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

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/GDBT改造随机森林

复习：线性回归的梯度方向

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\begin{aligned}
\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\frac{1}{2}(h_\theta(x) - y)^2 \\
&= 2\cdot\frac{1}{2}(h_\theta(x) - y)\cdot\frac{\partial}{\partial\theta_j}(h_\theta(x) - y) \\
&= (h_\theta(x) - y)\cdot\frac{\partial}{\partial\theta_j}\left(\sum_{i=0}^{n}\theta_i x_i - y\right) \\
&= (h_\theta(x) - y)\,x_j
\end{aligned}$$

复习：Logistic回归的梯度方向

$$
\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\
&= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x) \frac{\partial}{\partial \theta_j} \theta^T x \\
&= \left(y(1 - g(\theta^T x)) - (1 - y) g(\theta^T x) \right) x_j \\
&= (y - h_\theta(x)) \, x_j
\end{aligned}
$$

复习：梯度下降的自适应学习率

- x_k=a，沿着负梯度方向，移动到x_{k+1}=b，有：

$$b = a - \alpha \nabla F(a) \Rightarrow f(a) \geq f(b)$$

- 从x0为出发点，每次沿着当前函数梯度反方向移动一定距离αk，得到序列：

$$x_0, x_1, x_2, \cdots, x_n$$

- 对应的各点函数值序列之间的关系为：

$$f(x_0) \geq f(x_1) \geq f(x_2) \geq \cdots \geq f(x_n)$$

- 当n达到一定值时，函数f(x)收敛到局部最小值

学习率α的计算标准

- 因为梯度下降是寻找f(x)的最小值，那么，在$\mathbf{x_k}$和$\mathbf{d_k}$给定的前提下，即寻找函数$f(\mathbf{x_k}+\alpha\mathbf{d_k})$的最小值。即：

$$\alpha = \arg\min_{\alpha>0} h(\alpha) = \arg\min_{\alpha>0} f(x_k + \alpha d_k)$$

- 进一步，如果h(α)可导，局部最小值处的α满足：

$$h'(\alpha) = \nabla f(x_k + \alpha d_k)^T d_k = 0$$

线性搜索求学习率

- 二分线性搜索
  - 不断将区间[α1, α2]分成两半，选择端点异号的一侧，知道区间足够小或者找到当前最优学习率。
- Armijo准则 $f(x_k+\alpha d_k)\le f(x_k)+c_1\alpha\nabla f(x_k)^T d_k,\ c_1\in(0,1)$
  - 计算搜索方向上的最大步长，沿着搜索方向移动一个足够大的步长值，然后以迭代形式不断缩减步长，直到新步长使得函数值$f(\mathbf{x_k}+\alpha\mathbf{d_k})$相对于当前函数值$f(\mathbf{x_k})$的减小程度大于预设的期望值(即满足Armijo准则)为止。

复习：三种决策树学习算法

- ID3：使用信息增益/互信息g(D,A)进行特征选择
  - 取值多的属性，更容易使数据更纯，其信息增益更大。
  - 训练得到的是一棵庞大且深度浅的树：不合理。
- C4.5：信息增益率 $g_r(D,A) = g(D,A) / H(A)$
- CART：基尼指数

- 一个属性的信息增益(率)/gini指数越大，表明属性对样本的熵减少的能力更强，这个属性使得数据由不确定性变成确定性的能力越强。

剪枝系数的确定

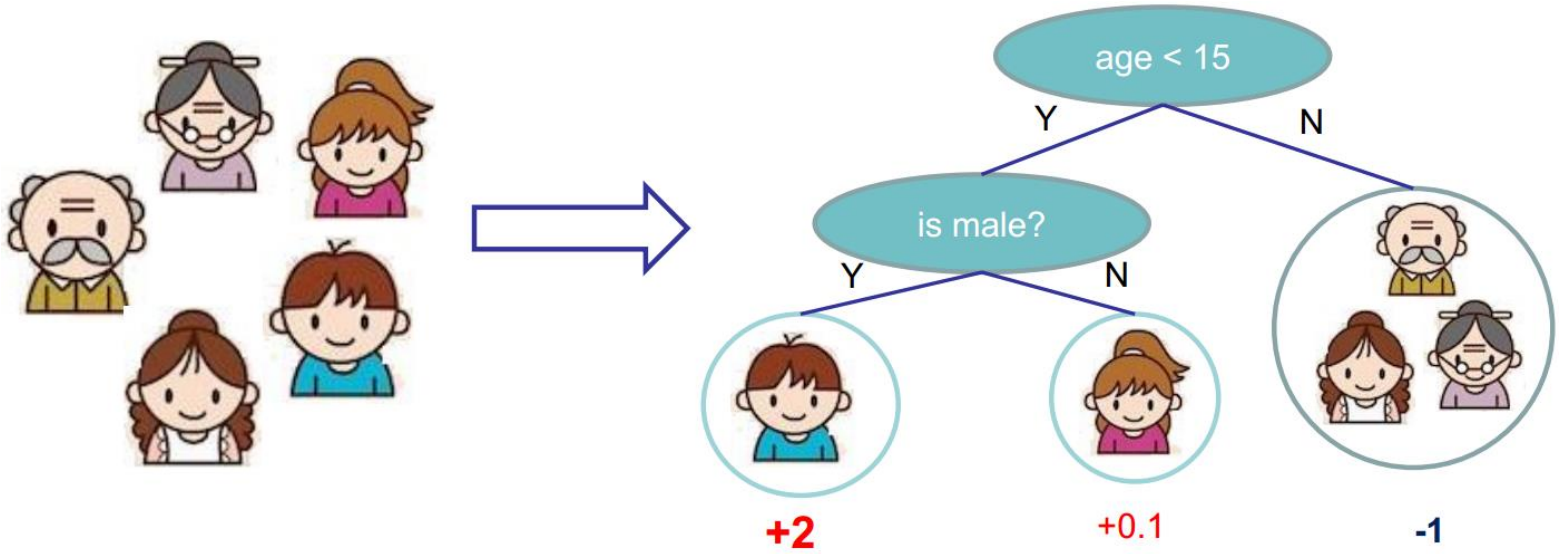
- 根据原损失函数 $C(T)=\sum_{t\in leaf} N_t \cdot H(t)$
- 叶结点越多，决策树越复杂，损失越大，修正：
  - 当α=0时，未剪枝的决策树损失最小；$C_\alpha(T)=C(T)+\alpha\cdot<T_{leaf}>$
  - 当α=+∞时，单根结点的决策树损失最小。
- 假定当前对以r为根的子树剪枝：
  - 剪枝后，只保留r本身而删掉所有的叶子
- 考察以r为根的子树：
  - 剪枝后的损失函数：$C_\alpha(r)=C(r)+\alpha$
  - 剪枝前的损失函数：$C_\alpha(R)=C(R)+\alpha\cdot<R_{leaf}>$
  - 令二者相等，求得：$\alpha=\frac{C(r)-C(R)}{<R_{leaf}>-1}$
- α称为结点r的剪枝系数。

随机森林

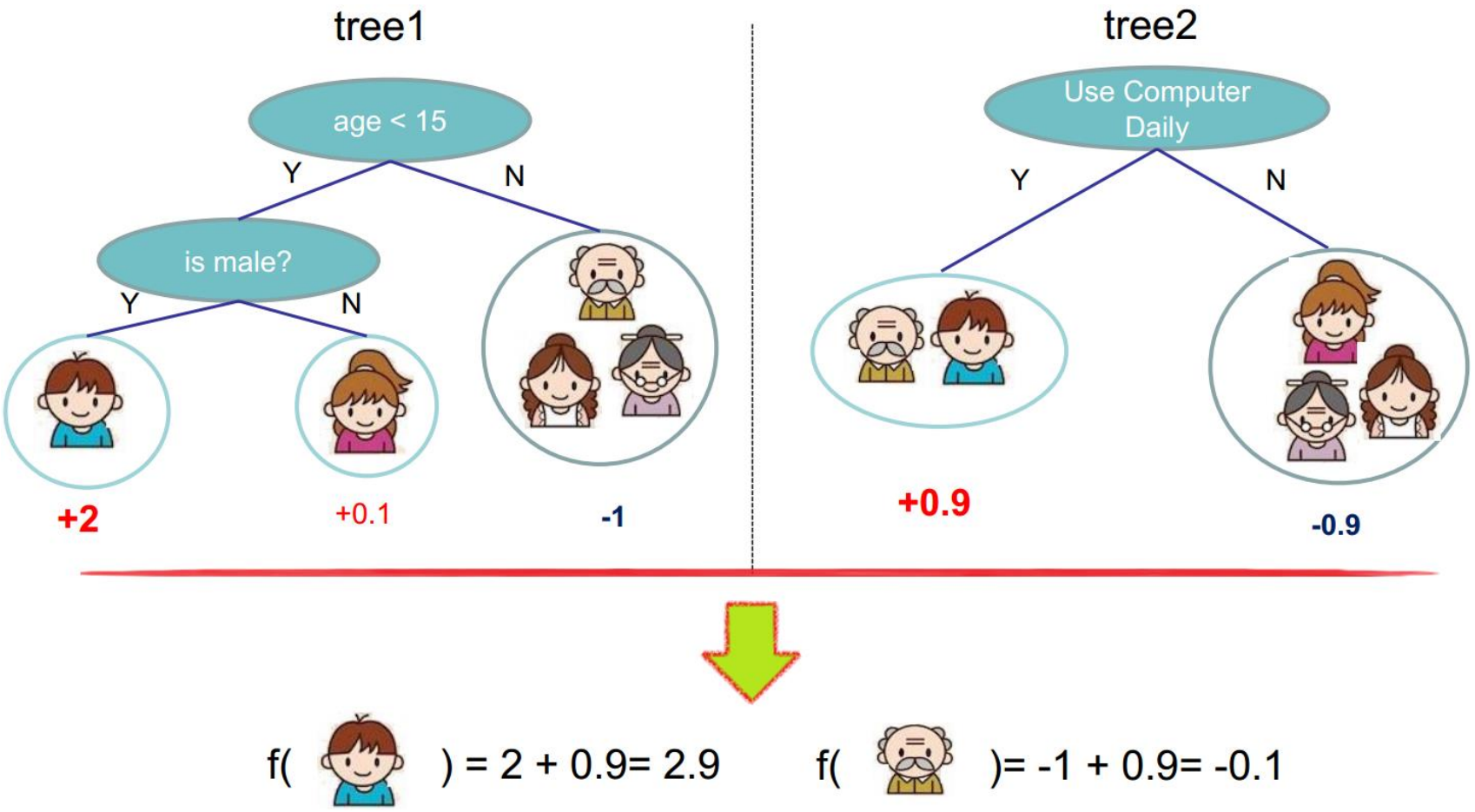
- 随机森林在bagging基础上做了修改。
  - 从样本集中用Bootstrap采样选出n个样本；
  - 从所有属性中随机选择k个属性，选择最佳分割属性作为节点建立CART决策树；
  - 重复以上两步m次，即建立了m棵CART决策树
  - 这m个CART形成随机森林，通过投票表决结果，决定数据属于哪一类

CART

- 输入数据x：M个样本数据，每个数据包括年龄、性别、职业、每日使用计算机时间等
- 输出y：该样本是否喜欢计算机游戏



随机森林


tree1
age < 15
Y
N
is male?
Y
N
+2
+0.1
-1
tree2
Use Computer Daily
Y
N
+0.9
-0.9
f( ) = 2 + 0.9= 2.9
f( )= -1 + 0.9= -0.1

由决策树和随机森林的关系的思考

- 以上即随机森林的整体过程和思路。
- 显然，随机森林是各个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？

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$(\mathbf{x_1},y_1),(\mathbf{x_2},y_2),\ldots,(\mathbf{x_n},y_n)$，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
- $L(y,F(x))$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F0(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$f(\mathbf{x_1}),f(\mathbf{x_2}),...,f(\mathbf{x_n})$，从而L退化为向量$L(y_1,f(\mathbf{x_1})),L(y_2,f(\mathbf{x_2})),...,L(y_n,f(\mathbf{x_n}))$

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

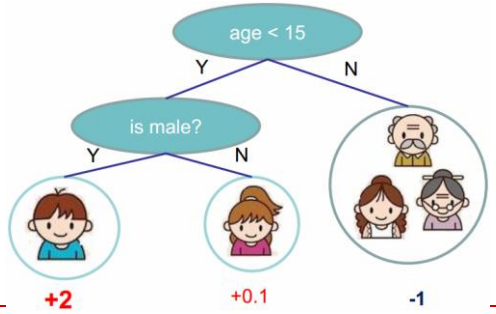
- 上式中的权值γ为梯度下降的步长，使用线性搜索求最优步长：

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

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 $r_{im} = \left[\frac{\partial L(y_i, F(\vec{x}_i))}{\partial F(\vec{x}_i)} \right]_{F(\vec{x})=F_{m-1}(\vec{x})} \quad i = 1,2,\cdots,n$
    - 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 $F_m(\vec{x}) = F_{m-1}(\vec{x}) - \gamma_m f_m(\vec{x}_i)$

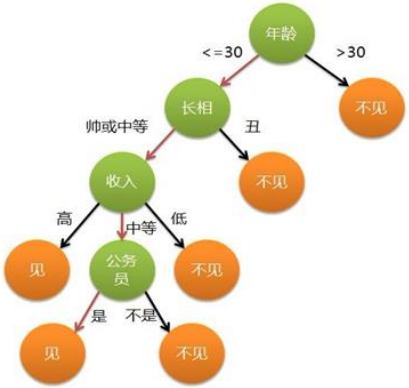
梯度提升决策树GBDT



- 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$t_m(x)$将输入空间划分为J个不相交区域$R_{1m},R_{2m},\ldots,R_{Jm}$，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(x)，对于输入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。



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

- 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 $F_m(\vec{x}) = F_{m-1}(\vec{x}) + \nu \cdot \gamma_m f_m(\vec{x}_i), \quad 0 < \nu \leq 1$
  - 称ν为学习率
  - ν=1即为原始模型；推荐选择ν<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。令训练样本数占所有伪残差样本的比例为f；f=1即为原始模型；推荐0.5≤f≤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。

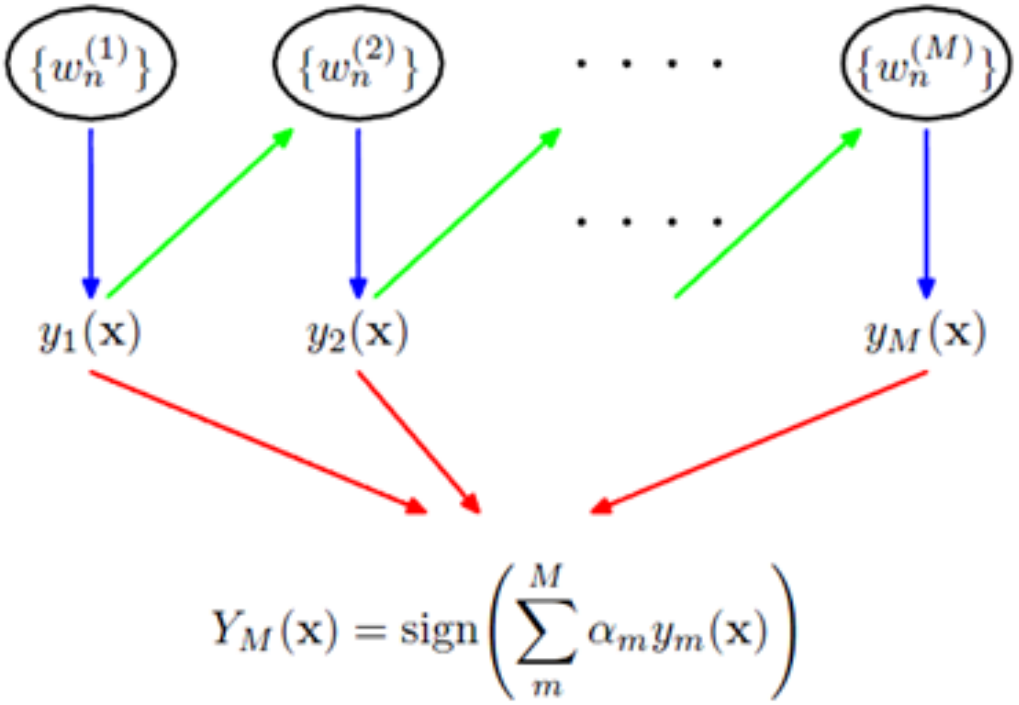
$$F^*(\vec{x}) = \arg\min_F E_{(x,y)}[L(y, F(\vec{x}))]$$

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？

boosting的思想



Adaboost

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

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

Adaboost：对于m=1,2,…M

- 使用具有权值分布Dm的训练数据集学习，得到基本分类器

$$G_m(x): \chi \to \{-1,+1\}$$

- 计算Gm(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Gm(x)的系数

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

Adaboost：对于m=1,2,…M

- 更新训练数据集的权值分布

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

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

- 这里，Zm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

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

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

- G1(x)在训练数据集上的误差率e1=P(G1(xi)≠yi) = 0.3
- 计算G1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$sign(f_1(x))$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：

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

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

- 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
  e2=P(G2(xi)≠yi) = 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：

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

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

- D3=(0.0455, 0.0455, 0.0455, 0.1667, 0.1667, 0.01667, 0.1060, 0.1060, 0.1060, 0.0455)
- f2(x)=0.4236G1(x) + 0.6496G2(x)
- 分类器sign(f2(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
  e3=P(G3(xi)≠yi) = 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：

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

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

- D4=(0.125, 0.125, 0.125, 0.102, 0.102, 0.102, 0.065, 0.065, 0.065, 0.125)
- f3(x)=0.4236G1(x) + 0.6496G2(x)+0.7514G3(x)
- 分类器sign(f3(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(xi)≠yi时，yi*f(xi)<0，$\exp(-y_i f(x_i))$≥前半部分得证。
- 后半部分：

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

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

后半部分

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

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

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

$$\frac{1}{N}\sum_i \exp\left(-y_i f(x_i)\right) = \sum_i \frac{1}{N}\exp\left(-\sum_{m=1}^{M}\alpha_m y_i G_m(x_i)\right)$$

$$= \sum_i w_{1i}\exp\left(-\sum_{m=1}^{M}\alpha_m y_i G_m(x_i)\right) = \sum_i w_{1i}\prod_{m=1}^{M}\exp\left(-\alpha_m y_i G_m(x_i)\right)$$

$$= \sum_i w_{1i}\exp\left(-\alpha_1 y_i G_1(x_i)\right)\prod_{m=2}^{M}\exp\left(-\alpha_m y_i G_m(x_i)\right)$$

$$= \sum_i Z_1 w_{2i}\prod_{m=2}^{M}\exp\left(-\alpha_m y_i G_m(x_i)\right) = Z_1\sum_i w_{2i}\prod_{m=2}^{M}\exp\left(-\alpha_m y_i G_m(x_i)\right)$$

$$= Z_1 Z_2\sum_i w_{3i}\prod_{m=3}^{M}\exp\left(-\alpha_m y_i G_m(x_i)\right)$$

$$= Z_1 Z_2\cdots Z_{M-1}\sum_i w_{Mi}\exp\left(-\alpha_M y_i G_M(x_i)\right)$$

$$= \prod_{m=1}^{M} Z_m$$

训练误差界

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

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

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

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

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

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

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

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

取γ1, γ2… 的最小值，记做γ

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

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={(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：

前向分步算法的算法框架

- 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：

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

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

基本分类器G*(x)

- 首先求分类器G*(x)
- 对于任意α>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，$\overline{w}_{mi} = \exp(-y_i f_{m-1}(x_i))$

权值的计算

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

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

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

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

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

权值的更新

- 由模型

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

- 以及权值

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

- 可以方便的得到：

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

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       # 建立多少颗树
tree_weight = [0] * tree_number  # 每颗树的权值
data = []              # 样本数据
```

```
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       # 该节点的左孩子
        self.right = -1      # 该节点的右孩子
        self.gini = 0        # 该结点的gini系数
```

```
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)
```

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

```
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
```

```
# 将样本分配到直方柱中
for i in range(m):
    d = data[self.sample[i]][-1] * sel
    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)
```

Code

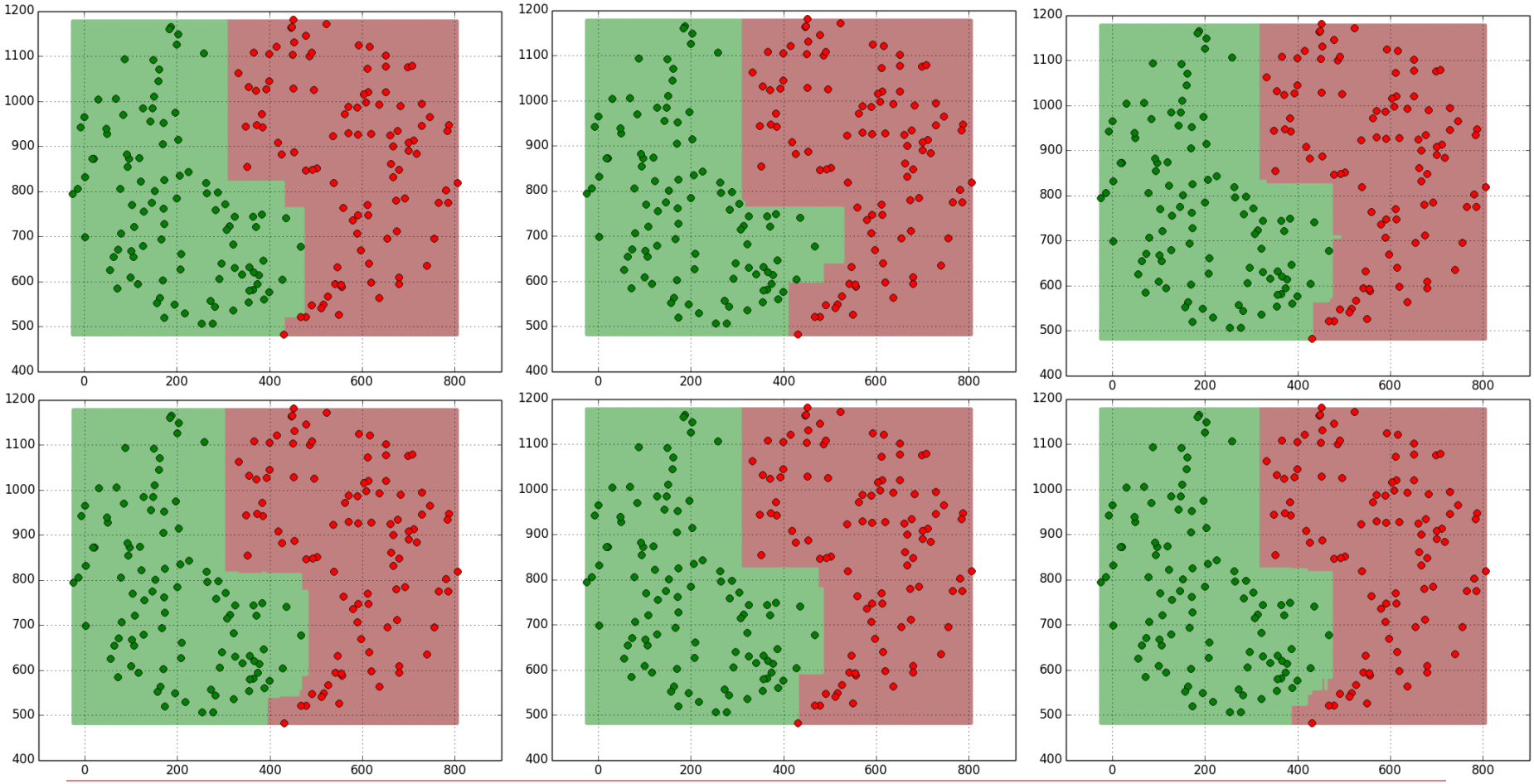
```
    def split(self, tree):
        f = self.select_feature()
        self.choose_value(f, tree)

    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

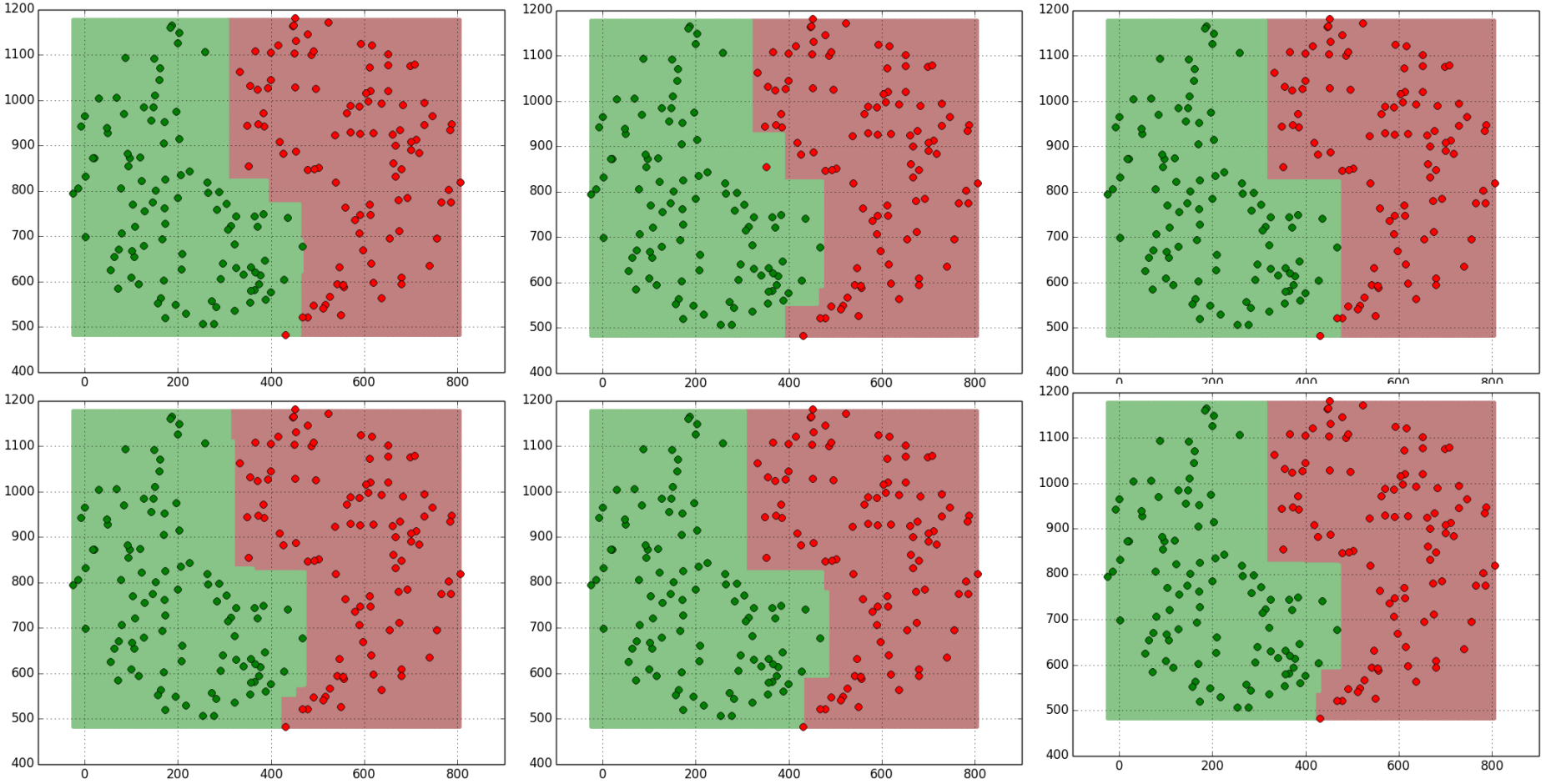
```
def choose_value(self, f, tree):
    f_max = self.calc_max(f)
    f_min = self.calc_min(f)
    step = (f_max - f_min) / granularity
    if step == 0:
        return f_min
    x_split = 0
    g_split = 0
    for x in numpy.arange(f_min+step, f_max, step):
        # if rf:    # 理论上，随机选择优于均匀采样
        #    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



参考文献

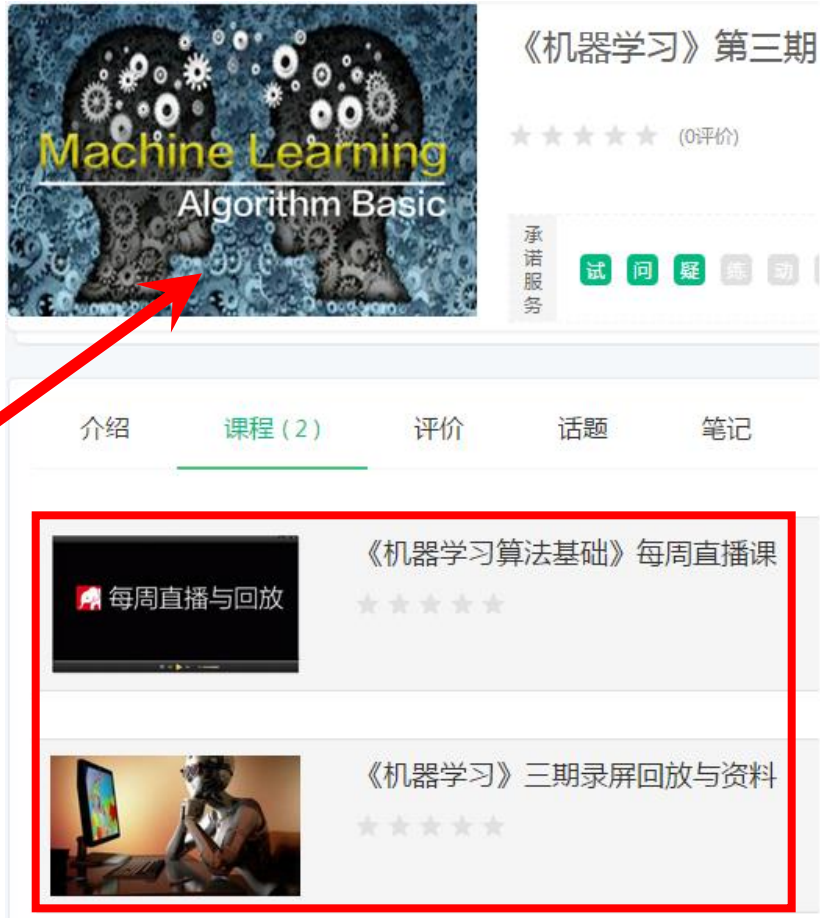
- Jerome H. Friedman. *Greedy Function Approximation: A Gradient Boosting Machine*. February 1999
- Jerome H. Friedman. *Stochastic Gradient Boosting*. March 1999
- 李航，统计学习方法，清华大学出版社，2012
- https://en.wikipedia.org/wiki/Gradient_boosting#Gradient_tree_boosting

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

课程资源

- 直播课的入口
- 录播视频和讲义资料



感谢大家！

恳请大家批评指正！