集成学习

集成学习是通过将多个弱学习器进行结合，提升为强学习器的算法。根据个体学习器的生成方式，目前的集成学习方法大致分为两大类，即个体学习器存在强依赖关系，必须串行生成序列化方法，例如（Boosting），以及个体学习器间不存在强依赖关系，可同时生成的并行化方法，例如（Bagging，Random Forest）。

Bagging

Bagging：bootstrap aggregating 的缩写，是一种并行式集成学习方法，可用于二分类，多分类，回归等任务。

基本流程：

1. 对于一个包含n个数据的数据集$\{x_1, x_2, ..., x_n\}$，进行m次有放回的采样。最后大约有63.2%的数据出现在采样集中。
2. 取T个这样的采样集。
3. 每个采样集训练一个基学习器。
4. 结合预测输出，对分类问题使用简单投票法，对回归任务使用简单平均法。

优点：能够进行包外预测；时间复杂度小；从偏差-方差角度看，减低方差

随机森林

在Bagging的基础上，基学习器为决策树，采用随机属性划分的方式，常常选择的属性子集大小为$k = \log_2 d$。

Boosting

Boosting是一种串行的集成学习方法，其工作机制：先从初始训练集训练一个基学习器，在根据基学习器的表现对训练样本分布进行调整，使得先前基学习器做错的训练样本在后续收到更多关注，然后基于调整后的样本分布来训练下一个基学习器；如此重复进行，直至基学习器数目达到事先指定的值T，最终将这T个基学习器进行加权结合。主要由两部分组成：加法模型和前向分布算法。

加法模型就是说强分类器由一系列弱分类器线性相加而成。一般组合形式如下：

$$H_m(x) = \sum_{i=1}^{m} \alpha_i h_i(x)$$

前向分步就是说在训练过程中，下一轮迭代产生的分类器是在上一轮的基础上训练得来的。也就是可以写成这样的形式：

$$H_m(x) = H_{m-1}(x) + \alpha_m h_m(x)$$

由于采用的损失函数不同，Boosting算法也因此有了不同的类型

AdaBoost

AdaBoost是损失函数为指数损失的Boosting算法。假设有数据集$\{x_1, x_2, ..., x_n\}$，他们的标签$y_i \in \{-1, 1\}$。指数损失函数

$$\ell(H_m|D) = \mathbb{E}_{x\sim D}\left[e^{-y_i H_m(x_i)}\right]$$

容易证明指数损失函数是分类任务原本$0/1$损失函数的一致的替代函数。

更新样本分布

Adaboost算法在获得H_{t-1}之后样本分布将进行调整，使下一轮的基学习器h_t能纠正H_{t-1}的错误，即

$$\begin{aligned}\ell(H_{t-1}+h_t|D) &= \mathbb{E}_{x\sim D}\left[e^{-y_i(H_{t-1}(x_i)+h_t)}\right] = \mathbb{E}_{x\sim D}\left[e^{-y_iH_{t-1}(x_i)}e^{-y_ih_t(x_i)}\right] \\ &= \mathbb{E}_{x\sim D}\left[e^{-y_iH_{t-1}(x_i)}\left(1-y_ih_t(x_i)+\frac{1}{2}\right)\right]\end{aligned}$$

理想的学习器应最小化上述指数损失函数

$$\begin{aligned}h_t^* &= \arg\min_h \ell(H_{t-1}+h_t|D) = \arg\max_h \mathbb{E}_{x\sim D}\left[e^{-y_iH_{t-1}(x_i)}y_ih_t(x_i)\right] \\ &= \arg\max_h \mathbb{E}_{x\sim D}\left[e^{-y_iH_{t-1}(x_i)}y_ih_t(x_i)\right] = \arg\max_h \mathbb{E}_{x\sim D}\left[\frac{e^{-y_iH_{t-1}(x_i)}}{\mathbb{E}_{x\sim D}\left[e^{-y_iH_{t-1}(x_i)}\right]}y_ih_t(x_i)\right]\end{aligned}$$

令

$$D_t(x) = \frac{D(x)e^{-yH_{t-1}(x)}}{\mathbb{E}_{x\sim D}\left[e^{-y_iH_{t-1}(x_i)}\right]}$$

优化目标变成

$$h_t^* = \arg\max_h \mathbb{E}_{x\sim D_t}\left[y_ih_t(x_i)\right] = \arg\min_h \mathbb{E}_{x\sim D_t}\left[\mathbb{I}(y_i \neq h_t(x_i))\right]$$

即h_t应是最小化在数据集D_t分布下错误率的基分类器。

更新学习器的权重α_t，在t次迭代中数据分布为D_t，指数损失函数可表示为

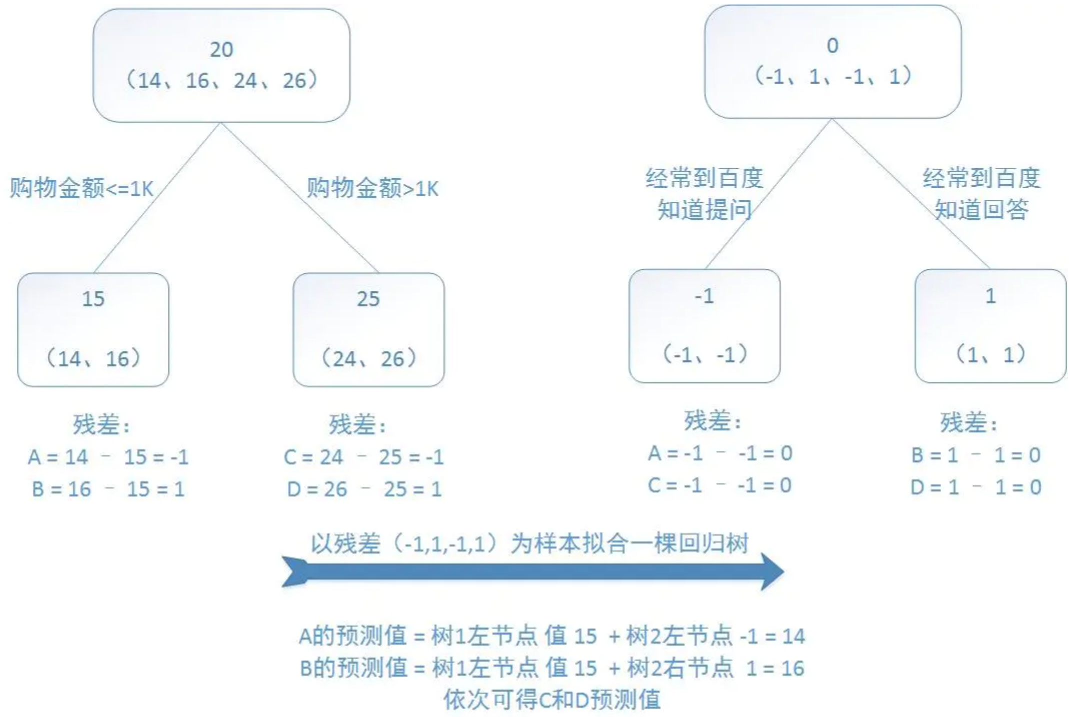
$$\begin{aligned}\ell(\alpha_th_t|D_t) &= \mathbb{E}_{x\sim D_t}\left[e^{-y_i\alpha h_t(x_i)}\right] = \mathbb{E}_{x\sim D_t}\left[e^{-\alpha_t}\mathbb{I}(y_i=h_t(x_i))+e^{\alpha_t}\mathbb{I}(y_i\neq h_t(x_i))\right] \\ &= e^{-\alpha_t}P_{x\sim D_t}(y_i=h_t(x_i))+e^{\alpha_t}P_{x\sim D_t}(y_i\neq h_t(x_i)) \\ &= e^{-\alpha_t}(1-\epsilon_t)+e^{\alpha_t}\epsilon_t\end{aligned}$$

其中$\epsilon_t = P_{x\sim D_t}(y_i \neq h_t(x_i))$，对指数函数求导

$$\frac{\partial \ell(\alpha_th_t|D_t)}{\partial \alpha_t} = -e^{-\alpha_t}(1-\epsilon_t)+e^{\alpha_t}\epsilon_t = 0, \quad \alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$$

Gradient Boosting Decision Tree

GBDT是一种集成模型，采用的基础模型是CART回归树。每一棵回归树学习的是之前所有树的结论和残差，拟合得到一个当前的残差回归树，



监督学习的关键概念：模型（model）、参数（parameters）、目标函数（objective function）

模型就是所要学习的条件概率分布或者决策函数，它决定了在给定特征向量时如何预测出目标。这里GBDT使用的是加法模型，即将所以的CART树预测结果相加。

参数就是我们要从数据中学习得到的内容。模型通常是由一个参数向量决定的函数。

目标函数这里使用平方误差损失函数

$$L(y, H_m(x)|D) = \sum_{i=1}^{m} \frac{1}{2}(y_i - H_m(x_i))^2$$

计算残差$\tilde{y}_i$，即损失函数的负梯度

$$\tilde{y}_i = -\frac{\partial L}{\partial H_m(x_i)}|_{m=t-1} = y_i - H_{t-1}(x_i)$$

将数据集$\{x_i, \tilde{y}_i\}$作为输入训练第t棵回归树

$$\omega^* = \arg\min_{\omega} \sum_{i=1}^{m} [\tilde{y}_i - h_t(x_i;\omega)]^2$$

为学习器赋予权重

$$\alpha_t^* = \arg\min_{\alpha} L(y, H_{t-1}(x) + \alpha_t h_t(x;\omega^*)|D)$$

可以得到最后的GBDT模型

$$H_t(x) = H_{t-1}(x) + \alpha_t^* h_t(x;\omega^*)$$

eXtreme Gradient Boosting

XGBoost（eXtreme Gradient Boosting）极致梯度提升，是基于GBDT的一种算法。整体的目标函数：

$$
\begin{aligned}
L(y, H_t(x)|D) &= \sum_{i=1}^{m} \ell(y_i, H_t(x_i)) + \sum_k \Omega(h_k(x)) \\
&= \sum_{i=1}^{m} \ell(y_i, H_{t-1}(x_i) + h_t(x_i)) + \sum_k \Omega(h_k(x)) \\
&\approx \sum_{i=1}^{m} \left[g_i h_t(x_i) + \frac{1}{2} h_i h_t^2(x_i) \right] + \Omega(h_t(x))
\end{aligned}
$$

这里移除了常数项$\ell(y_i, H_{t-1}), \sum_{k\neq t} \Omega(h_k(x))$。接下来重新定义一棵树，假设有$T$个街道，有映射关系$q(x) = \{1, .., T\}$，不同节点的权重$w_{q(x)}$。描述树的复杂度

$$
\Omega(h_t(x)) = \gamma T + \frac{\lambda}{2} \sum_{i=1}^{T} \omega_i^2
$$

第一部分限制节点数，第二部分限制权重的l_2范数。因为$h_t(x) = \omega_{q(x)}$，代入目标函数

$$
\begin{aligned}
L(y, H_t(x)|D) &= \sum_{j=1}^{T} \left[\sum_{q(x_i)=j} g_i \omega_j + \frac{1}{2} (\lambda + \sum_{q(x_i)=j} h_i) \omega_j^2 \right] + \gamma T \\
&= \sum_{j=1}^{T} \left[G_j \omega_j + \frac{1}{2} (\lambda + H_j) \omega_j^2 \right] + \gamma T
\end{aligned}
$$

最小化目标函数

$$
h(q(x_i) = j) = \omega_j^* = \arg\min_{\omega_j} L(y, H_t(x); \omega_j|D) = -\frac{G_j}{H_j + \lambda}
$$

得到目标函数的最小值

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

用启发式的方式生长树，

$$
Gain = Obj_m^t - Obj_{m-1}^t = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma
$$

LightGBM

参考[标点符]

LightGBM在很多方面会比XGBoost表现的更为优秀。它有以下优势：

- 更快的训练效率
- 低内存使用
- 更高的准确率
- 支持并行化学习
- 可处理大规模数据
- 支持直接使用category特征

概括来说，lightGBM主要有以下特点：

- 基于Histogram的决策树算法
- 带深度限制的Leaf-wise的叶子生长策略
- 直方图做差加速
- 直接支持类别特征(Categorical Feature)
- Cache命中率优化

- 基于直方图的稀疏特征优化
- 多线程优化