第三讲：Boosting原理

——重回XGBoost

AI100学院

2017年6月

Roadmap

- Boosting
- Gradient Boosting
- XGBoost

一、 Boosting

▶ 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：性能比随机猜测好（层数不深的CART是一个好选择）

- 亦可视为一种自适应基模型：

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

  - 其中$\phi_m(\mathbf{x})$ 为基函数 / 弱学习器。

AdaBoost

- 样本权重 / “过滤“
  - 没有先验知识的情况下，初始的分布为等概分布，即训练集如果有N个样本，每个样本的分布概率为 $1/N$
  - 每次循环后提高误分样本的分布概率，误分样本在训练集中所占权重增大，使得下一次循环的弱学习器能够集中力量对这些误分样本进行判断
- 模型组合: 弱学习器线性组合
  - 准确率越高的弱学习机权重越高
  - $f(\mathbf{x}) = \operatorname{sgn}(\sum_{m=1}^{M} \alpha_m \phi_m(\mathbf{x}))$

AdaBoost M1算法

- 给定训练集 :$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, 其中.$y_i \in \{1,-1\}$ 表示$\mathbf{x}_i$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$\phi_m(\mathbf{x})$
- 计算该弱分类器在分布w_m上的误差：$\varepsilon_m = \frac{\sum_{i=1}^{N} w_{m,i}\, \mathbb{I}(\phi_m(\mathbf{x}_i) \neq \boldsymbol{y}_i)}{\sum_{i=1}^{N} w_{m,i}}$
- 计算该弱分类器的权重：$\alpha_m = \frac{1}{2} log \frac{1-\varepsilon_m}{\varepsilon_m}$
- 更新训练样本的分布:$w_{m+1,i} = \frac{w_{m,i} exp(-\alpha_m \boldsymbol{y}_i \phi_m(\mathbf{x}_i))}{Z_m}$

  其中Z_m为归一化常数，使得w_{m+1}是一个分布。
- 最后的强分类器为：$f(\mathbf{x}) = \mathrm{sgn}(\sum_{m=1}^{M} \alpha_m \phi_m(\mathbf{x}))$

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

►证明

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

- 1. 对w_{M+1} 进行迭代展开

$$w_{M+1,i} = w_{M,i} \frac{exp\left(-\alpha_M y_i \phi_M(\mathbf{x}_i)\right)}{Z_M} = w_{1,i} \frac{exp(-y_i \boxed{\sum_{m=1}^{M} \alpha_m \phi_m(\mathbf{x}_i)})}{\prod_{m=1}^{M} Z_m}$$

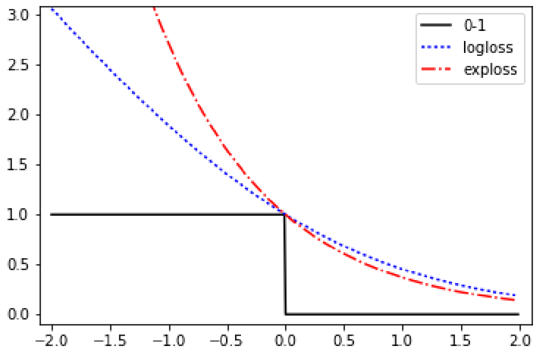
$$= w_{1,i} \frac{exp\left(-y_i f(\mathbf{x}_i)\right)}{\prod_{m=1}^{M} Z_m}$$

- 由于w_{M+1}是一个分布，所以$\sum_{i=1}^{N} w_{M+1,i} = 1$
- 所以$\prod_{m=1}^{M} Z_m = w_{1,i} \sum_{i=1}^{N} exp\left(-y_i f(\mathbf{x}_i)\right) = \frac{1}{N} \sum_{i=1}^{N} exp\left(-y_i f(\mathbf{x}_i)\right)$。

►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：$ERR_{train}(f(\mathbf{x})) = \frac{1}{N}\left|\{i: y_i \neq sgn(f(\mathbf{x}_i))\}\right|$



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

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

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

►证明（cont.）

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

$$\boxed{Z_m = \sum_{i=1}^{N} w_{m,i} exp\left(-\alpha_m y_i \phi_m(\mathbf{x}_i)\right)}$$

- 3. 证明弱分类器权重为$\alpha_m = \frac{1}{2} log \frac{1-\varepsilon_m}{\varepsilon_m}$
- 问题：给定弱分类器的集合$\Delta = \{\phi_1(\mathbf{x}), \ldots, \phi_M(\mathbf{x})\}$，确定弱分类器$\phi_m$及其权重$\alpha_m$
- $(\phi, \alpha)^* = \underset{\phi,\alpha}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} exp\left(-y_i f(\mathbf{x}_i)\right) = \underset{\phi,\alpha}{\operatorname{argmin}} \prod_{m=1}^{M} Z_m$
- 具体实现时，首先选一个错误率最小的弱分类器ϕ_m，然后确定其权重α_m：
- $\frac{\partial Z_m}{\partial \alpha_m} = \frac{\partial \sum_{i=1}^{N} w_{m,i} exp\left(-\alpha_m y_i \phi_m(\mathbf{x}_i)\right)}{\partial \alpha_m}$
- $= -\sum_{i=1}^{N} w_{m,i} y_i \phi_m(\mathbf{x}_i) exp\left(-\alpha_m y_i \phi_m(\mathbf{x}_i)\right)$

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

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

- $\frac{\partial Z_m}{\partial \alpha_m} = -\sum_{i=1}^{N} w_{m,i} y_i \phi_m(\mathbf{x}_i) exp\left(-\alpha_m y_i \phi_m(\mathbf{x}_i)\right)$
- $= \begin{cases} -\sum_{\mathbf{x}_i \in A} w_{m,i} exp(-\alpha_m) & if\ \mathbf{x}_i \in A, A = \{\mathbf{x}_i : y_i \phi_m(\mathbf{x}_i) = 1\} \text{ 分类正确的样本集合} \\ \sum_{\mathbf{x}_i \in \bar{A}} w_{m,i} exp(\alpha_m) & if\ \mathbf{x}_i \in \bar{A}, \bar{A} = \{\mathbf{x}_i : y_i \phi_m(\mathbf{x}_i) = -1\} \text{ 分类错误的样本集合} \end{cases}$
- $\frac{\partial Z_m}{\partial \alpha_m} = 0 \ ==> \sum_{\mathbf{x}_i \in A} w_{m,i} exp(-\alpha_m) = \sum_{\mathbf{x}_i \in \bar{A}} w_{m,i} exp(\alpha_m)$ 两边同乘以$exp(\alpha_m)$
- $\underbrace{\boxed{\sum_{\mathbf{x}_i \in A} w_{m,i}}}_{1-\varepsilon_m} = exp(2\alpha_m) \underbrace{\boxed{\sum_{\mathbf{x}_i \in \bar{A}} w_{m,i}}}_{\varepsilon_m} \Rightarrow 1 - \varepsilon_m = \varepsilon_m exp(2\alpha_m)$

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

错误率小的弱分类器的权重更大

二、Gradient Boosting

►前向逐步递增

Forward stagewise additive modeling

- 还可以从另外一个角度来看AdaBoost：前向逐步递增
  - 要找到最优的f很难 → 每次递增

- 损失函数：$L(f(\mathbf{x}),y)$
- 目标函数：$\min_{f} \frac{1}{N}\sum_{i=1}^{N} L(f(\mathbf{x}_i), y_i)$
- 前向逐步递增
  - 初始化：$f_0(\mathbf{x}) = \arg\min_{f} \frac{1}{N}\sum_{i=1}^{N} L(f(\mathbf{x}_i), y_i)$
  - 递增：$(\beta_m, \phi_m) = \arg\min_{\beta,\phi} \frac{1}{N}\sum_{i=1}^{N} L(f_{m-1}(\mathbf{x}_i) + \beta\phi(\mathbf{x}_i), y_i)$

$$f_m(\mathbf{x}_i) = f_{m-1}(\mathbf{x}_i) + \beta_m\phi_m(\mathbf{x}_i)$$ 前向逐步递增

► AdaBoost as前向逐步递增

- 将指数损失$L(f(\mathbf{x}), y) = exp(-yf(\mathbf{x}))$代入，
- 第m步，最小化
- $L_m = \sum_{i=1}^{\mathrm{N}} L(f(\mathbf{x}_i), y_i)$
- $= \sum_{i=1}^{\mathrm{N}} exp\left(-y_i\left(f_{m-1}(\mathbf{x}_i) + \beta\phi(\mathbf{x}_i)\right)\right)$
- $= \sum_{i=1}^{\mathrm{N}} \underbrace{exp\left(-y_i f_{m-1}(\mathbf{x}_i)\right)}_{w_{m,i}} exp\left(-y_i\beta\phi(\mathbf{x}_i)\right)$

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

- $= \left(e^{-\beta} \sum_{y_i=\phi(\mathbf{x}_i)} w_{m,i} + e^{\beta} \sum_{y_i\neq\phi(\mathbf{x}_i)} w_{m,i}\right)$
- $= \left(\left(e^{\beta} - e^{-\beta}\right) \sum_{i=1}^{N} w_{m,i}\, \mathbb{I}\left(y_i \neq \phi(\mathbf{x}_i)\right) + \left(e^{-\beta}\right) \sum_{i=1}^{N} w_{m,i}\right)$

AdaBoost as前向逐步递增(cont.)

- $L_m = \left(\left(e^{\beta} - e^{-\beta}\right)\sum_{i=1}^{N} w_{m,i}\,\mathbb{I}\left(y_i \neq \phi(\mathbf{x}_i)\right) + \left(e^{-\beta}\right)\sum_{i=1}^{N} w_{m,i}\right)$
- 得到$\phi_m(\mathbf{x}) = \underset{\phi}{\operatorname{argmin}} \sum_{i=1}^{N} w_{m,i}\mathbb{I}\left(y_i \neq \phi(\mathbf{x}_i)\right)$，即最佳的$\phi_m$为错误率最小的弱分类器。

推导过程同之前的$\frac{\partial Z_m}{\partial \alpha_m}$推导

- 将ϕ_m代入L_m，并令$\frac{\partial \mathrm{L}_m}{\partial \beta_m} = 0 \;\; ==> \beta_m = \frac{1}{2} log \frac{1-\varepsilon_m}{\varepsilon_m}$，
- 其中错误率$\varepsilon_m = \sum_{i=1}^{N} w_{m,i}\,\mathbb{I}\left(y_i \neq \phi_m(\mathbf{x}_i)\right) / \sum_{i=1}^{N} w_{m,i}$。

►前向逐步递增—其他损失函数

- 指数损失对outliers比较敏感，而且也不是任何二值变量y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：$L(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2$
  - 初始化： $f_0(\mathbf{x}) = \bar{y}$

- 在第m步，损失函数的形式为
- $L(f_{m-1}(\mathbf{x}_i) + \beta_m \phi_m(\mathbf{x}_i), y_i) = (f_{m-1}(\mathbf{x}_i) + \beta_m \phi_m(\mathbf{x}_i) - y_i)^2$
- $= \left(-(y_i - f_{m-1}(\mathbf{x}_i)) + \beta_m \phi_m(\mathbf{x}_i)\right)^2 = \left(r_{m,i} - \beta_m \phi_m(\mathbf{x}_i)\right)^2$
- 其中 $r_{m,i} = f_{m-1}(\mathbf{x}_i) - y_i$
- 不失一般性，假设β=1，因此用弱学习器来预测残差 $r_{m,i}$ ，称为L2Boosting。

Shrinkage

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

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

- 其中$0 < \eta < 1$ ，通常取一个较小的值，如$\eta = 0.1$。

- 较小的收缩因子通常意味着更多弱分学习器。

► Boosting as 函数梯度下降

- 前向逐步递增建模部分我们讨论了不同损失函数对应的boosting算法，其实可推导更一般的模型：gradient boosting

- 目标：$\min_{\mathbf{f}} \frac{1}{N} \sum_{i=1}^{N} L(f(\mathbf{x}_i), y_i)$，
- 其中$\mathbf{f} = \{f(\mathbf{x}_1)，\ldots, f(\mathbf{x}_N)\}$ 为“参数”

- 优化：逐步梯度下降（stagewise，gradient boosting）

►Gradient Boosting

- 目标：$\min_{\mathbf{f}} \frac{1}{N} \sum_{i=1}^{\mathrm{N}} L(f(\mathbf{x}_i), y_i)$，
- 在第m步，$\mathbf{f} = \mathbf{f}_{\mathrm{m-1}}$
- 梯度为：$g_{m,i} = \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{\mathbf{f}=\mathbf{f}_{\mathrm{m-1}}}$
- 然后更新：$\mathbf{f} = \mathbf{f}_{\mathrm{m-1}} - \beta_m \mathbf{g}_m$, 其中$\mathbf{g}_m = \left(g_{m,1}, \dots, g_{m,N}\right)^T$
- 其中$\beta_m = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{\mathrm{N}} L\left(f_{\mathrm{m-1}}(\mathbf{x}_i) - \beta g_{m,i}, y_i\right)$为步长

► Gradient Boosting

- 上述算法只在N个数据点优化**f**
- 将上述算法修改为用一个弱学习器近似负梯度，即
- $\phi_m = \underset{\phi}{\operatorname{argmin}} \sum_{i=1}^{N} \left(-g_{m,i} - \phi(\mathbf{x}_i)\right)^2$
- 对上述算法，损失函数取L2，得到L2Boosting
- 该一般框架对很多损失函数都适用
  - Logistic损失
  - Huber损失
  - …

► Gradient Boosting Algorithm

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

►Scikit-learn中的GBM

- 分类器：GradientBoostingClassifier

- sklearn.ensemble.**GradientBoostingClassifier**(*loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_split=1e-07, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto'*)
  – 由于弱学习器为CART，所以很多参数与树模型的参数相同
  – 额外的参数（红色）主要关于弱学习器组合

参数	说明
loss	待优化的目标函数，‘deviance’表示采用logistic损失，输出概率值；‘exponential’表示采用指数损失。缺省 ‘deviance’
learning_rate	学习率或收缩因子。学习率和迭代次数 / 弱分类器数目n_estimators相关。 缺省：0.1
n_estimators	当数 / 弱分类器数目. 缺省:100
subsample	学习单个弱学习器的样本比例。缺省为：1.0

三、XGBoost

XGBoost

- XGBoost：eXtreme Gradient Boosting
  - 可自定义损失函数：损失函数采用二阶近似
  - 规范化的正则项：叶子节点数目、叶子结点的分数
  - 建树与剪枝：先建完全树后剪枝
    - 支持分裂点近似搜索
    - 稀疏特征处理
    - 缺失值处理
  - 特征重要性与特征选择
  - 并行计算
  - 内存缓存

►损失函数的二阶近似

- Gradient Boosting算法虽然对常见损失函数适用，但除了L2损失函数，其他损失函数推导还是比较复杂

- XGBoost：对损失函数用二阶Taylor展开近似

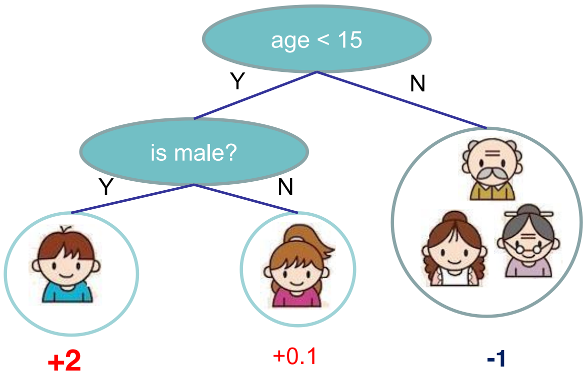
►损失函数的二阶近似

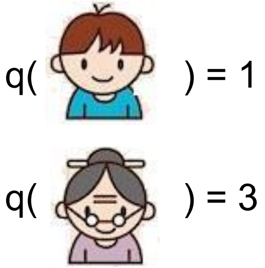
$$f(x+\Delta x) \cong f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$$

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

树

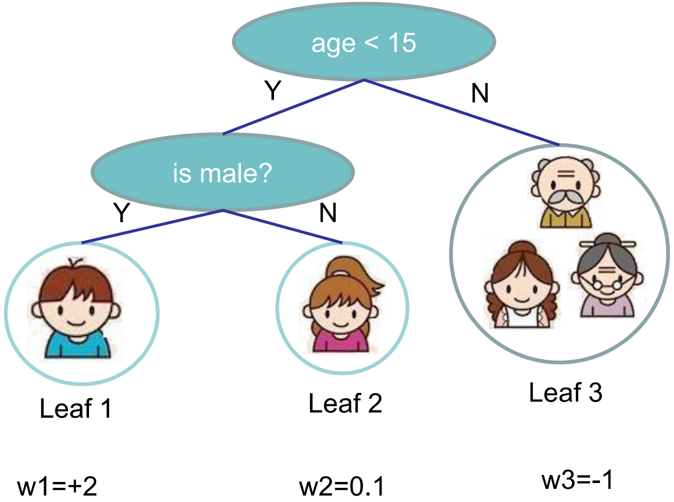
- Recall：树的定义：把树拆分成结构部分q和叶子分数部分w
- $\phi(\mathbf{x}) = w_{q(\mathbf{x})}$ ，$\mathbf{w} \in R^T, q\colon R^D \rightarrow \{1, \dots, T\}$
  - 结构函数q：把输入映射到叶子的索引号
  - w：给出每个索引号对应的叶子的分数
  - T为树中叶子结点的数目，D为特征维数





树的复杂度

- 树的复杂度定义为（不是唯一的定义方式）
- $\Omega(\phi(\mathbf{x})) = \gamma T + \frac{1}{2}\lambda \sum_{t=1}^{T} w_t^2$
  - 叶子节点的数目、叶子节点分数的L2正则



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(\underbrace{4 + 0.01 + 1}_{2^2+0.1^2+(-1)^2})$$

目标函数

- 令每个叶子t上的样本集合为$I_t = \{i|q(\mathbf{x}_i) = t\}$
- $J(\boldsymbol{\theta}) = \sum_{i=1}^{N} L(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) + \Omega(\boldsymbol{\theta})$
- $\cong \sum_{i=1}^{N} g_{m,i}\phi(\mathbf{x}_i) + \frac{1}{2}h_{m,i}\phi(\mathbf{x}_i)^2 + \gamma T + \frac{1}{2}\lambda \sum_{t=1}^{T} w_t^2$
- $= \sum_{i=1}^{N} g_{m,i}w_{q(\mathbf{x}_i)} + \frac{1}{2}h_{m,i}{w_{q(\mathbf{x}_i)}}^2 + \gamma T + \frac{1}{2}\lambda \sum_{t=1}^{T} w_t^2$
- $= \sum_{t=1}^{T} \left[\sum_{i\in I_t} g_{m,i}\, \mathrm{w}_t + \frac{1}{2}\sum_{i\in I_t} h_{m,i}\, {\mathrm{w}_t}^2 + \frac{1}{2}\lambda \sum_{t=1}^{T} w_t^2\right] + \gamma T$
- $= \sum_{t=1}^{T} \left[\underbrace{\sum_{i\in I_t} g_{m,i}}_{G_t} \mathrm{w}_t + \frac{1}{2}\left(\underbrace{\sum_{i\in I_t} h_{m,i}}_{H_t} + \lambda\right) w_t^2\right] + \gamma T$
- $= \sum_{t=1}^{T} \left[G_t \mathrm{w}_t + \frac{1}{2}(H_t + \lambda)w_t^2\right] + \gamma T$

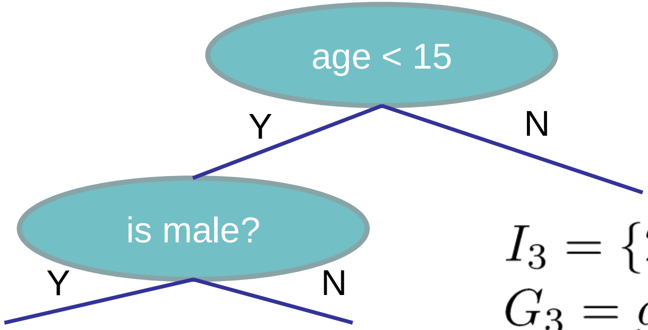
T个独立的二次函数之和

▶目标函数

- 假设我们已经知道树的结构q,
- $J(\boldsymbol{\theta}) = \sum_{t=1}^{T}\left[G_t w_t + \frac{1}{2}(H_t + \lambda)w_t^2\right] + \gamma T$
- 则由$\frac{\partial J(\boldsymbol{\theta})}{\partial w_t} = G_t + (H_t + \lambda)w_t = 0$
- 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：
- $J(\boldsymbol{\theta}) = -\frac{1}{2}\sum_{t=1}^{T}\left[\frac{G_t^2}{H_t + \lambda}\right] + \gamma T$ 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



$$I_1 = \{1\} \quad G_1 = g_1 \quad H_1 = h_1$$

$$I_2 = \{4\} \quad G_2 = g_4 \quad H_2 = h_4$$

$$I_3 = \{2, 3, 5\} \quad G_3 = g_2 + g_3 + g_5 \quad H_3 = h_2 + h_3 + h_5$$

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

The smaller the score is, the better the structure is

►建树

- 枚举可能的树结构
- 计算结构分数
- $J(\boldsymbol{\theta}) = -\frac{1}{2}\sum_{t=1}^{T}\left[\frac{G_t^2}{H_t+\lambda}\right] + \gamma T$
- 选择分数最小的树结构，并且运用最优的权重/分数

- 但是，树结构有很多可能 → 贪心算法

►建树（cont.）

- 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，$I = I_L \cup I_R$,
  - $G_L = \sum_{i\in I_L} g_{m,i}$，$G_R = \sum_{i\in I_R} g_{m,i}$，$H_L = \sum_{i\in I_L} h_{m,i}$，$H_R = \sum_{i\in I_R} h_{m,i}$，
  - 则增加分裂点后目标函数的变化为

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

- 但是：怎么找到最优的分裂点？

►建树——精确搜索算法

- 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
  - 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)

  $G_L \leftarrow 0,\ H_L \leftarrow 0$

  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do** (以第k维特征为分裂特征，第j个样本$\mathbf{x}_{jk}$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

►建树——近似搜索算法

- 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，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：在每次分裂都重新提出候选
    - 对层次较深的树更适合

Algorithm 2: Approximate Algorithm for Split Finding

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

  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.

  Proposal can be done per tree (global), or per split(local).

end

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

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

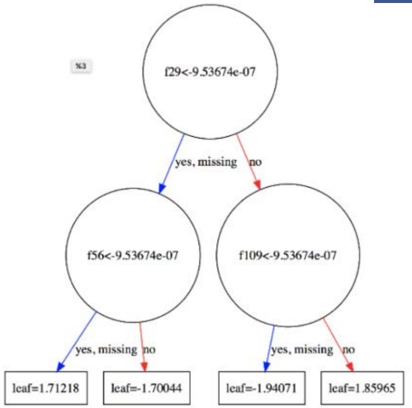
  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$

end

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

▶建树——稀疏特征

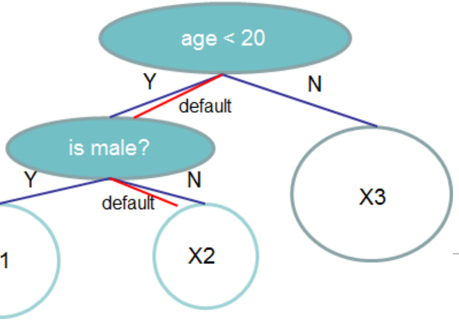
- 在实际任务中，极有可能遇到稀疏特征
  - 缺失数据
  - 人工设计的特征：如one-hot编码



- XGBoost：在树的每个结点设置一个缺省方向

Data

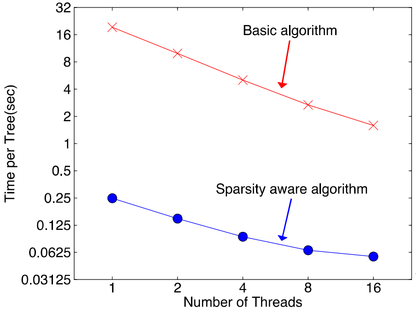
Example	Age	Gender
X1	?	male
X2	15	?
X3	25	female



建树——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* m **do** (假设缺省方向为右边)

  // *enumerate missing value goto right*

  $G_L \leftarrow 0,\ H_L \leftarrow 0$

  for j *in sorted*(I_k, ~~*ascent order by* $\mathbf{x}_{jk}$~~) **do**

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

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

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

  end (假设缺省方向为左边)

  // *enumerate missing value goto left*

  $G_R \leftarrow 0,\ H_R \leftarrow 0$

  for j *in sorted*(I_k, *descent order by* $\mathbf{x}_{jk}$) **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

剪枝和正则

- Recall 分裂的增益：

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

- 增益可能为负：引入新叶子有复杂度惩罚
- 提前终止
  - 如果出现负值，提前停止（scikit-learn中采用的策略）
  - 但被提前终止掉的分裂可能其后续的分裂会带来好处
- 过后剪枝
  - 将树分裂到最大深度，然后再基于上述增益计算剪枝
  - 有必要：在实现时还有学习率 / 收缩，给后续轮留机会，进一步防止过拟合：$f_m(\mathbf{x}_i) = f_m(\mathbf{x}_i) + \eta\phi_m(\mathbf{x}_i)$

► 再探XGBoost

- xgboost.**XGBClassifier**(*max_depth=3, learning_rate=0.1, n_estimators=100, silent=True, objective=‘binary:logistic’, nthread=-1, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None , **kwargs*)

- sklearn.ensemble.**GradientBoostingClassifier**(*loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_split=1e-07, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto'*)

参数	说明
max_depth	树的最大深度。树越深通常模型越复杂，更容易过拟合
learning_rate	学习率或收缩因子。学习率和迭代次数 / 弱分类器数目n_estimators相关。 缺省：0.1 （与直接调用xgboost的eta参数含义相同）
n_estimators	弱分类器数目. 缺省:100
slient	参数值为1时，静默模式开启，不输出任何信息
objective	待优化的目标函数，常用值有： binary:logistic 二分类的逻辑回归，返回预测的概率 multi:softmax 使用softmax的多分类器，返回预测的类别(不是概率)。 multi:softprob 和 multi:softmax参数一样，但是返回的是每个数据属于各个类别的概率。支持用户自定义目标函数
nthread	用来进行多线程控制。 如果你希望使用CPU全部的核，那就不用缺省值-1，算法会自动检测它。
booster	选择每次迭代的模型，有两种选择： gbtree：基于树的模型，为缺省值。 gbliner：线性模型
gamma	节点分裂所需的最小损失函数下降值
min_child_weight	叶子结点需要的最小样本权重（hessian）和
max_delta_step	允许的树的最大权重

参数	说明
subsample	构造每棵树的所用样本比例（样本采样比例），同GBM
colsample_bytree	构造每棵树的所用特征比例
colsample_bylevel	树在每层每个分裂的所用特征比例
reg_alpha	L1 正则的惩罚系数
reg_lambda	L2 正则的惩罚系数
scale_pos_weight	正负样本的平衡，通常用于不均衡数据
base_score	每个样本的初始估计，全局偏差
random_state	随机种子
seed	随机种子
missing	当数据缺失时的填补值。缺省为np.nan
kwargs	XGBoost Booster的Keyword参数

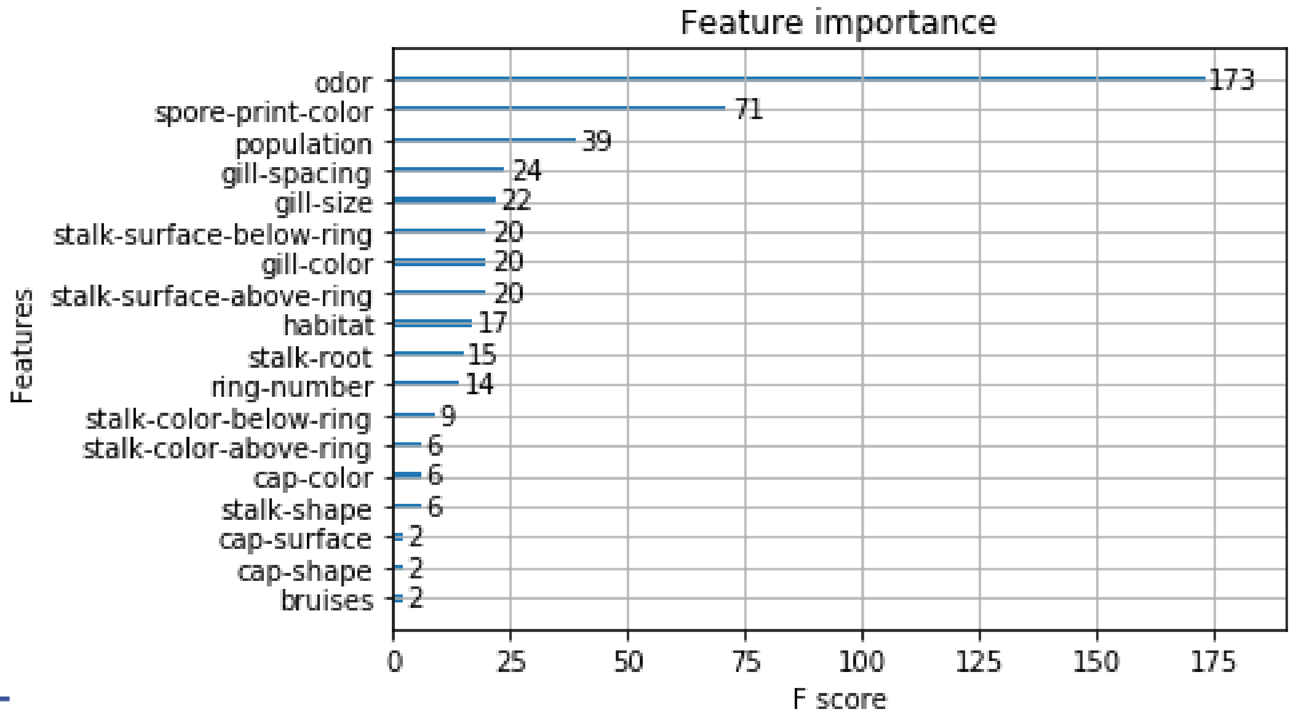
►特征重要性

- 采用树集成学习（如Gradient Boosting）的好处之一是可以从训练好的预测模型得到特征的重要性
- 单棵树中的特征重要性：每个特征分裂点对性能的提升量，并用每个结点的样本数加权。
  - 性能测量可以是用于选择分裂点的指标（纯度）或其他更特别的误差函数
- 整个模型中的特征重要性：模型中每棵树的平均

►特征重要性(cont.)

- 在XGBoost中已经自动算好，存放在feature_importances_
  - from matplotlib import pyplotpyplot.bar(range(len(model_XGB.feature_importances_)), model_XGB.feature_importances_)
  - pyplot.show()
  - 按特征顺序打印
- 还可以使用XGBoost内嵌的函数，按特征重要性排序
  - from xgboost import plot_importance
  - plot_importance(model_XGB)
  - pyplot.show()

►例：蘑菇数据集的特征重要性



►特征选择

- 可以根据特征重要性进行特征选择
  - from sklearn.feature_selection import SelectFromModel

- 输入一个（在全部数据集上）训练好的模型
- 给定阈值，重要性大于阈值的特征被选中
  - selection = SelectFromModel(model, threshold=thresh, prefit=True)
    select_X_train = selection.transform(X_train)

►例：蘑菇数据集上的特征选择

- Thresh=0.000, n=22, Accuracy: 100.00%
- Thresh=0.000, n=22, Accuracy: 100.00%
- Thresh=0.000, n=22, Accuracy: 100.00%
- Thresh=0.000, n=22, Accuracy: 100.00%
- Thresh=0.004, n=18, Accuracy: 100.00%
- Thresh=0.004, n=18, Accuracy: 100.00%
- Thresh=0.004, n=18, Accuracy: 100.00%
- Thresh=0.013, n=15, Accuracy: 100.00%
- Thresh=0.013, n=15, Accuracy: 100.00%
- Thresh=0.013, n=15, Accuracy: 100.00%
- Thresh=0.019, n=12, Accuracy: 100.00%
- Thresh=0.030, n=11, Accuracy: 100.00%
- Thresh=0.032, n=10, Accuracy: 100.00%
- Thresh=0.036, n=9, Accuracy: 100.00%
- Thresh=0.043, n=8, Accuracy: 100.00%
- Thresh=0.043, n=8, Accuracy: 100.00%
- Thresh=0.043, n=8, Accuracy: 100.00%
- **Thresh=0.047, n=5, Accuracy: 100.00%**
- Thresh=0.051, n=4, Accuracy: 99.51%
- Thresh=0.083, n=3, Accuracy: 99.45%
- Thresh=0.152, n=2, Accuracy: 99.45%
- Thresh=0.270, n=1, Accuracy: 98.52%

5个特征就足够好了：

odor

spore-print-color

population

gill-spacing

gill-size

► Kaggle案例：Higgs Boson竞赛

- 竞赛官网： https://www.kaggle.com/c/higgs-boson/
- 两类分类任务：将事件分类为"tau tau decay of a Higgs boson" 或 "background"
  - 每个事件有一个ID，30个特征，权重，和标签
  - 用交叉验证选择迭代次数：3_HiggsBoson_cv.ipynb
  - 与sklearn中的GBM速度与性能比较（ 3_higgsboson_speed.ipynb ）

THANK YOU

北京智百科技有限公司