j] += hist[j-1]
s = 0
for j in range(n):
    s += hist[j]
```

```
self.gini = 1 - (2*float(s)/float(m) - 1) / float(n)
```

```
def gini_coefficient(self):
    m = len(self.sample)
    if m == 0: # 该结点尚未分配样本
        self.gini = 0
        self.type = 0
        return
    n = 10 # 分成10等份
    hist = [0] * n # 每个直方柱的样本个数
    # 求最值
    m1 = m2 = data[self.sample[0]][-1] * self.weight[0]
    data_s = 0 # 该结点包含样本的和
    for i in range(m):
        d = data[self.sample[i]][-1] * self.weight[i]
        data_s += d
        if m2 < d:
            m2 = d
        elif m1 > d:
            m1 = d
    # 计算该结点的类型
    if data_s > 0:
        self.type = 1
    else:
        self.type = -1
    if m2 - m1 < 0.0001: # 最大值等于最小值, 则数据完全相等
        self.gini = 1
    return
```

Code

```
def split(self, tree):
    f = self.select_feature()
    self.choose_value(f, tree)

def select_feature(self):    # 返回当前结点最合适的特征编号
    n = len(data[0])
    gini_f = 0              # gini指数最小是0
    f = -1                  # 当前尚未选择合适特征
    for i in range(n-1):
        g = self.gini_feature(i)
        if gini_f < g:
            gini_f = g
            f = i
    return f

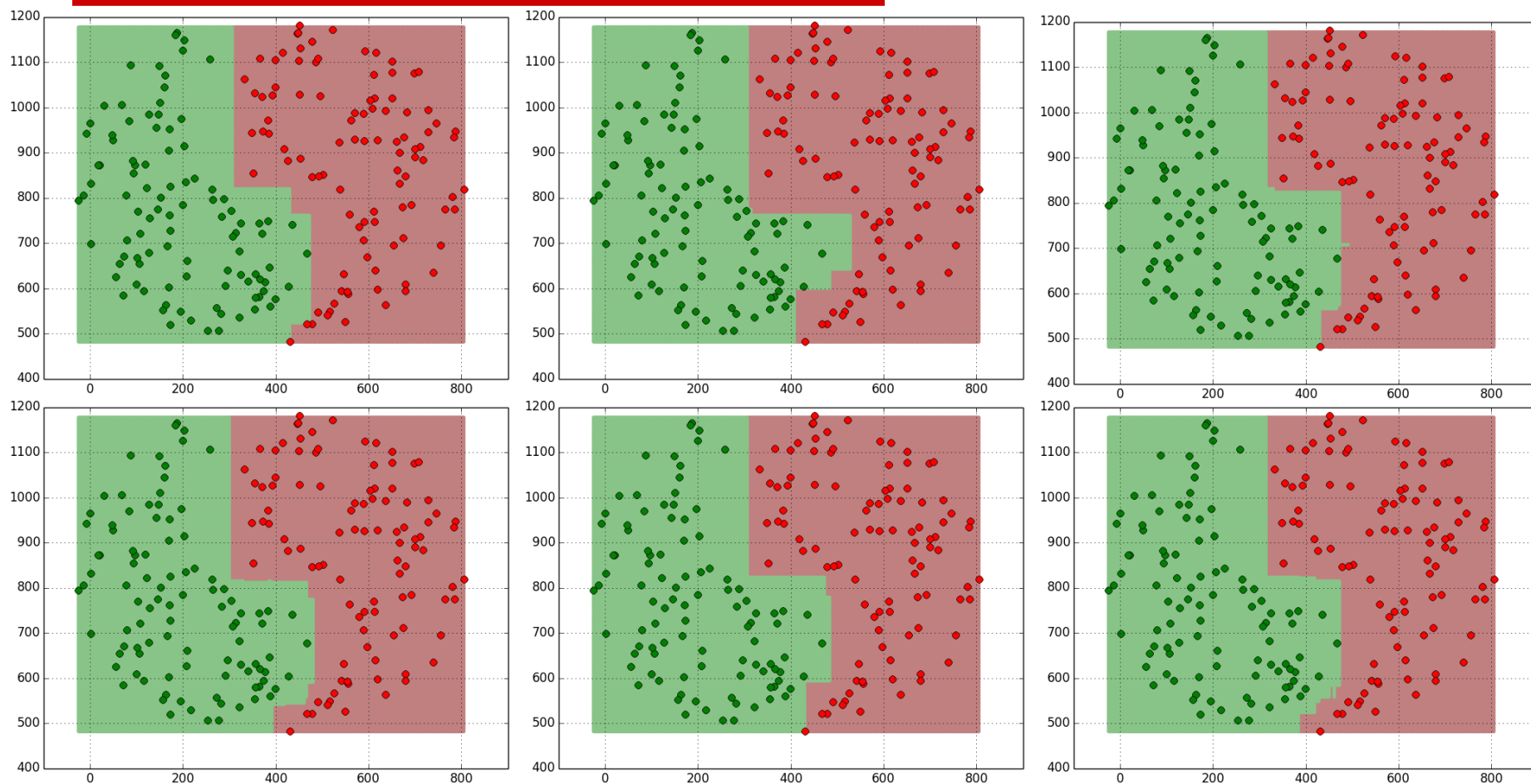
def gini_feature(self, f):  # 若用第f号特征做分割, 计算gini指数
    if len(self.sample) == 0:
        return 0
    average = self.calc_average(f)
    return self.gini_coefficient2(f, average)
```


Code

```
def choose_value(self, f, tree):
    f_max = self.calc_max(f)
    f_min = self.calc_min(f)
    step = (f_max - f_min) / granularity
    if step == 0:
        return f_min
    x_split = 0
    g_split = 0
    for x in numpy.arange(f_min+step, f_max, step):
        # if rf:      # 理论上, 随机选择优于均匀采样
        #     x = random.uniform(f_min, f_max)
        g = self.gini_coefficient2(f, x)
        if g_split < g:
            g_split = g
            x_split = x
    if g_split > self.gini: # 分割后gini系数要变大才有意义
        self.value = x_split
        self.feature = f
        t = TreeNode()
        t.sample, t.weight = self.choose_sample(f, x_split, True)
        t.gini_coefficient()
        self.left = len(tree)
        tree.append(t)
        t = TreeNode()
        t.sample, t.weight = self.choose_sample(f, x_split, False)
        t.gini_coefficient()
        self.right = len(tree)
        tree.append(t)
```

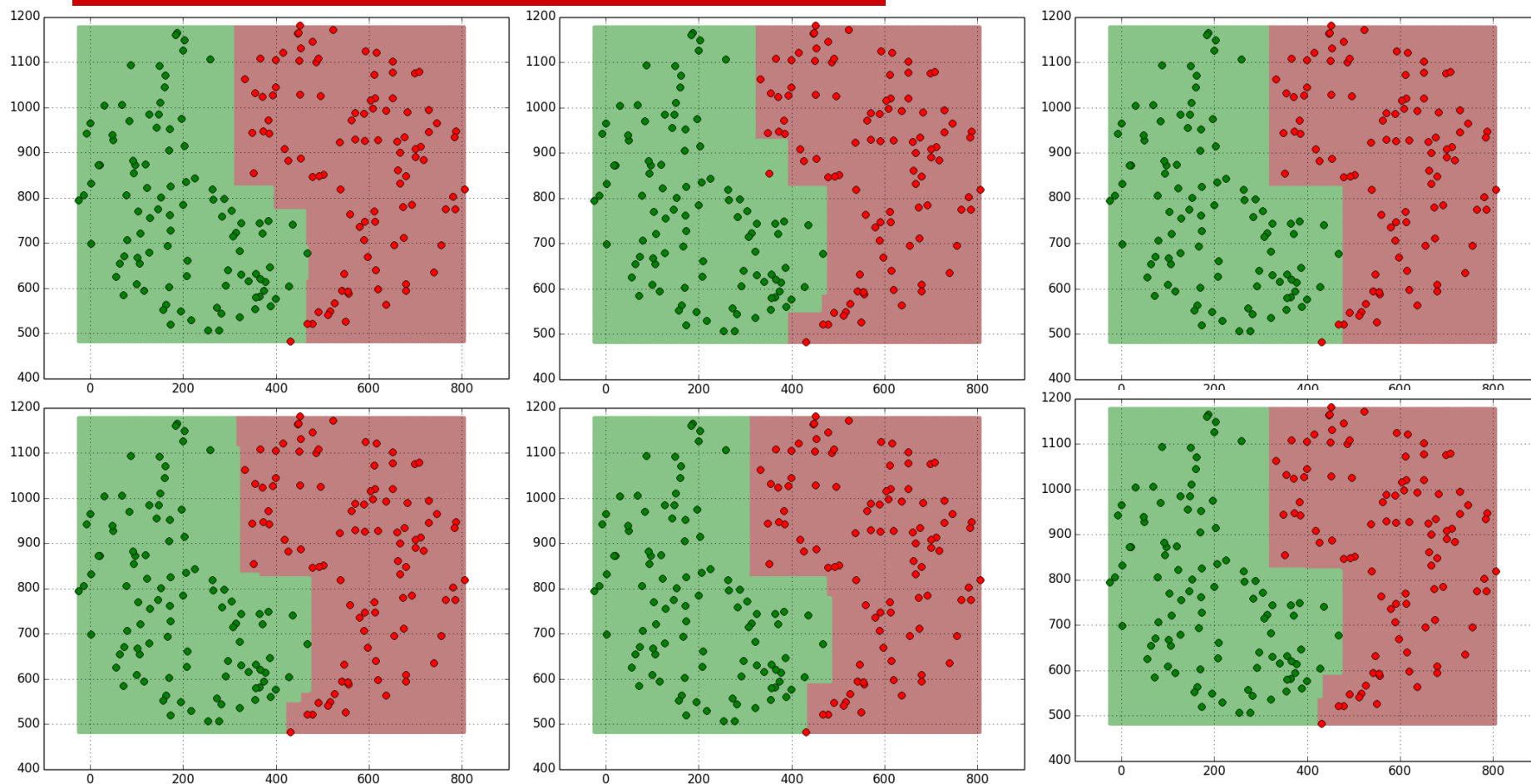
Adaboost: 5, 树的数目

10	20	30
40	50	60



Adaboost: 50, 树的层数

2	3	4
5	6	7



复习：偏差与方差

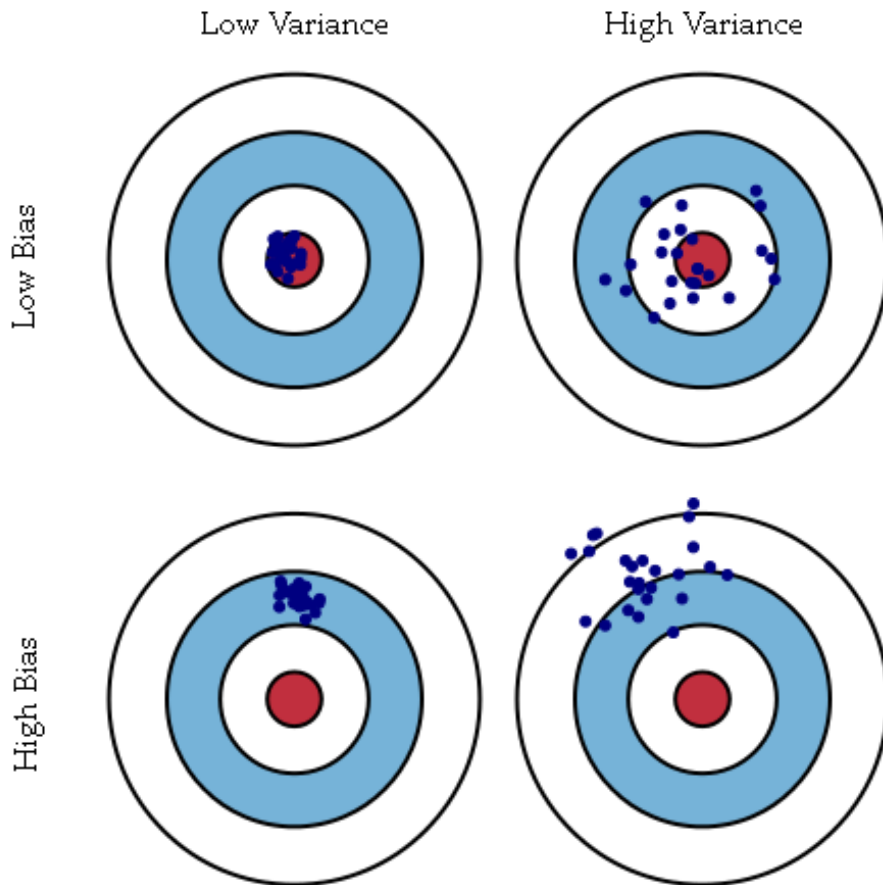
□ 给定数据 D ，自变量 x 的相应真实值为 $y(x)$ ，预测值为 $h_{\theta}(x, D)$ ，使用平方误差作为目标函数： $E_D[y(x) - h_{\theta}(x, D)]^2$

□ 得，

$$\begin{aligned} & E_D[y(x) - h_{\theta}(x, D)]^2 \\ &= E_D[y(x) - E_D[y(x)] + E_D[y(x)] - h_{\theta}(x, D)]^2 \\ &= E_D[\{y(x) - E_D[y(x)]\}^2 + \{E_D[y(x)] - h_{\theta}(x, D)\}^2 + 2\{y(x) - E_D[y(x)]\}\{E_D[y(x)] - h_{\theta}(x, D)\}] \\ &= E_D[\{y(x) - E_D[y(x)]\}^2 + \{E_D[y(x)] - h_{\theta}(x, D)\}^2] \quad // E\{y(x) - E_D[y(x)]\} = 0 \\ &= \underbrace{E_D[\{y(x) - E_D[y(x)]\}^2]}_{\text{方差Var}} + \underbrace{E_D[\{E_D[y(x)] - h_{\theta}(x, D)\}^2]}_{\text{偏差Bias}^2} \end{aligned}$$

Bias-variance dilemma

- 圆心为完美预测的模型，蓝色点代表某个模型的学习结果。离靶心越远，准确率越低。
- 低Bias表示离圆心近，高Bias表示离圆心远；
- 高Variance表示学习结果分散，低Variance表示学习结果集中。



总结

□ 方差与偏差

- Bagging能够减少训练方差(Variance), 对于不剪枝的决策树、神经网络等学习器有良好的集成效果;
- Boosting减少偏差(Bias), 能够基于泛化能力较弱的学习器构造强学习器。

□ 除了GBDT中使用关于分类器的一阶导数进行学习之外, 也可以借鉴(逆)牛顿法的思路, 使用二阶导数学习弱分类器。

作业

- GBDT和随机森林的区别是什么？
- 作为最受关注的Boosting方法Adaboost，它的优势是什么？能否从损失函数的角度谈谈Adaboost的工作原理？

参考文献

- Jerome H. Friedman. *Greedy Function Approximation: A Gradient Boosting Machine*. February 1999
- Jerome H. Friedman. *Stochastic Gradient Boosting*. March 1999
- 李航, 统计学习方法, 清华大学出版社, 2012
- https://en.wikipedia.org/wiki/Gradient_boosting#Gradient_tree_boosting

我们在这里

□ <http://wenda.ChinaHadoop.cn>

■ 视频/课程/社区

□ 微博

■ @ChinaHadoop

■ @邹博_机器学习

□ 微信公众号

■ 小象

■ 大数据分析挖掘



感谢大家！

恳请大家批评指正！

附：中位数是绝对最小最优解的证明

□ 给定样本 $x_1, x_2 \cdots x_n$ ，计算 $\mu^* = \arg \min \sum_{i=1}^n |x_i - \mu|$

□ 解：为方便推导，由于样本顺序无关，不妨假定样本 $x_1, x_2 \cdots x_n$ 是递增排序的，则：

$$J(\mu) = \sum_{i=1}^n |x_i - \mu| = \sum_{i=1}^k (\mu - x_i) + \sum_{i=k+1}^n (x_i - \mu)$$

□ 求偏导： $\frac{\partial J(\mu)}{\partial \mu} = \sum_{i=1}^k (1) + \sum_{i=k+1}^n (-1) \xrightarrow{\text{定义}} 0$

□ 从而，前 k 个样本数目与后 $n-k$ 个样本数目相同，即 μ 为中位数。