

第十一章 集成学习 ——Boosting

卿来云

➤ 集成学习

- 我们已经开发了很多机器学习算法/代码。
- 单个模型的性能已经调到最优，很难再有改进。
- 集成学习：用很少量的工作，组合多个基模型，使得系统总的性能提高
 - 基模型最好变化多样，这样不同的基模型集成后形成互补。

Outline

- 模型性能评价
 - No Free Lunch Theorems
 - Occam剃刀原理
 - 偏差-方差折中
- Bagging
 - 随机森林
- Boosting
 - AdaBoost
 - Gradient Boosting Decision Tree (GBDT)
 - XGBoost
 - LightGBM
- Stacking

➤ Boosting

- Boosting: 将弱学习器组合成强分类器
 - 构造一个性能很高的预测（强学习器）是一件很困难的事情
 - 但构造一个性能一般的预测（弱学习器）并不难
 - 弱学习器：性能比随机猜测略好（如层数不深的决策树）
- Boosting学习框架
 - 学习第一个弱学习器 ϕ_1
 - 学习第二个弱学习器 ϕ_2 ， ϕ_2 要能帮助 ϕ_1 （ ϕ_2 和 ϕ_1 互补）
 - ...
 - 组合所有的弱学习器： $f(\mathbf{x}) = \sum_{m=1}^M \alpha_m \phi_m(\mathbf{x})$

弱学习器是按顺序学习的。

➤ 怎样得到互补的学习器？

- 在不同的训练集上训练学习器。
- 怎么得到不同的训练集?
 - 对原始训练集重采样
 - 对原始训练集重新加权
 - 在实际操作中，改变目标函数即可：

$(\mathbf{x}_1, y_1, \textcolor{red}{w}_1)$

...

$(\mathbf{x}_N, y_N, \textcolor{red}{w}_N)$

$$J(f, \lambda) = \sum_{i=1}^N L(y_i, f(\mathbf{x}_i)) + \lambda R(f)$$



$$J(f, \lambda) = \sum_{i=1}^N \textcolor{red}{w}_i L(y_i, f(\mathbf{x}_i)) + \lambda R(f)$$

➤ AdaBoost的基本思想

- 在弱学习器 ϕ_1 失败的样本上学习第二个弱学习器 ϕ_2
- 令弱学习器 ϕ_1 在其训练集上的误差为：

$\mathbb{I}(condition)$: 示性 (Indicator) 函数
满足条件值为1，否则为0

$$\varepsilon_1 = \sum_{i=1}^N w_{1,i} \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i)) < \frac{1}{2},$$

- 将权重由 w_1 变成 w_2 ，使得

$$\sum_{i=1}^N w_{2,i} \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i)) = \frac{1}{2}$$

学习器 ϕ_1 在训练集2上的性能为随机猜测

- 根据权重 w_2 训练弱学习器 ϕ_2

➤ 样本重新加权

■ 如何加权？

- 分对的样本，其权重除以 d_1 权重减小
- 分错的样本，其权重乘以 d_1 权重增大

■ d_1 的值？

■ 为了使得 \mathbf{w}_2 还是一个分布， \mathbf{w}_1 乘以/除以 d_1 后的权重，再除以归一化因子： $Z_1 = \sum_{i=1}^N \mathbf{w}_{1,i} d_1 \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i)) + \sum_{i=1}^N \mathbf{w}_{1,i} / d_1 \mathbb{I}(y_i = \phi_1(\mathbf{x}_i))$ ，

■ 即

- 分对的样本的权重： $w_{2,i} = \frac{\mathbf{w}_{1,i} / d_1}{Z_1}$
- 分错的样本的权重： $w_{2,i} = \frac{\mathbf{w}_{1,i} d_1}{Z_1}$

➤ 样本重新加权

$$Z_1 = \sum_{i=1}^N w_{1,i} d_1 \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i)) + \sum_{i=1}^N w_{1,i}/d_1 \mathbb{I}(y_i = \phi_1(\mathbf{x}_i))$$

$$\sum_{i=1}^N w_{2,i} \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i)) = \sum_{i=1}^N \frac{w_{1,i} d_1}{Z_1} \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i))$$

$$= \frac{\sum_{i=1}^N w_{1,i} d_1 \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i))}{\sum_{i=1}^N w_{1,i} d_1 \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i)) + \sum_{i=1}^N w_{1,i}/d_1 \mathbb{I}(y_i = \phi_1(\mathbf{x}_i))} = \frac{1}{2}$$

$$\sum_{i=1}^N w_{1,i} d_1 \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i)) = \sum_{i=1}^N w_{1,i}/d_1 \mathbb{I}(y_i = \phi_1(\mathbf{x}_i))$$

➤ 样本重新加权

$$\varepsilon_1 = \sum_{i=1}^N w_{1,i} \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i))$$

$$\sum_{i=1}^N w_{1,i} d_1 \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i)) = \sum_{i=1}^N w_{1,i} / d_1 \mathbb{I}(y_i = \phi_1(\mathbf{x}_i))$$

$$d_1 \underbrace{\sum_{i=1}^N w_{1,i} \mathbb{I}(y_i \neq \phi_1(\mathbf{x}_i))}_{\varepsilon_1} = \frac{1}{d_1} \underbrace{\sum_{i=1}^N w_{1,i} \mathbb{I}(y_i = \phi_1(\mathbf{x}_i))}_{(1-\varepsilon_1)}$$

$$d_1 \varepsilon_1 = \frac{1}{d_1} (1 - \varepsilon_1) \quad \rightarrow \quad d_1 = \sqrt{1 - \varepsilon_1 / \varepsilon_1} > 1$$

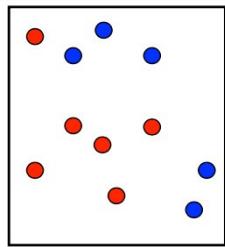
➤ AdaBoost M1算法

- 给定训练集： $(x_1, y_1), \dots, (x_N, y_N)$ ，其中 $y_i \in \{0,1\}$ 表示 x_i 的类别标签
- 训练集上样本的初始分布： $w_{1,i} = \frac{1}{N}$
- 对 $m=1:M$ ，
 - 对训练样本采用权重 $w_{m,i}$ ，计算弱分类器 $\phi_m(\mathbf{x})$
 - 计算该弱分类器 $\phi_m(\mathbf{x})$ 在分布 w_m 上的误差： $\varepsilon_m = \sum_{i=1}^N w_{m,i} \mathbb{I}(\phi_m(\mathbf{x}_i) \neq y_i)$
 - 计算： $d_m = \sqrt{1 - \varepsilon_m / \varepsilon_m}$, $\alpha_m = \log(d_m) = \frac{1}{2} \log \frac{1-\varepsilon_m}{\varepsilon_m}$
 - 更新训练样本的分布： $w_{m+1,i} = \begin{cases} \frac{w_{m,i}/d_m}{Z_m} = \frac{w_{m,i} d_m^{-1}}{Z_m} & y_i \phi_m(\mathbf{x}_i) = 1 \text{ (对分样本)} \\ \frac{w_{m,i} d_m}{Z_m} = \frac{w_{m,i} d_m^1}{Z_m} & y_i \phi_m(\mathbf{x}_i) = -1 \text{ (分错样本)} \end{cases} = \frac{w_{m,i} d_m^{-y_i \phi_m(\mathbf{x}_i)}}{Z_m} = \frac{w_{m,i} \exp(-\alpha_m y_i \phi_m(\mathbf{x}_i))}{Z_m}$ ，其中 Z_m 为归一化常数，使得 w_{m+1} 是个分布。
- 最后的强分类器为： $f(\mathbf{x}) = \operatorname{sgn}(\sum_{m=1}^M \alpha_m \phi_m(\mathbf{x}))$

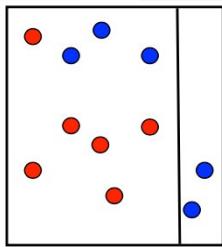
准确率越高的弱学习机权重越高。

➤ Toy Example

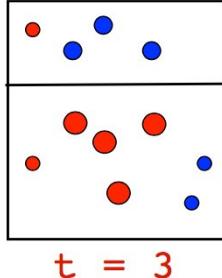
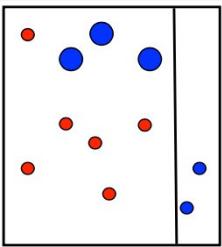
初始样本权重



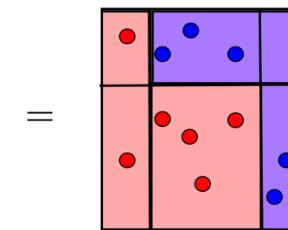
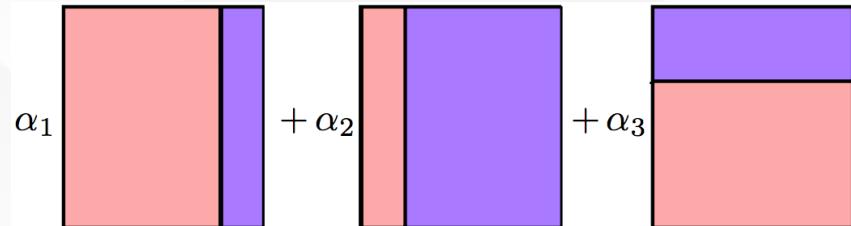
增加弱学习器之
前的样本权重



增加弱学习器之
后的样本权重



弱学习器：深度为1的决策树（树桩，stumps）



强学习器

➤ 训练误差

■ 可以证明：随着弱学习器数目增多，训练误差越来越小。

证明

■ 1. 对 w_{M+1} 进行迭代展开

$$\begin{aligned} w_{M+1,i} &= w_{M,i} \frac{\exp(-\alpha_M y_i \phi_M(\mathbf{x}_i))}{Z_M} = w_{1,i} \frac{\exp(-y_i \sum_{m=1}^M \alpha_m \phi_m(\mathbf{x}_i))}{\prod_{m=1}^M Z_m} \\ &= w_{1,i} \frac{\exp(-y_i f(\mathbf{x}_i))}{\prod_{m=1}^M Z_m} \end{aligned}$$

■ 由于 w_{M+1} 是一个分布，所以 $\sum_{i=1}^N w_{M+1,i} = 1$

■ 所以 $\prod_{m=1}^M Z_m = \sum_{i=1}^N w_{1,i} \exp(-y_i f(\mathbf{x}_i)) = \frac{1}{N} \sum_{i=1}^N \exp(-y_i f(\mathbf{x}_i))$ 。

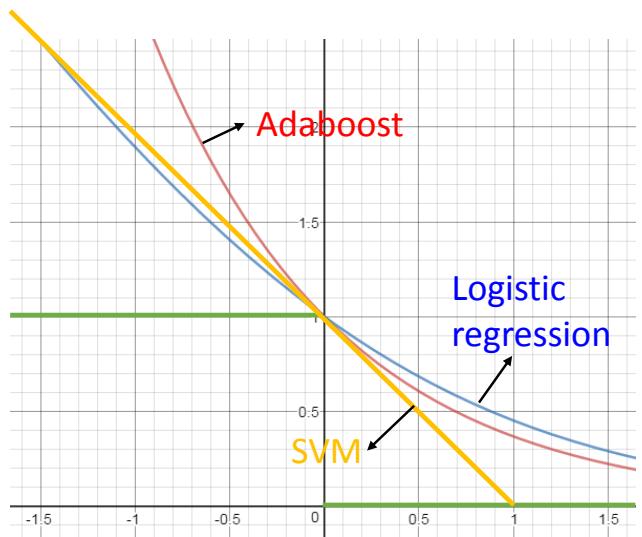
$$f(\mathbf{x}) = \sum_{m=1}^M \alpha_m \phi_m(\mathbf{x})$$

形式与训练误差很像

➤ 证明 (cont.)

$$\prod_{m=1}^M Z_m = \frac{1}{N} \sum_{i=1}^N \exp(-y_i f(\mathbf{x}_i))$$

■ 2. 训练误差为 : $ERR_{train}(f(\mathbf{x})) = \frac{1}{N} |\{i : y_i \neq \text{sgn}(f(\mathbf{x}_i))\}|$



$$= \frac{1}{N} \sum_{i=1}^N \begin{cases} 1 & y_i \neq \text{sgn}(f(\mathbf{x}_i)) \\ 0 & \text{else} \end{cases} \quad \text{0-1损失}$$

$$\leq \frac{1}{N} \sum_{i=1}^N \exp(-y_i f(\mathbf{x}_i)) \quad \text{指数损失}$$

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

相当于损失函数为指数损失函数 : $L(f(\mathbf{x}), y) = \exp(-y f(\mathbf{x}))$, 训练误差为 :

$$ERR_{train}(f(\mathbf{x})) = \frac{1}{N} \sum_{i=1}^N \exp(-y_i f(\mathbf{x}_i))$$

➤ 证明 (cont.)

■ 3. Z_m 的计算公式为 : 对 $m = 1, \dots, M$,

$$\begin{aligned} Z_m &= \sum_{i=1}^N w_{m,i} = \sum_{i=1}^N w_{m,i} d_m \mathbb{I}(y_i \neq \phi_m(\mathbf{x}_i)) + \sum_{i=1}^N w_{m,i} / d_m \mathbb{I}(y_i = \phi_m(\mathbf{x}_i)) \\ &= d_m \sum_{i=1}^N w_{m,i} \mathbb{I}(y_i \neq \phi_m(\mathbf{x}_i)) + \frac{1}{d_m} \sum_{i=1}^N w_{m,i} \mathbb{I}(y_i = \phi_m(\mathbf{x}_i)) \\ &= d_m \varepsilon_m + \frac{1}{d_m} (1 - \varepsilon_m) \\ &= \varepsilon_m \sqrt{1 - \varepsilon_m / \varepsilon_m} + (1 - \varepsilon_m) \sqrt{\varepsilon_m / (1 - \varepsilon_m)} \\ &= \sqrt{\varepsilon_m (1 - \varepsilon_m)} + \sqrt{\varepsilon_m (1 - \varepsilon_m)} \\ &= 2 \sqrt{\varepsilon_m (1 - \varepsilon_m)} \end{aligned}$$

$$d_m = \sqrt{1 - \varepsilon_m / \varepsilon_m}$$

➤ 证明 (cont.)

- 4. 训练误差为 :

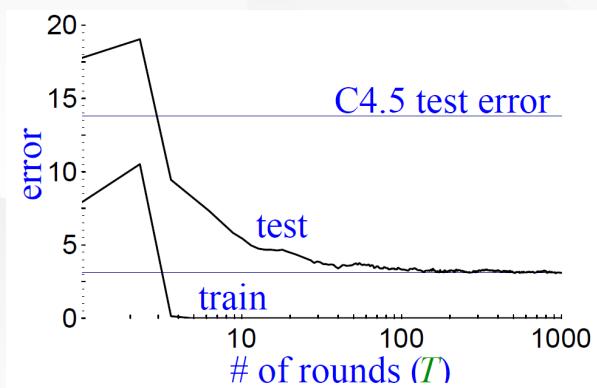
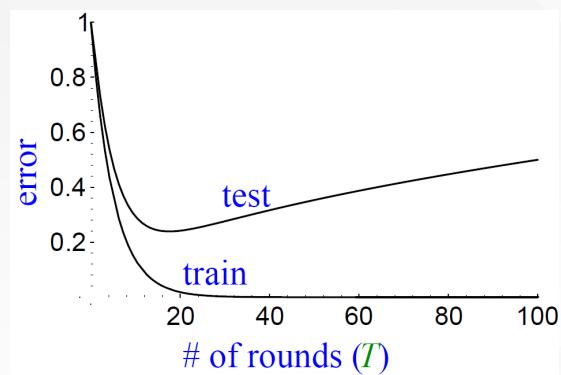
$$Z_m = 2\sqrt{\varepsilon_m(1 - \varepsilon_m)}$$

$$\frac{1}{N} \sum_{i=1}^N \exp(-y_i f(\mathbf{x}_i)) = \prod_{m=1}^M Z_m = 2^M \prod_{m=1}^M \sqrt{\varepsilon_m(1 - \varepsilon_m)}$$

- 由于 $0 < \varepsilon_m < \frac{1}{2}$, $0 < 1 - \varepsilon_m < 1$, $\sqrt{\varepsilon_m(1 - \varepsilon_m)} < 1$,
- 所以训练误差随着 M 的增加越来越小。

➤ 测试误差

■ 训练误差随着 M 的增加而减小。测试误差呢？



(boosting C4.5 on
“letter” dataset)

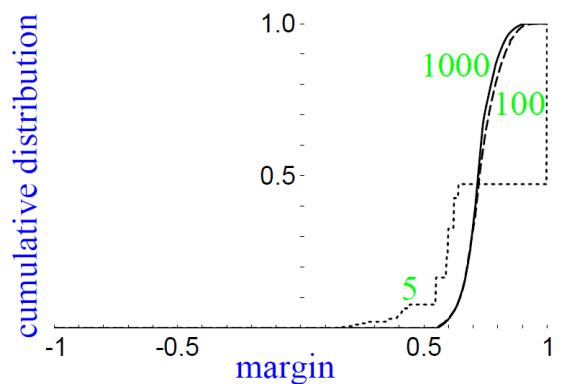
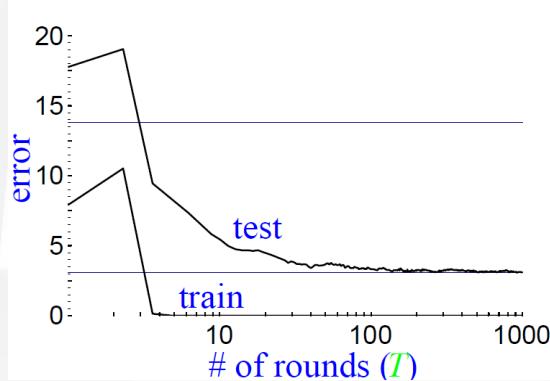


	# rounds	5	100	1000
train error	0.0	0.0	0.0	
test error	8.4	3.3	3.1	

➤ 测试误差

回忆SVM中间隔的概念

- 测试误差随着 M 的增加而减小可用间隔来解释。
- 上述训练误差（0-1损失）只考虑了分类是否正确，还应该考虑分类的置信度。
- 分类的置信度：判别函数的值 $yf(\mathbf{x})$



	# rounds	5	100	1000
train error	0.0	0.0	0.0	0.0
test error	8.4	3.3	3.1	3.1
% margins ≤ 0.5	7.7	0.0	0.0	0.0
minimum margin	0.14	0.52	0.55	0.55

Outline

- 模型性能评价
 - No Free Lunch Theorems
 - Occam剃刀原理
 - 偏差-方差折中
- Bagging
 - 随机森林
- Boosting
 - AdaBoost
 - Gradient Boosting Decision Tree (GBDT)
 - XGBoost
 - LightGBM
 - Stacking

➤ Boosting的一般框架

- 1. 初始化 $f_0(\mathbf{x})$
- 2. for $m = 1: M$ do
 - 找一个弱学习器 $\phi_m(\mathbf{x})$ ，使得 $\phi_m(\mathbf{x})$ 能改进 $f_{m-1}(\mathbf{x})$ ；
 - 更新 $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \alpha_m \phi_m(\mathbf{x})$
- 3. return $f(\mathbf{x}) = f_M(\mathbf{x})$

► Gradient Boosting

- 目标：损失函数 $J(f) = \sum_{i=1}^N L(f(\mathbf{x}_i), y_i)$ 最小
- 若已有 $f_m(\mathbf{x})$ ，如何在 $f_m(\mathbf{x})$ 的基础上改进 $f(\mathbf{x})$ ？
- 梯度下降：
$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) - \eta \left[\frac{\partial J(f)}{\partial f(\mathbf{x})} \right]_{f(\mathbf{x})=f_{m-1}(\mathbf{x})}$$
- 对比：
$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \alpha_m \phi_m(\mathbf{x})$$

用弱学习器来拟合目标函数的负梯度。

► Gradient Boosting Algorithm

- 1. Initialize $f_0(\mathbf{x}) = \operatorname{argmin}_f \frac{1}{N} \sum_{i=1}^N L(y_i, f(\mathbf{x}_i))$
- 2. for $m = 1: M$ do
 - Compute the gradient residual using $r_{m,i} = - \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f=f_{m-1}}$
 - Use the weak learner which minimizes $\sum_{i=1}^N (r_{m,i} - \phi_m(\mathbf{x}_i))^2$
 - Update $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \eta \phi_m(\mathbf{x})$
- 3. return $f(\mathbf{x}) = f_M(\mathbf{x})$

从Gradient Descent 到 Gradient Boosting

参数空间

$$\theta^t = \theta^{t-1} + \theta_t$$

第t次迭代后的参数 第t-1次迭代后的参数 第t次迭代的参数增量

$$\theta_t = -\alpha_t g_t$$

参数更新方向为负梯度方向

$$\theta = \sum_{t=0}^T \theta_t$$

最终参数等于每次迭代的增量的累加和，

θ_0 为初值

函数空间

$$f^t(x) = f^{t-1}(x) + f_t(x)$$

第t次迭代后的函数 第t-1次迭代后的函数 第t次迭代的函数增量

$$f_t(x) = -\alpha_t g_t(x)$$

同样地，拟合负梯度

$$F(x) = \sum_{t=0}^T f_t(x)$$

最终函数等于每次迭代的增量的累加和，

$f_0(x)$ 为模型初始值，通常为常数

From : <http://wepon.me/files/gbdt.pdf>

➤ AdaBoost as Gradient Boosting

- 将指数损失 $L(f(\mathbf{x}), y) = \exp(-yf(\mathbf{x}))$ 代入，
- $J(f) = \sum_{i=1}^N L(f(\mathbf{x}_i), y_i) = \sum_{i=1}^N \exp(-y_i f(\mathbf{x}_i))$
- $\frac{\partial J(f)}{\partial f} = \sum_{i=1}^N -y_i \exp(-y_i f(\mathbf{x}_i))$
- 第 m 步，负梯度为： $-\left[\frac{\partial J(f)}{\partial f}\right]_{f=f_{m-1}} = \sum_{i=1}^N y_i \underbrace{\exp(-y_i f_{m-1}(\mathbf{x}_i))}_{w_{m,i}}$ AdaBoost中的样本权重
- 弱学习器 $\phi_m(\mathbf{x})$ 要尽可能拟合负梯度，则 $\phi_m(\mathbf{x}_i)$ 尽可能与 y_i 同号
(预测正确)，即最佳的 ϕ_m 为错误率最小的弱分类器 (每个样本的权重为 $w_{m,i}$)。

➤ AdaBoost as Gradient Boosting

- 令 $\frac{\partial J(f)}{\partial \alpha_m} = 0$ ，得到 $\alpha_m = \frac{1}{2} \log \frac{1-\varepsilon_m}{\varepsilon_m}$ 。
- 证明： $J(f) = \sum_{i=1}^N L(f(\mathbf{x}_i), y_i)$

$$= \sum_{i=1}^N \exp \left(-y_i (f_{m-1}(\mathbf{x}_i) + \alpha_m \phi_m(\mathbf{x}_i)) \right)$$

$$= \sum_{i=1}^N \underbrace{\exp(-y_i f_{m-1}(\mathbf{x}_i))}_{w_{m,i}} \exp(-y_i \alpha_m \phi_m(\mathbf{x}_i)) \text{ AdaBoost中的样本权重}$$

$$= e^{-\alpha_m} \sum_{y_i=\phi_m(\mathbf{x}_i)} w_{m,i} + e^{\alpha_m} \sum_{y_i \neq \phi_m(\mathbf{x}_i)} w_{m,i}$$

$$\begin{cases} y_i = \phi_m(\mathbf{x}_i) & y_i \phi_m(\mathbf{x}_i) = 1 \\ y_i \neq \phi_m(\mathbf{x}_i) & y_i \phi_m(\mathbf{x}_i) = -1 \end{cases}$$

➤ AdaBoost as Gradient Boosting (cont.)

$$\blacksquare \quad \frac{\partial J(f)}{\partial \alpha_m} = -e^{-\alpha_m} \sum_{y_i=\phi_m(\mathbf{x}_i)} w_{m,i} + e^{\alpha_m} \sum_{y_i \neq \phi_m(\mathbf{x}_i)} w_{m,i} = 0$$

$$e^{-\alpha_m} \sum_{y_i=\phi_m(\mathbf{x}_i)} w_{m,i} = e^{\alpha_m} \sum_{y_i \neq \phi_m(\mathbf{x}_i)} w_{m,i}$$

$$\blacksquare \text{错误率 } \varepsilon_m = \sum_{i=1}^N w_{m,i} \mathbb{I}(y_i \neq \phi_m(\mathbf{x}_i)) = \sum_{y_i \neq \phi_m(\mathbf{x}_i)} w_{m,i} ,$$

$$\blacksquare \sum_{y_i=\phi_m(\mathbf{x}_i)} w_{m,i} = 1 - \varepsilon_m ,$$

$$\blacksquare \text{则 } e^{-\alpha_m} (1 - \varepsilon_m) = e^{\alpha_m} \varepsilon_m \quad \rightarrow$$

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

错误率小的弱分类器的权重更大

➤ Gradient Boosting — 其他损失函数

- 指数损失对离群点（outliers）比较敏感，而且也不是任何二值变量的概率密度取log后的表示。
- 因此另一种选择是损失函数取负log似然损失，得到logitBoost。
- 对回归问题，损失函数可取L2损失，得到L2boosting。

➤ L2Boosting

- 对L2损失： $L(f(\mathbf{x}), y) = \frac{1}{2} (f(\mathbf{x}) - y)^2$
- $J(f) = \sum_{i=1}^N L(f(\mathbf{x}_i), y_i) = \frac{1}{2} \sum_{i=1}^N (f(\mathbf{x}_i) - y_i)^2$
- $\frac{\partial J(f)}{\partial f} = \sum_{i=1}^N (f(\mathbf{x}_i) - y_i)$
- 初始值： $f_0(\mathbf{x}) = \bar{y}$
- 在第 m 步，负梯度为： $-\left[\frac{\partial J(f)}{\partial f}\right]_{f=f_{m-1}} = -\sum_{i=1}^N (f_{m-1}(\mathbf{x}_i) - y_i) = -r_{m,i}$
为预测残差。
- 不失一般性，假设 $\alpha = 1$ ，因此用弱学习器来预测残差 r_m ，称为L2Boosting。

➤ L2Boosting

- 对L2损失，亦可直接从损失函数推导：
- $L(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2$
- 在第 m 步，损失函数的形式为
- $L(f_{m-1}(\mathbf{x}_i) + \alpha_m \phi_m(\mathbf{x}_i), y_i) = (f_{m-1}(\mathbf{x}_i) + \alpha_m \phi_m(\mathbf{x}_i) - y_i)^2$
- $=(-(y_i - f_{m-1}(\mathbf{x}_i)) + \alpha_m \phi_m(\mathbf{x}_i))^2 = (r_{m,i} - \alpha_m \phi_m(\mathbf{x}_i))^2$
- 其中 $r_{m,i} = f_{m-1}(\mathbf{x}_i) - y_i$
- 不失一般性，假设 $\alpha = 1$ ，因此用弱学习器来预测残差 r_m ，称为L2Boosting。

➤ Shrinkage

- 通常对系数增加一个小小的收缩因子(亦被称为学习率) , 测试性能更好 , 即

$$f_m(\mathbf{x}_i) = f_{m-1}(\mathbf{x}_i) + \eta \alpha_m \phi_m(\mathbf{x}_i)$$

- 其中 $0 < \eta < 1$, 通常取一个较小的值 , 如 $\eta = 0.01$ 。
- 较小的收缩因子通常意味着更多弱分学习器。

Outline

- 模型性能评价

 - No Free Lunch Theorems

 - Occam剃刀原理

 - 偏差-方差折中

- Bagging

 - 随机森林

- Boosting

 - AdaBoost

 - Gradient Boosting Decision Tree
(GBDT)

 - XGBoost

 - LightGBM

- Stacking

➤ XGBoost

- XGBoost : eXtreme Gradient Boosting
- XGBoost并不是一个新的算法，是Gradient Boosting 的C++优化实现，快速有效。

The name xgboost, though, actually refers to the engineering goal to **push the limit of computations resources for boosted tree algorithms**. Which is the reason why many people use xgboost.

– Tianqi Chen, on

Quora.com

➤ XGBoost

- 可自定义损失函数：采用二阶Taylor展开近似损失函数
- 规范化的正则项：叶子节点数目、叶子结点的分数
- 建树
 - 支持分裂点近似搜索
 - 稀疏特征处理
 - 缺失值处理
- 并行计算
- 内存缓存

推荐阅读：深入理解XGBoost (<https://blog.csdn.net/fengdu78/article/details/104284924>)

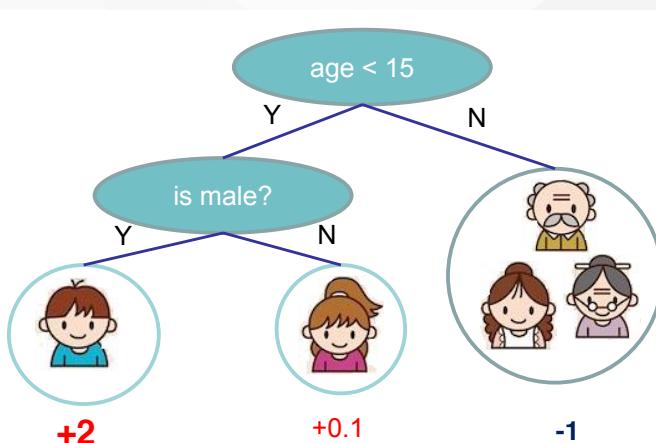
➤ 损失函数的二阶近似

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

- XGBoost : 对损失函数的二阶Taylor展开近似
- 在第 m 步时 , 令 $g_{m,i} = \left[\frac{\partial L(f(\mathbf{x}_i), y_i)}{\partial f(\mathbf{x}_i)} \right]_{f=f_{m-1}}$, $h_{m,i} = \left[\frac{\partial^2 L(f(\mathbf{x}_i), y_i)}{\partial^2 f(\mathbf{x}_i)} \right]_{f=f_{m-1}}$
- $L(y_i, f_{m-1}(\mathbf{x}_i) + \phi(\mathbf{x}_i)) = \underbrace{L(f_{m-1}(\mathbf{x}_i), y_i)}_{\text{与未知量 } \phi(\mathbf{x}_i) \text{无关}} + g_{m,i} \phi(\mathbf{x}_i) + \frac{1}{2} h_{m,i} \phi(\mathbf{x}_i)^2$
- 所以 $L(f_{m-1}(\mathbf{x}_i) + \phi(\mathbf{x}_i), y_i) = g_{m,i} \phi(\mathbf{x}_i) + \frac{1}{2} h_{m,i} \phi(\mathbf{x}_i)^2$
 - 对L2损失 , $L(f(\mathbf{x}), y) = \frac{1}{2}(f(\mathbf{x}) - y)^2$, $\nabla_f L(\boldsymbol{\theta}) = f(\mathbf{x}) - y$, $\nabla_f^2 L(\boldsymbol{\theta}) = 1$ 。
 - 所以 $g_{m,i} = f_{m-1}(\mathbf{x}_i) - y_i$, $h_{m,i} = 1$ 。

➤ 正则项

- 基学习器为决策树：把树拆分成结构部分 q 和叶子分数部分 w
- $\phi(\mathbf{x}) = w_{q(\mathbf{x})}$, $\mathbf{w} \in R^T$, $q: R^D \rightarrow \{1, \dots, T\}$
 - 结构函数 q ：把输入 \mathbf{x} 映射到叶子的索引号
 - w ：给出每个索引号对应的叶子的分数
 - T 为树中叶子结点的数目， D 为特征维数



q() = 1
q() = 3

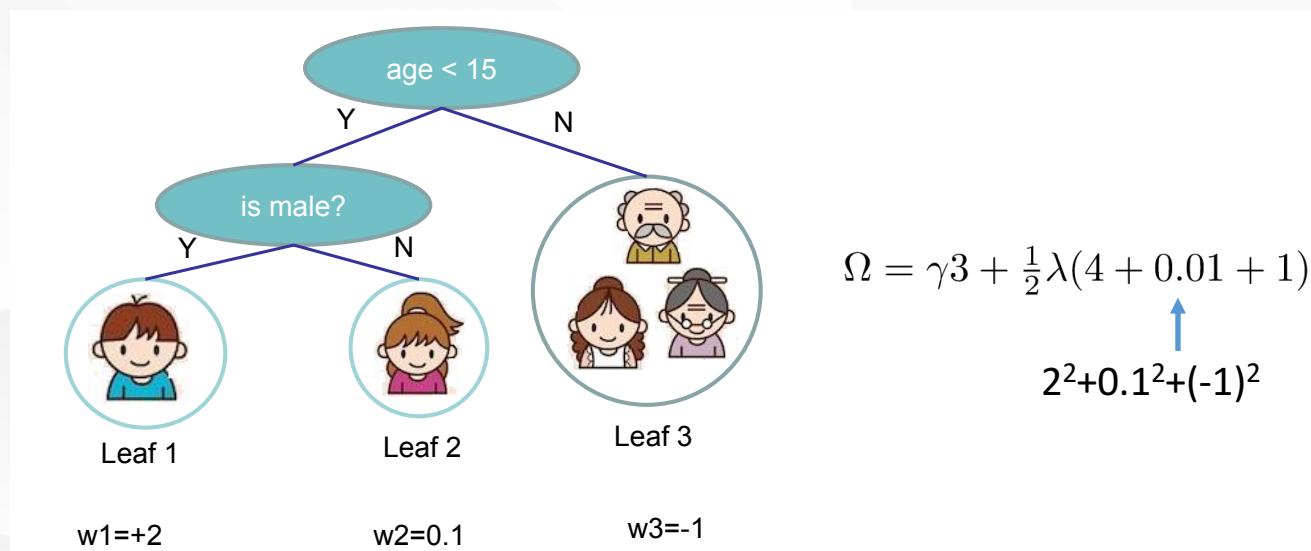
The right side of the slide shows two examples of mapping input features to leaf indices. The first example shows a boy icon next to the equation $q(\text{boy}) = 1$. The second example shows a girl icon next to the equation $q(\text{girl}) = 3$.

➤ 正则项

- 树的**复杂度**定义为（不是唯一的定义方式）：

$$R(\phi(x)) = \gamma T + \frac{1}{2} \lambda \sum_{t=1}^T w_t^2$$

- 叶子节点的数目 T （L1正则）、叶子节点分数的平方（L2正则）



▶ 目标函数

■ 令每个叶子 t 上的样本集合为 $I_t = \{i | q(\mathbf{x}_i) = t\}$

$$\begin{aligned} J(f) &= \sum_{i=1}^N L(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) + R(f) \\ &\cong \sum_{i=1}^N \left(g_{m,i} \phi(\mathbf{x}_i) + \frac{1}{2} h_{m,i} \phi(\mathbf{x}_i)^2 \right) + \gamma T + \frac{1}{2} \lambda \sum_{t=1}^T w_t^2 \\ &= \sum_{i=1}^N \left(g_{m,i} w_{q(\mathbf{x}_i)} + \frac{1}{2} h_{m,i} w_{q(\mathbf{x}_i)}^2 \right) + \gamma T + \frac{1}{2} \lambda \sum_{t=1}^T w_t^2 \\ &= \sum_{t=1}^T \left[\sum_{i \in I_t} g_{m,i} w_t + \frac{1}{2} \sum_{i \in I_t} h_{m,i} w_t^2 \right] + \frac{1}{2} \lambda \sum_{t=1}^T w_t^2 + \gamma T \\ &= \sum_{t=1}^T \left[\underbrace{\sum_{i \in I_t} g_{m,i}}_{G_t} w_t + \frac{1}{2} \left(\underbrace{\sum_{i \in I_t} h_{m,i}}_{H_t} + \lambda \right) w_t^2 \right] + \gamma T \\ &= \sum_{t=1}^T \left[G_t w_t + \frac{1}{2} (H_t + \lambda) w_t^2 \right] + \gamma T \end{aligned}$$

T 个独立的二次函数之和

» 目标函数

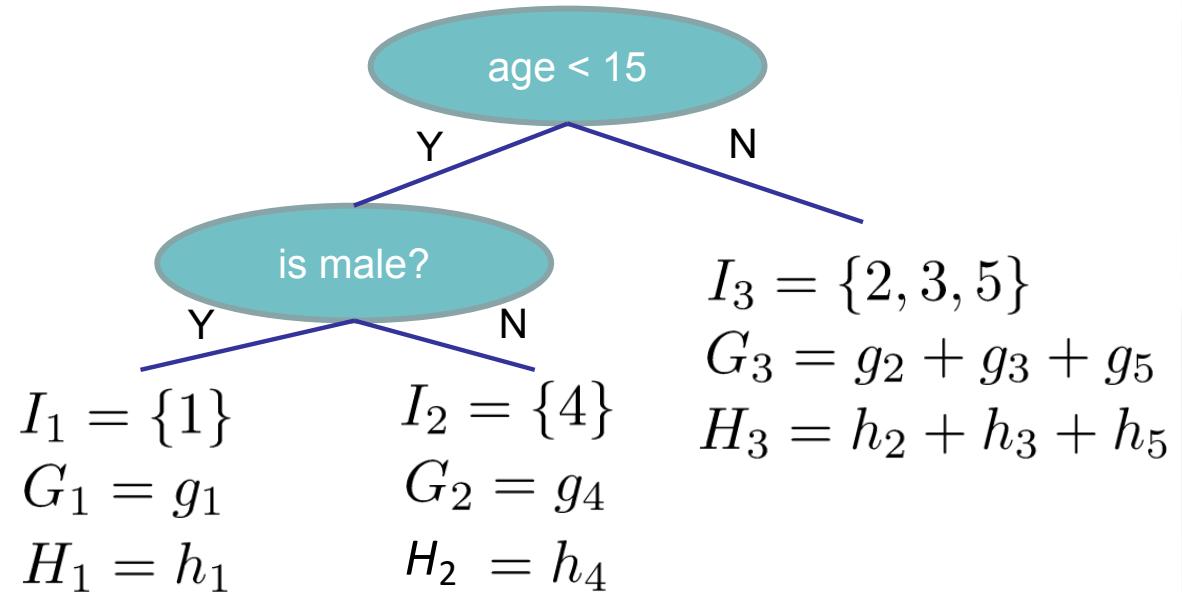
- 假设我们已经知道树的结构 q ,
- $J(f) = \sum_{t=1}^T \left[G_t w_t + \frac{1}{2}(H_t + \lambda)w_t^2 \right] + \gamma T$
- 则由 $\frac{\partial J(f)}{\partial w_t} = G_t + (H_t + \lambda)w_t = 0$
- 得到最佳的 w : $w_t = -\frac{G_t}{H_t + \lambda}$
- 以及最佳的 w 对应的目标函数，可视为树的分数：
- $J(f) = -\frac{1}{2} \sum_{t=1}^T \left[\frac{G_t^2}{H_t + \lambda} \right] + \gamma T$

分数越小的树越好！

➤ 例：树的分数

Instance index gradient statistics

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



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

The smaller the score is, the better the structure is

➤ 回归树的学习策略

■当树的结构确定时，我们前面已经推导出其最优的叶节点分数以及对应的最小损失值。

■如何确定树的结构？

- 暴力枚举所有可能的树结构，选择损失值最小的 —— NP难问题

- 贪心法：每次尝试分裂一个叶节点，计算分裂前后的增益，选择增益最大的分裂

■如何度量分裂的增益？

➤ 分裂的增益度量

- ID3算法采用信息增益
- C4.5算法采用信息增益比
- CART采用Gini系数
- XGBoost : $J(f) = -\frac{1}{2} \sum_{t=1}^T \left[\frac{G_t^2}{H_t + \lambda} \right] + \gamma T$

➤ 增益

- 实践中，我们贪婪的增加树的叶子结点数目：
- (1) 从深度为0的树开始
- (2) 对于树的每个叶子节点，尝试增加一个分裂点：
 - 令 I_L 和 I_R 分别表示加入分裂点后左右叶子结点的样本集合， $I = I_L \cup I_R$,
 - $G_L = \sum_{i \in I_L} g_{m,i}$, $G_R = \sum_{i \in I_R} g_{m,i}$, $H_L = \sum_{i \in I_L} h_{m,i}$, $H_R = \sum_{i \in I_R} h_{m,i}$,
 - 则该分裂的增益为增加分裂点后目标函数的变化：

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

➤ 精确贪心算法

■ 对每一个结点，穷举所有特征、所有可能的分裂点

- 对每个特征，通过特征值将实例进行排序
- 运用线性扫描来寻找该特征的最优分裂点（增益最大的分裂）
- 对所有特征，采用最佳分裂点

■ 深度为 k 的树的时间复杂度：

- 对于一层排序，需要时间 $N \log(N)$ ， N 为样本数目
- 由于有 D 个特征， k 层，所以为 $kDN \log(N)$

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** D **do** (对每维特征 k)

$G_L \leftarrow 0, H_L \leftarrow 0$

for j in $sorted(I, by x_{jk})$ **do** (以第 k 维特征为分裂特征，第 j 个样本 x_{jk} 的值为阈值)

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

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

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

end

end

Output: Split with max score

首先，对所有特征都按照特征的数值进行预排序。

其次，在遍历分割点的时候用 $O(\#data)$ 的代价找到一个特征上的最好分割点。

最后，找到一个特征的分割点后，将数据分裂成左右子节点。

➤ 精确贪心算法

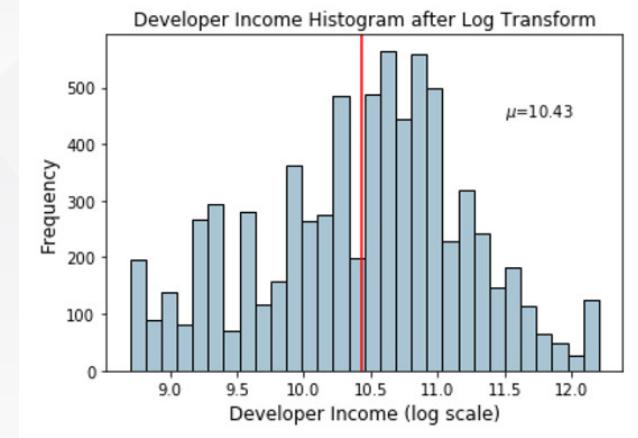
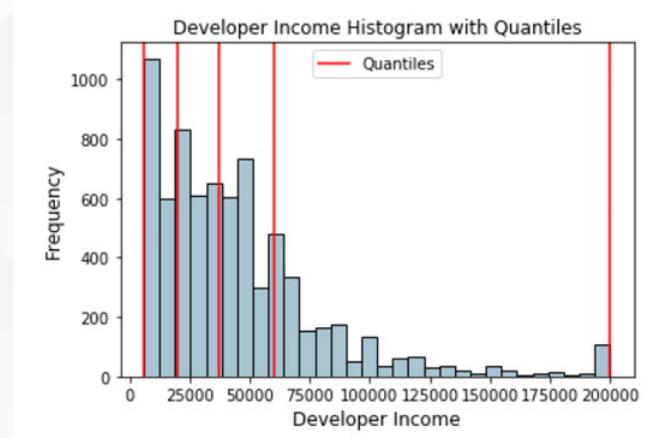
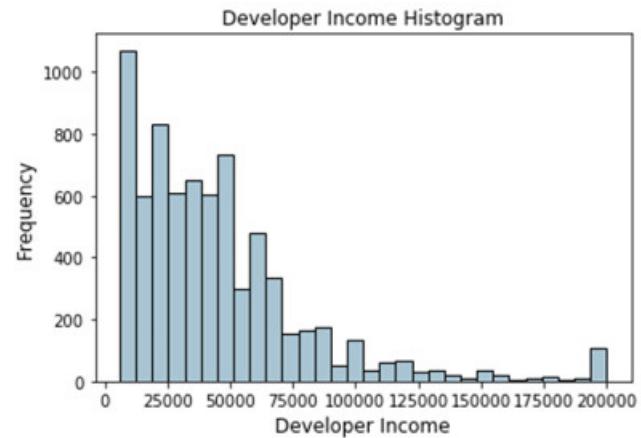
■ 预排序算法能精确地找到分割点，但缺点也很明显：

- 空间消耗大：算法需要保存数据的特征值，还保存了特征排序的结果，需要的内存约是训练数据的两倍($2N \times D \times 4$ 个字节)，其中 N 为样本数目， D 为特征维数，特征值和排序索引需要用32位来保存。
 - Histogram 算法：只需要 $N \times D \times 1$ 个字节的内存，仅为预排序算法的1/8。因为 histogram 算法仅需要存储特征离散化后的数值，256个bin索引用 8个bit (1个字节) 存储。
- 时间上开销大：可能的分割点多，每个分割点都需要进行分裂增益的计算。
- 对cache优化不友好：在预排序后，特征对梯度的访问是随机的，且不同的特征访问的顺序不一样，无法对cache进行优化。同时，在每一层长树的时候，需要随机访问一个行索引到叶子索引的数组，并且不同特征访问的顺序也不一样，也会造成较大的cache miss。

➤ 基于直方图的近似算法

- 当数据太多不能装载到内存时，不能进行精确搜索分裂，只能近似
 - 根据特征分布的百分位数（或一般的直方图），提出特征的一些候选分裂点
 - 将连续特征值映射到桶里（候选点对应的分裂），然后根据桶里样本的统计量，从这些候选中选择最佳分裂点
- 根据候选提出的时间，分为
 - 全局近似：在构造树的初始阶段提出所有的候选分裂点，然后对各个层次采用相同的候选
 提出候选的次数少，但每次的候选数目多（因为候选不更新）
 - 局部近似：在每次分裂重新提出候选
 对层次较深的树更适合

➤ 对长尾分布，直方图可能利用不充分



Log变换



Algorithm 2: Approximate Algorithm for Split Finding

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

$G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$

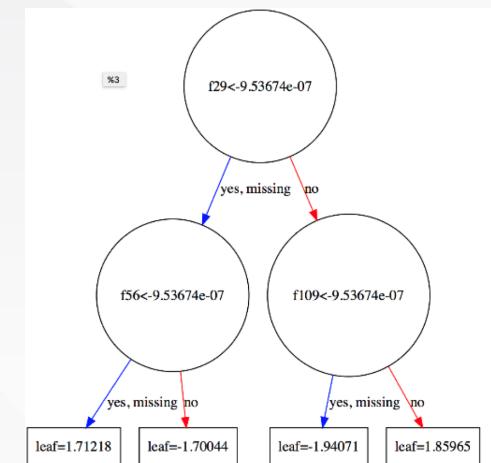
$H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$

end

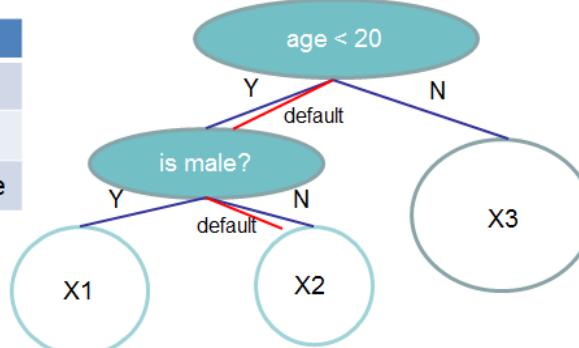
Follow same step as in previous section to find max score only among proposed splits.

➤ 稀疏特征

- 在实际任务中，极有可能遇到稀疏特征
 - 缺失数据
 - 人工设计的特征：如独热（one-hot）编码
- XGBoost：在树的每个结点设置一个缺省方向



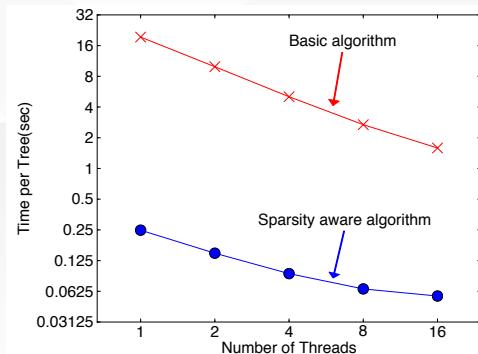
Data		
Example	Age	Gender
X1	?	male
X2	15	?
X3	25	female



➤ 稀疏特征

- 统一的稀疏特征处理方案：将稀疏特征视为缺失值
- 最佳缺省方向确定：
 - 只访问非缺失数据
 - 计算复杂度与非缺失数据数目线性相关

在数据高度稀疏的
Allstate-10K 数据集上
稀疏算法比基本算法快
近50倍



Algorithm 3: Sparsity-aware Split Finding

```
Input:  $I$ , instance set of current node
Input:  $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$ 
Input  $D$ , feature dimension
Also applies to the approximate setting, only collect
statistics of non-missing entries into buckets
 $gain \leftarrow 0$ 
 $G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$ 
for  $k = 1$  to  $D$  do
    // enumerate missing value goto right
     $G_L \leftarrow 0, H_L \leftarrow 0$           (假设缺省方向为右边)
    for  $j$  in sorted( $I_k$ , ascent order by  $x_{jk}$ ) 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
    // enumerate missing value goto left
     $G_R \leftarrow 0, H_R \leftarrow 0$           (假设缺省方向为左边)
    for  $j$  in sorted( $I_k$ , descent order by  $x_{jk}$ ) do
         $G_R \leftarrow G_R + g_j, H_R \leftarrow H_R + h_j$ 
         $G_L \leftarrow G - G_R, H_L \leftarrow H - H_R$ 
        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 and default directions with max gain
```

Outline

- 模型性能评价

 - No Free Lunch Theorems

 - Occam剃刀原理

 - 偏差-方差折中

- Bagging

 - 随机森林

- Boosting

 - AdaBoost

 - Gradient Boosting Decision Tree
(GBDT)

 - XGBoost

 - LightGBM

- Stacking

➤ LightGBM : Light Gradient Boosting

Machine

- LightGBM 是Microsoft开发的一个GBDT算法框架，支持高效率的并行训练，并且具有以下优点：
 - 更快的训练速度
 - 更低的内存消耗
 - 分布式支持，可以快速处理海量数据

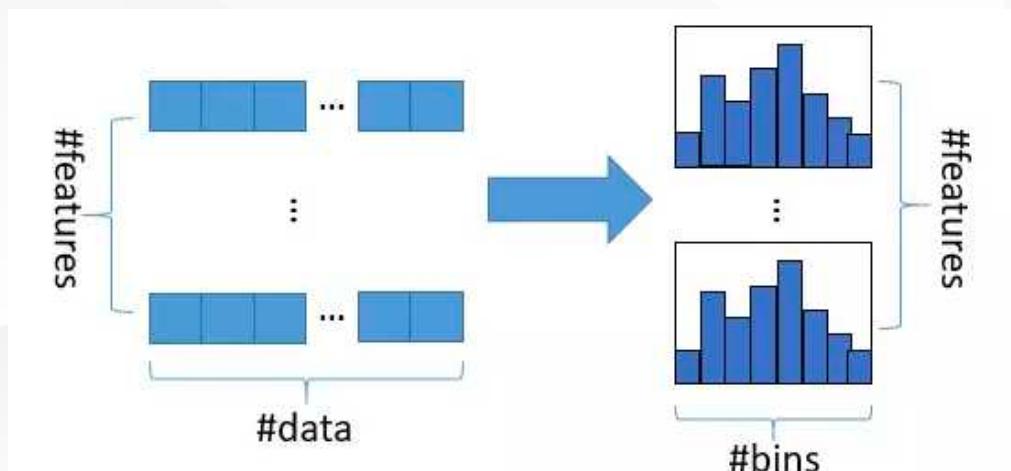
<https://github.com/Microsoft/LightGBM>

➤ LightGBM的优化

- 基于直方图的分裂算法
- 直方图加速构造
- 带深度限制的Leaf-wise的叶子生长策略
- 直接支持类别型特征(Categorical Feature)
- Cache命中率优化
- 多线程优化

➤ 基于Histogram的决策树算法

- 把特征值离散化成 K 个整数
- 构造一个箱子（bin）数为 K 的直方图
- 遍历数据时，根据特征离散化后的值作为索引在直方图中累积统计量
- 根据直方图的离散值，遍历寻找最优的分割点



相当于数值特征离散化 / 分组：一个bin为一组

➤ Histogram 算法的计算优势

- Histogram 算法可大幅减少计算分割点增益的次数
 - 对于一个特征，预排序需要对每一个不同特征值都计算一次分割增益，而 histogram 只需要计算 #bin 次。

➤ Histogram 算法的缺点

- Histogram 算法不能找到精确的分割点，训练误差没有预排序算法好。
- 但从实验结果来看，Histogram 算法在测试集的误差预排序算法差异并不是很大，甚至有时候效果更好。
 - 实际上可能决策树对于分割点的精确程度并不太敏感，而且较“粗”的分割点也自带正则化的效果。

➤ Histogram 做差加速

- Histogram做差加速：一个叶子的直方图可以由它的父亲节点的直方图与它兄弟的直方图做差得到。
- 通常构造直方图，需要遍历该叶子上的所有数据
- 直方图做差仅需遍历直方图的 K 个桶。利用这个方法，LightGBM可以在构造一个叶子的直方图后，可以用非常微小的代价得到它兄弟叶子的直方图，在速度上可以提升一倍。

➤ 直方图加速计算

■ 直方图：遍历所有叶子结点的所有样本

- 时间复杂度： $O(\#data \times \#feature)$

■ 加速加速

- 减小样本数目： $\#data \downarrow$ （基于梯度的单边采样）
- 减小特征数目： $\#feature \downarrow$ （互斥特征捆绑）

➤ 直方图加速：基于梯度的单边采样

- 减少样本数：抛弃那些不太重要的样本
- 梯度：样本重要性度量
 - 梯度很小：该样本的训练误差很小，或者说该样本已经能被模型很好表示
- 但直接抛弃梯度很小的样本会改变样训练集的分布，使得模型准确率下降 \longrightarrow 根据样本的梯度的绝对值进行采样，减少实际访问样本数

➤ 直方图加速：基于梯度的单边采样

■ 基于梯度的单边采样 (Gradient-based One-Side Sampling , GOSS)

- 1 . 根据梯度的绝对值将样本进行降序排序；
- 2 . 保留所有的梯度较大的样本 \mathcal{A} ：选择前 $a \times 100\%$ 的样本；
- 3 . 随机采样梯度小的样本：剩下 $(1 - a) \times 100\%$ 的数据中，随机抽取 $b \times 100\%$ 的数据，这些样本称为 \mathcal{B} ；
- 4 . 为了抵消对数据分布的影响，在计算增益时，对小梯度数扩大常量倍：对 \mathcal{B} 中样本的梯度 $(1 - a)/b$ 倍。

➤ 直方图加速：互斥特征捆绑

- 高维数据通常是非常稀疏的，而且很多特征是互斥的（两个或多个特征列不会同时为0）
- 互斥特征捆绑（Exclusive Feature Bundling, EFB）：将互斥特征捆绑成一束 \longrightarrow 降低特征维度，#feature \rightarrow #bundle。
- 要完成这个任务，需解决两个问题：
 - 1 . 哪些特征可以捆绑在一起，组成一束；
 - 2 . 如何构建特征束，从而实现特征降维。

➤ 直方图加速：互斥特征捆绑

■ 特征捆绑：图着色问题

■ 图 \mathcal{G} ：

- 顶点：特征
- 边：两个特征之间的总冲突值为边的权重
- 着色：相邻的顶点涂上不同的颜色（不互斥的顶点放在不同的特征束），同时总的颜色最少越好（总的特征束越少越好）

➤ 直方图加速：互斥特征捆绑

■ 特征捆绑：图着色问题

■ 1. 初始化：

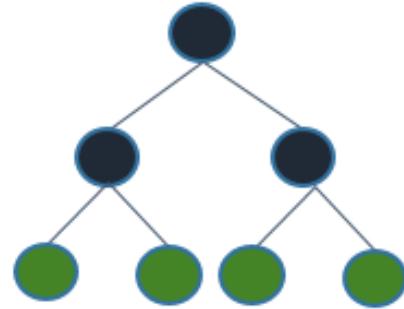
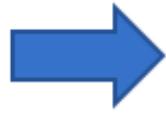
- 对计算结点的度，根据度对特征进行排序
- 特征束为空

■ 2. 对每个特征（根据度的大小）

- 计算特征与已有束的冲突程度。如冲突小，就加入到该计算束；如果与所有现有的束的冲突都较大，则生成一个新束；

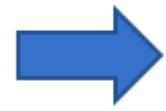
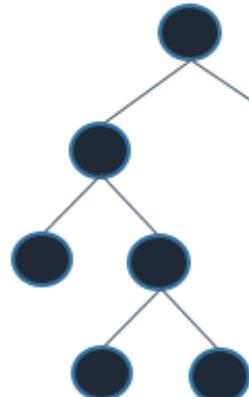
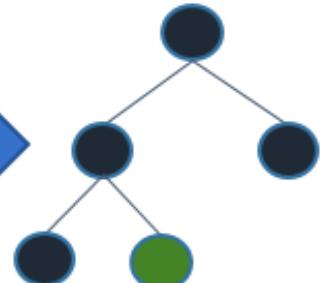
➤ 带有深度限制的按叶子生长 (leaf-wise) 算法

- Level-wise 过一次数据可以同时分裂同一层的叶子，容易进行多线程优化，不容易过拟合
- 但 Level-wise 不加区分的对待同一层的叶子，带来了很多没必要的开销，比较低效（分裂增益较低，没必要进行搜索和分裂）
- Leaf-wise 每次从当前所有叶子中，找到分裂增益最大(一般也是数据量最大)的一个叶子进行分裂。同 level-wise 相比，在分裂次数相同的情况下，leaf-wise 可以降低更多的误差，得到更好的精度。
- Leaf-wise 的缺点是可能会长出比较深的决策树，产生过拟合。因此 LightGBM 在 leaf-wise 之上增加了一个最大深度的限制，在保证高效率的同时防止过拟合。



.....

Level-wise tree growth



.....

Leaf-wise tree growth

➤ 直接支持类别型特征

- 实际上大多数机器学习工具都无法直接支持类别特征，一般需要把类别特征，转化到多维的0/1特征，降低了空间和时间的效率。
- LightGBM优化了对类别特征的支持，可以直接输入类别特征，不需要额外的0/1展开。并在决策树算法上增加了类别特征的决策规则。

➤ XGBoost资源

- XGBoost官方文档：<https://xgboost.readthedocs.io/en/latest/>
 - Python API：http://xgboost.readthedocs.io/en/latest/python/python_api.html
 - 参数说明：<http://xgboost.readthedocs.io/en/latest/parameter.html#general-parameters>
- Github：<https://github.com/dmlc/xgboost>
 - 很多有用的资源：
<https://github.com/dmlc/xgboost/blob/master/demo/README.md>
 - GPU加速：
https://github.com/dmlc/xgboost/blob/master/plugin/updater_gpu/README.md
 - 示例代码：<https://github.com/dmlc/xgboost/tree/master/demo/guide-python>
- XGBoost原理：XGBoost: A Scalable Tree Boosting System
 - <https://arxiv.org/abs/1603.02754>

➤ XGBoost资源

- XGBoost参数调优：
 - <https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/>
 - 中文版：<http://blog.csdn.net/u010657489/article/details/51952785>
 - http://blog.csdn.net/han_xiaoyang/article/details/52665396
 - [使用sklearn进行集成学习——实践](#)
 - <https://www.cnblogs.com/jasonfreak/p/5720137.html>
- Owen Zhang, Winning Data Science Competitions
 - https://www.slideshare.net/OwenZhang2/tips-for-data-science-competitions?from_action=save
- XGBoost User Group：
 - <https://groups.google.com/forum/#!forum/xgboost-user/>

➤ LightGBM资源

- LightGBM文档：
 - <http://lightgbm.readthedocs.io/en/latest/index.html>
 - 中文参考：<http://lightgbm.apache.org/cn/latest/Quick-Start.html>
- LightGBM参数说明：
 - <http://lightgbm.readthedocs.io/en/latest/Parameters.html>
- 从结构到性能，一文概述XGBoost、LightGBM和CatBoost的同与不同
 - <https://www.jiqizhixin.com/articles/2018-03-18-4>
 - <https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db>

➤ 例：Otto商品分类

- XGBoost
- LightGBM

The End

➤ XGBoost可视为Newton Boosting

参数空间

$$\theta^t = \theta^{t-1} + \theta_t$$

第t次迭代后的参数 第t-1次迭代后的参数 第t次迭代的参数增量

$$\theta_t = -H_t^{-1}g_t$$

与梯度下降法唯一不同的就是参数增量

$$\theta = \sum_{t=0}^T \theta_t$$

最终参数等于每次迭代的增量的累加和，

θ_0 为初值

函数空间

$$f^t(x) = f^{t-1}(x) + f_t(x)$$

第t次迭代后的函数 第t-1次迭代后的函数 第t次迭代的函数增量

$$f_t(x) = -\frac{g_t(x)}{h_t(x)}$$

$$F(x) = \sum_{t=0}^T f_t(x)$$

最终函数等于每次迭代的增量的累加和，

$f_0(x)$ 为模型初始值，通常为常数