Boosting

> version: 1.0
>
> - 理论待完善
> - 代码待跟进

理论基础

名词解释

集成学习：通过构建并结合多个学习器来完成学习任务。

同质 homogeneous：决策树集成中全是决策树，神经网络集成中全是神经网络。

基学习器 base learner：同质集成中的个体学习器。

基学习算法 base learning algorithm：基学习器所使用的学习算法。

异质 heterogenous：集成包含不同类型的个体学习器。

组件学习器 component learner：和基学习器对应，它们统称为个体学习器。

集成学习的可行性证明

假设二分类问题 $y \in \{-1, +1\}$ 和真实函数 f，假定基分类器的错误率是 ϵ，即对每个**基分类器** h_i 有：

$$P\left(h_i(\boldsymbol{x}) \neq f(\boldsymbol{x})\right) = \epsilon$$

假设集成通过投票结合 T 个基分类器，若有超过半数的基分类器正确，则集成分类就正确：

$$H(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i=1}^{T} h_i(\boldsymbol{x})\right)$$

根据 Hoeffding 不等式，得到集成后的错误率：

$$
\begin{aligned}
P(H(\boldsymbol{x}) \neq f(\boldsymbol{x})) &= \sum_{k=0}^{\lfloor T/2 \rfloor} \binom{T}{k} (1-\epsilon)^k \epsilon^{T-k} \\
&\leqslant \exp\left(-\frac{1}{2} T (1-2\epsilon)^2\right)
\end{aligned}
$$

> $P(H(n) \leqslant k)$ 是另一种写法，含义相同。
>
> 由这条表达式，我们有：
>
> $$
> h_i(x) = \begin{cases} 1 & C_n^x p^x (1-p)^{n-x} >= 0.5 \\ -1 & C_n^x p^x (1-p)^{n-x} < 0.5 \end{cases}
> $$

第一个等号表示 n 个基学习器中分类正确的个数小于 k 的概率。若假定集成通过简单投票法结合 n 个分类器，超过半数的基学习器正确，则集成分类就正确，即 $k = 0.5 * n = (1-\epsilon-\delta)n$ 。

第二个等号的 Hoeffding 不等式的定义，$\delta > 0$：

$$
P(H(n) \leqslant (p-\delta)n) \leqslant e^{-2\delta^2 n}
$$

其中 $\binom{T}{k}$ 表示 C_T^k ，$\delta = 0.5 - \epsilon$ 。

当 $\epsilon >= 0.5$ 时，上式不成立。随着集成中个体分类器数目 T 的增大，集成的错误率将指数级下降，最终趋向于零。

> 在现实中，个体学习器是解决同一个问题训练出来的，它们不可能相互独立，如何生成不同的个体学习器，是集成学习研究的核心。

根据个体学习器的生成方式，目前集成学习方法大致分为两大类：**个体学习期之间存在强依赖关系、必须串行生成的序列化方法—— Boosting**；**个体学习器之间不能存在强依赖关系、可同时生成的并行化方法—— Bagging**。

Boosting

> 先从初始训练集训练出一个基学习器，再根据基学习器的表现对训练样本进行调整，使得先前基学习器出错的训练样本在后续受到更多关注，然后基于调整后的样本分布来训练下一个基学习器。

AdaBoost

只适用二分类任务，比较容易理解的是基于“线性模型” additive model ，即基学习器的线性组合：

$$
H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})
$$

最小化指数损失函数：

$$
\ell_{\exp}(H|\mathcal{D}) = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}} \left[e^{-f(\boldsymbol{x}) H(\boldsymbol{x})} \right]
$$

$f(x)$ 只有两个结果，1 或 -1 ：

$$\ell_{\exp}(H|\mathcal{D}) = e^{-H(x)}P(f(x)=1) + e^{H(x)}P(f(x)=-1)$$

若 $H(x)$ 可以将损失函数最小化，那么对它求偏导：

$$\frac{\partial \ell_{\exp}(H|\mathcal{D})}{\partial H(\boldsymbol{x})} = -e^{-H(\boldsymbol{x})}P(f(\boldsymbol{x})=1|\boldsymbol{x}) + e^{H(\boldsymbol{x})}P(f(\boldsymbol{x})=-1|\boldsymbol{x})$$

令上式为0，可以解出：

$$H(\boldsymbol{x}) = \frac{1}{2}\ln\frac{P(f(x)=1|\boldsymbol{x})}{P(f(x)=-1|\boldsymbol{x})}$$

因此：

$$\begin{aligned}\operatorname{sign}(H(\boldsymbol{x})) &= \operatorname{sign}\left(\frac{1}{2}\ln\frac{P(f(x)=1|\boldsymbol{x})}{P(f(x)=-1|\boldsymbol{x})}\right)\\ &= \begin{cases}1, & P(f(x)=1|\boldsymbol{x}) > P(f(x)=-1|\boldsymbol{x})\\ -1, & P(f(x)=1|\boldsymbol{x}) < P(f(x)=-1|\boldsymbol{x})\end{cases}\\ &= \underset{y\in\{-1,1\}}{\arg\max} P(f(x)=y|\boldsymbol{x})\end{aligned}$$

我们发现，指数损失函数最小化，则分类错误率也将最小，即达到了贝叶斯最优错误率。

当基分类器得到以后，该基分类器的权重 a_t 应该使得 $a_t h_t$ 最小化指数损失函数:

$$\ell_{\exp}\left(\alpha_t h_t|\mathcal{D}_t\right) = e^{-\alpha_t}\left(1-\epsilon_t\right) + e^{\alpha_t}\epsilon_t$$

其中 $\epsilon_t = P_{x\sim\mathcal{D}_t}\left(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x})\right)$，考虑指数损失函数的导数：

$$\frac{\partial \ell_{\exp}\left(\alpha_t h_t|\mathcal{D}_t\right)}{\partial \alpha_t} = -e^{-\alpha_t}\left(1-\epsilon_t\right) + e^{\alpha_t}\epsilon_t$$

上式为0，可以得到**权重更新公式**：

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

AdaBoost 算法在下一轮基学习中纠正错误，那么：

$$\ell_{\exp}\left(H_{t-1}+h_t|\mathcal{D}\right) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})(H_{t-1}(\boldsymbol{x})+h_t(\boldsymbol{x}))}\right]$$

！！！它可以进行泰勒展开，同时注意到 $f^2(x)=h_t^2(x)=1$：

$$\begin{aligned}\ell_{\exp}\left(H_{t-1}+h_t|\mathcal{D}\right) &\simeq \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\left(1-f(\boldsymbol{x})h_t(\boldsymbol{x})+\frac{f^2(\boldsymbol{x})h_t^2(\boldsymbol{x})}{2}\right)\right]\\ &= \mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(x)H_{t-1}(x)}\left(1-f(x)h_t(x)+\frac{1}{2}\right)\right]\end{aligned}$$

理想的基学习器：

$$
\begin{aligned}
h_t(\boldsymbol{x}) &= \arg\min_{\boldsymbol{h}} \ell_{\exp}\left(H_{t-1}+h|\mathcal{D}\right) \\
&= \arg\min_{h} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\left(1-f(\boldsymbol{x})h(\boldsymbol{x})+\frac{1}{2}\right)\right] \\
&= \arg\max_{h} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}f(\boldsymbol{x})h(\boldsymbol{x})\right] \\
&= \arg\max_{h} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]}f(\boldsymbol{x})h(\boldsymbol{x})\right]
\end{aligned}
$$

因为 $\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]$ 是一个常数，令 $\mathcal{D}_t$ 表示一个分布：

$$
\mathcal{D}_t(\boldsymbol{x}) = \frac{\mathcal{D}(\boldsymbol{x})e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{w})}\right]}
$$

这等价于令：

$$
\begin{aligned}
h_t(\boldsymbol{x}) &= \arg\max_{\boldsymbol{h}} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]}f(\boldsymbol{x})h(\boldsymbol{x})\right] \\
&= \arg\max_{\boldsymbol{h}} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[f(\boldsymbol{x})h(\boldsymbol{x})\right]
\end{aligned}
$$

由于 $f(x), h(x)\in\{-1,+1\}$，有：

$$
f(\boldsymbol{x})h(\boldsymbol{x}) = 1-2\mathbb{I}(f(\boldsymbol{x})\neq h(\boldsymbol{x}))
$$

那么理想的基学习器：

$$
h_t(\boldsymbol{x}) = \arg\min_{h} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[\mathbb{I}(f(\boldsymbol{x})\neq h(\boldsymbol{x}))\right]
$$

样本分布更新公式：

$$
\begin{aligned}
\mathcal{D}_{t+1}(\boldsymbol{x}) &= \frac{\mathcal{D}(\boldsymbol{x})e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}\right]} \\
&= \frac{\mathcal{D}(\boldsymbol{x})e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}\right]} \\
&= \mathcal{D}_t(\boldsymbol{x})\cdot e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}\frac{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}\right]}
\end{aligned}
$$

> 重赋权法：根据样本分布为每个样本重新赋予一个权重。
>
> 重采样法：根据样本分布对训练集重新采样，再用采样样本集对基学习器进行训练。
>
> Boosting 主要关注降低偏差，因此 Boosting 能基于泛化性能相当弱的学习器构建很强的集成。

提升树

以决策树为基函数的提升方法被称为提升树，对分类问题决策树是二叉分类树，对回归问题决策树是二叉回归树。提升树可以表示为决策树的加法模型：

$$f_M(x)=\sum_{m=1}^{M}T\left(x;\Theta_m\right)$$

其中 $T\left(x;\Theta_n\right)$ 表示决策树；Θ_n 表示决策树的参数； M 是树的个数。

提升树算法采用**前向分步算法**。首先确定初始提升树 $f_0(x)=0$ ，第 m 步的模型是：

$$f_m(x)=f_{m-1}(x)+T\left(x;\Theta_m\right)$$

通过**经验风险极小化**确定下一棵决策树的参数：

$$\hat{\Theta}_m=\arg\min_{\theta_e}\sum_{i=1}^{N}L\left(y_i,f_{m-1}\left(x_i\right)+T\left(x_i;\Theta_m\right)\right)$$

> 这里的 T 指的就是下一棵决策树。

不同问题的提升树学习算法，主要区别在于使用的损失函数不同，平方误差损失函数的回归问题，指数损失函数的分类问题。下面叙述回归问题的提升树：

$$T(x;\Theta)=\sum_{j=1}^{J}c_jI\left(x\in R_j\right)$$

x 是输入， y 是输出，c 是输出常量，J 是回归树的复杂度即叶节点的个数，
$\Theta=\left\{\left(R_1,c_1\right),\left(R_2,c_2\right),\cdots,\left(R_J,c_J\right)\right\}$ 表示树的区域划分和各区域上的常数。

回归问题提升树使用以下前向分布算法：

$$f_0(x)=0$$

$$f_m(x)=f_{m-1}(x)+T\left(x;\Theta_m\right),\quad m=1,2,\cdots,M$$

$$f_M(x)=\sum_{m=1}^{M}T\left(x;\Theta_m\right)$$

在前向分布算法的第 m 步，给定当前模型 $f_{m-1}(x)$ ，需求解：

$$\hat{\Theta}_m=\arg\min_{\Theta_m}\sum_{i=1}^{N}L\left(y_i,f_{m-1}\left(x_i\right)+T\left(x_i;\Theta_m\right)\right)$$

当使用平方误差损失函数时：

$$L(y,f(x))=(y-f(x))^2$$

$$\begin{aligned}L\left(y,f_{m-1}(x)+T\left(x;\Theta_m\right)\right)&=\left[y-f_{m-1}(x)-T\left(x;\Theta_m\right)\right]^2\\&=\left[r-T\left(x;\Theta_m\right)\right]^2\end{aligned}$$

这里， $r=y-f_{m-1}(x)$ ，是当前模型拟合数据的残差。

回归问题的提升树算法：

输入：训练数据集 $T=(x1,y1),(x2,y2),\ldots(xn,yn),xi,yi$

输出：提升树 $f_M(x)$

1. 初始化 $f_0(x)=0$
2. 开始循环 m = 1,2,...M
3. 计算残差：$r_{mi}=y_i-f_{m-1}\left(x_i\right), \quad i=1,2,\cdots,N$
4. 拟合残差 r_{mi} 学习一个回归树，得到 $T\left(x;\Theta_m\right)$
5. 更新 $f_m(x)=f_{m-1}(x)+T\left(x;\Theta_m\right)$
6. 得到回归问题提升树 $f_M(x)=\sum_{m=1}^{M}T\left(x;\Theta_m\right)$

梯度提升

这是利用最速下降法的近似方法，其关键是利用损失函数的负梯度在当前模型的值：

$$-\left[\frac{\partial L\left(y,f\left(x_i\right)\right)}{\partial f\left(x_i\right)}\right]_{f(x)=f_{m+1}(x)}$$

作为回归问题提升树算法中的残差的近似值。

输入：训练数据集 $T=(x1,y1),(x2,y2),\ldots(xn,yn),xi,yi$； 损失函数 $L(y,f(x))$

输出：回归树 $\hat{f}\left(x\right)$

1. 初始化：$f_0(x)=\arg\min_c\sum_{i=1}^{N}L\left(y_i,c\right)$
2. 开始循环 m 从 1 到 M
3. 对于 i 从 1 到 N，计算：$r_{ml}=-\left[\frac{\partial L(y_i,f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$
4. 对 r_{mi} 拟合一个回归树，得到第 m 棵树的叶节点区域 R_{mj}
5. 对 j 从 1 到 J，计算：$c_{mj}=\arg\min_c\sum_{x_i\in R_{mj}}L\left(y_i,f_{m-1}\left(x_i\right)+c\right)$
6. 更新 $f_m(x)=f_{m-1}(x)+\sum_{j=1}^{J}c_{mj}I\left(x\in R_{mj}\right)$
7. 得到回归树 $\hat{f}\left(x\right)=f_M(x)=\sum_{m=1}^{M}\sum_{j=1}^{J}c_{mj}I\left(x\in R_{mj}\right)$

XGBoost

原始的 GBDT 算法基于经验损失函数的负梯度来构造新的决策树，只是在决策树构建完成后再进行剪枝，而 XGBoost 在决策树构建阶段就加入了正则化：

$$L_t=\sum_{i=1}^{n}l\left(y_i,F_{t-1}(x_i)+f_t(x_i)\right)+\sum_{k=1}^{K}\Omega\left(f_k\right)$$

正则化定义：

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

其中 T 是决策树 f_t 中叶子节点的个数， w_j 是第 j 个叶子节点的预测值，该损失函数在 F_{t-1} 处进行二阶泰勒展开：

$$L_i\approx\overline{L}_r=\sum_{j=1}^{T}[G_jw_j+\frac{1}{2}(H_j+\lambda)w_j^2]+\gamma T$$

这里 G 是一阶导，H 是二阶导，通过将损失函数对 w_j 的导数为 0，可以求出在最小化损失函数的情况下各个叶子节点上的预测值：

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

将预测值带入损失函数可以得到损失函数的最小值：

$$\tilde{L}_t^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

分裂前后损失函数的差值：

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

XGBoost 就是最大化这个差来进行决策树的构建，总的来说， XGBoost 和 GDBT 的差别和联系：

- GDBT 是机器学习算法， XGBoost 是该算法的工程实现。
- XGBoost 加入了正则化，支持多种类型的基分类器，支持对数据采样（和 RF 类似），能对缺省值处理。

算法实现

AdaBoost 伪码

```
"""
训练集 D = {(x1, y1), (x2, y2)..., (xm, ym)}
基学习算法 L
训练轮数 T
"""
D[1] = 1/m # 初始化样本权值分布
for t in range(T):
  h[t] = L(D, D[t]) # Dt 是数据分布
  e[t] = P(ht(x), f(x)) # 分类器 ht 的误差， ht(x) 是预测结果， f(x) 是真实结果
  if e[t] > 0.5:
    break
  a[t] = 0.5*np.log((1-e[t])/e[t])
  D[t+1] = D[t] / Z[t] # Zt 是规范化因子，以确保 D[t+1] 是一个分布
  if (h[t](x) == f(x)) D[t+1] *= exp(-a[t])
  else D[t+1] *= exp(a[t])
```

最终返回 $H(x) = \text{sign}\left(\sum_{t=1}^{T}\alpha_t h_t(\boldsymbol{x})\right)$ 。

> sign 表达符号函数。

经典题目

XGBoost 与 GBDT 的联系与区别有哪些?

从方差和偏差的角度解释 Boosting 和 Bagging 的原理?