机器学习笔记–*AdaBoost* 和 *XGBoost*

空修菜

1 自适应提升（Boosting）方法: AdaBoost 和 XGBoost

自适应提升 (adaptive boost), 简写为 AdaBoost. 该算法的思想是: 先根据情况预设一个基分类器, 然后用该基分类器来对训练集进行训练, 将没有被该基分类器正确归类的样本点的权重加大, 使得下一个分类器再训练的时候更加关注前一个分类器分类失败的样本点, 这样一次进行下去就会得到许多不同分类器, 一般称这样的分类器为"弱分类器", 后一个分类器都会考虑前一个分类器分类失败的样本点, 最后将这些弱分类器相加就得到了最终的模型, 该模型称为强分类器.

1.1 分类 AdaBoost 算法

设 $x^{(i)} \in X \subset \mathbb{R}^n,\ y^{(i)} \in \{-1, +1\}$, 二分训练集为 S,

$$S = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})\}.$$

1. 初始化训练集的权重, 一般为 $1/N$. D_1 是训练集的权重向量, 分量 w_{1i} 是训练集 S 的第 i 个样本点 $x^{(i)}$ 的权重, 加上权重的样本点为 $w_{1i}x^{(i)}$.

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

2. 假设需要训练的弱分类器的数量为 M, 对于 $m = 1, 2, \ldots, M$

   (1) 使用加上权重 D_m 的训练集进行训练, 得到基本分类器 $G_m(x)$:

$$G_m : X \to \{-1, +1\}.$$

   (2) 计算已得弱分类器 G_m 在加上权重的训练集上的分类误差率 e_m

$$e_m = \sum_{i=1}^{N} P\big(G_m(x^{(i)}) \neq y^{(i)}\big) = \sum_{i=1}^{N} w_{mi} I\{G_m(x^{(i)}) \neq y^{(i)}\}. \quad (1.1)$$

(3) 计算 $G_m(x)$ 的系数

$$\alpha_m = \frac{1}{2}\log\frac{1-e_m}{e_m}. \tag{1.2}$$

(4) 在训练下一个弱分类器之前, 更新训练集的权重,

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2}, \ldots, w_{m+1,N}), \tag{1.3}$$

$$w_{m+1,i} = \frac{w_{m,i}}{Z_m}\exp\big(-\alpha_m y^{(i)} G_m(x^{(i)})\big), \tag{1.4}$$

$$Z_m = \sum_{i=1}^{N} w_{m,i}\exp\big(-\alpha_m y^{(i)} G_m(x^{(i)})\big). \tag{1.5}$$

3. 将所得的 M 个弱分类器 G_m 乘以各自的系数 α_m 做线性组合, 得到 f, 最终的强分类器为 $G = \mathrm{sign}(f)$.

$$f(x) := \sum_{m=1}^{M} \alpha_m G_m(x)$$
$$G(x) := \mathrm{sign}(f(x)).$$

以上就是 AdaBoost 算法. 关于分类误差率 (1.1) 式, 当求第一个弱分类器 ($m = 1$) 时, (1.1) 式子等于

$$e_m = \sum_{i=1}^{N} P\big(G_m(x^{(i)}) \neq y^{(i)}\big) = \frac{1}{N}\sum_{i=1}^{N} I\{G_m(x^{(i)}) \neq y^{(i)}\},$$

此时训练集的每一个样本点的权重都是 $1/N$, 所以分类误差率就等于没算对的个数除以总行数, 由于行数 N 是固定的, 只需确定没算对的个数.

关于弱分类器的系数 α_m, 由 (1.2) 式可知, $\alpha_m(e_m)$ 是一个关于 e_m 的减函数, 当分类误差率较小的时候, 相应的弱分类器的权值就较大.

在进行新的弱分类器的训练之前, 需要更新训练集的权重, 使得被 G_1 归类错误的样本点被赋予较大的权重. 假设第 k 个样本点被归类错误, 则

$$\exp\big(-\alpha_m y^{(k)} G_m(x^{(k)})\big) = \exp(\alpha_m),$$

再将 $\exp(\alpha_m)$ 与原来的权重系数 $w_{m,k}$ 相乘, 就得到关于样本点 $x^{(k)}$ 的权重.

由 (1.4) 式、(1.5) 式可知,

$$\sum_{j=1}^{N} w_{m,j} = 1.$$

最后, 在训练后面的 $M-1$ 个弱分类器时, 是将分类器分类错误的样本点的权重系数相加.

1.2 Summary of BoostingTree

1. 使用 AdaBoost 的注意事项:

   (1). 计算的权重向量不能与训练集相乘, 权重向量用于计算弱估计器的误差率, 在得到一个弱估计器 $G_m(X)$ 后, 计算误差率. 此时的误差率不是估计器 $G_m(X)$ 预测正确的个数与训练集大小的比, 而是 $G_m(X)$ 预测错误的个数和与之相对权重向量的积的和:

   $$e_m = \sum_{i=1}^{N} w_{mi} I\{G_m(x^{(i)}) \neq y^{(i)}\}.$$

   (2).

2. 弱估计器 (树桩 Stump) 的生成. 弱估计器的生成关键在于阈值的确定, 或者说划分标准的选择, 有两种方式可以得到阈值:

   (1). 取每一个特征的最大值与最小值, 然后用最大值减去最小值, 得到对应特征的变化范围, 最后再将所得的差除以 10, 得到变化的步长. 也不一定要除以 10, 其它数也可以, 但是这个数不能过大, 因为过大的数将使得步长很小, 步长小可能会使得估计器的误差率高.

   $$\text{threshold} = \frac{\max(\text{feature}) - \min(\text{feature})}{10}.$$

   (2). 另一个方式是: 去掉特征的重复值, 然后将去重复的特征升序排序. 取去重排序的特征的一前一后的中点作为阈值.

3. 阈值的使用. 在得到相应的阈值后, 要使用该阈值对数据集进行分割, 并计算分类错误率, 此时的分类错误率就是简单的计算错误的个数除以样本总数（不要与弱估计器的 e_m 混淆）.

   此时的问题是: 到底是将小于阈值的数据归类为正类, 还是将小于阈值的数据归类为负类? 此时都要计算一边, 然后取误差率最小的符号. 比如阈值是 3.5, 将小于 3.5 的数据归类为正, 所得的误差率是 0.3; 将小

于 3.5 的数据归类为负, 所得的误差率是 0.2, 则该阈值对应的符号就是小于.

4. 树桩的建立. 此时需要的树是一个父节点带两个叶节点, 是简单树. 所以不用递归地建立树, 只需要得到一个节点, 然后将对应的节点属性放入其中即可, 父节点需要的属性有:

   - 状态 `State` 为 0 或 1, 用于确定阈值将那一部分归类为正、将那一部分归类为负. 当`State == 0`, 小于该阈值的就是正类; 当`State == 1` 大于该阈值的就是正类; `State` 初始化为 1.
   - `MinError` 用于记录每一个估计器的最小误差. 此时的最小误差值的是权重向量与估计错误向量的内积.
   - `BestSplitFea` 是最优的特征的位置索引, `Threshold` 是对应索引的阈值.

5. 分类 AdaBoost 的损失函数是指数损失函数:

$$\mathcal{L}(y, f(x)) = \exp[-yf(x)],$$

   其中, y 是 x 对应的类别, $y \in \{-1, 1\}$, $f(x)$ 是相应的强分类器.

1.3 回归 Adaboost 算法

1. 回归 AdaBoost 的损失函数一般是平方误差损失函数:

$$\mathcal{L}(y, f(x)) = \frac{1}{2}\Big(y - f(x)\Big)^2.$$

2. 残差 (residual) R 是真实目标值与预测的目标值的差: $y - f(x)$. 当损失函数是平方误差损失函数时, 残差与损失函数的导数相等.

$$\frac{\partial \mathcal{L}(y, f)}{\partial x} = y - f(x) = R$$

3. 回归树对残差进行拟合, 第一棵树确定的是均值, 第二棵树确定的则是前一棵树的值与目标 y 值之间的距离, 最后将所有的树相加得到预测函数.

4. 之前的树在建立的时候, 节点的切分准则不是 Gini 系数, 就是 Entropy, 此处应该将节点切分准则设置为平方误差, 找到使得平方误差最小的特征作为切分特征以及对应的阈值.

2 梯度提升 (Gradient Boosting Tree)

2.1 回归的梯度提升树

1. 只写了分类树的代码, 但是分类的梯度提升树以梯度提升回归树作为基础, 这实际上也就写了梯度提升回归树的代码.

2. 梯度提升树也是一个前向加法模型. $f_m(x) = f_{m-1}(x) + h_m(x)$, 其中, $f_{m-1}(x)$ 是第 $m-1$ 个弱学习器, 第 m 个模型, 由第 $m-1$ 个模型 $f_{m-1}(x)$ 加上一个新的项 $h_m(x)$ 构成, 所以第 m 步要学习的其实是 h_m.

3. 在 AdaBoost 算法中, 就是直接拟合残差 $(y - f(x))$, 但是梯度提升拟合的是损失函数的一阶导数. 当损失函数是平方误差损失函数时, 残差与损失函数的导数相等.

$$\frac{\partial \mathcal{L}(y, f)}{\partial x} = y - f(x) = R.$$

   所以, AdaBoost 回归算法实际上是梯度提升回归算法的一个特例. 梯度提升算法可以适用于多种多样的损失函数, 只要该损失函数可导且一阶导数存在.

4. 假设 $\mathcal{L}$ 是损失函数, $h_m(x) \in H$, H 是函数空间. 要使得损失函数达到最小:

$$\min_{h_m \in H} \mathcal{L}(y, f_m(x)) = \min_{h_m \in H} \mathcal{L}\big(y, f_{m-1}(x) + h_m(x)\big)$$

   在 $f_{m-1}(x)$ 对 $\mathcal{L}$ 一阶泰勒展开:

$$\mathcal{L}\big(y, f_{m-1}(x) + h_m(x)\big) \approx \mathcal{L}\big(y, f_{m-1}(x)\big) + \frac{\partial \mathcal{L}(y, f_{m-1}(x))}{\partial f_{m-1}(x)} h_m(x).$$

   类似于参数的梯度下降法, 取

$$h_m(x) = -\frac{\partial \mathcal{L}(y, f_{m-1}(x))}{\partial f_{m-1}(x)},$$

   此时损失函数下降最快.

5. 所以, 梯度下降法拟合的是损失函数的负梯度, 即负的一阶导数.

6. 对于树的节点值 c_{mj} 的具体计算:

$$c_{mj} = \arg\min_{c} \sum_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i) + c),$$

对 c 求偏导, 则有

$$\begin{aligned}\frac{\partial L}{\partial c} &= \frac{\partial \sum\limits_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i) + c)}{\partial c} \\ &= \frac{\partial \sum\limits_{x_i \in R_{mj}} \frac{1}{2}(y_i - f_{m-1}(x_i) - c)^2}{\partial c} \\ &= \sum_{x_i \in R_{mj}} [c - (y_i - f_{m-1}(x_i))]\end{aligned}$$

令 $\frac{\partial L}{\partial c} = 0$, 则

$$\begin{aligned}c_{mj} &= \frac{1}{N_{mj}} \sum_{x_i \in R_{mj}} (y_i - f_{m-1}(x_i)) \\ &= \frac{1}{N_{mj}} \sum_{x_i \in R_{mj}} r_{mi},\end{aligned}$$

其中 N_{mj} 代表的是第 m 棵决策树的第 j 个叶结点区域包含的样本数量.

2.2 分类的梯度提升树

1. 二分类 GBDT 的损失函数 $L(y, f(x))$ 定义为:

$$L(y, f(x)) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}),$$

其中, $\hat{y}$ 是样本 x 为 1 的概率 $P(y = 1 \mid x)$,

$$\hat{y} = \frac{1}{1 + e^{-f(x)}},$$

所以, 总体的损失函数 J:

$$J = \sum_{i=1}^{N} L(y_i, f_m(x_i)),$$

因此,

$$\frac{\partial J}{\partial f_{m-1}(x_i)} = \frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)} = \frac{1}{1 + e^{-f_{m-1}(x_i)}} - y_i = \hat{y}_i - y_i.$$

所以, 负梯度为

$$-\frac{\partial J}{\partial f_{m-1}(x_i)} = y - \hat{y}_i.$$

2. 二分类的 GBDT 算法:

输入: 训练集 $S = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$, $x_i \in \mathbb{R}^n$, $y \in \{0, 1\}$, 学习率 β_m.

输出: 提升树 $f_M(x)$.

(1) 初始化 $f_0(x) = \log \frac{p_1}{1-p_1}$, 其中, p_1 是正类占总样本的比例.

(2) 对于 $m = 1, 2, \cdots, M$:

(a) 计算负梯度:

$$r_{mi} = y_i - \hat{y}_i, \ i = 1, 2, \cdots, N,$$

其中,

$$\hat{y}_i = \frac{1}{1 + e^{-f_{m-1}(x_i)}}$$

是 $y_i = 1$ 的概率.

(b) 拟合负梯度 r_{mj} 学习一棵回归树, 得到第 m 棵树的叶节点区域 R_{mj}, $j = 1, 2, \cdots, J$. 拟合这棵树所使用的损失函数依旧是平方误差损失.

(c) 对 $j = 1, 2, \cdots, J$, 计算节点值:

$$c_{mj} = \arg\min_c \sum_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i) + c),$$

具体地,

$$c_{mj} = \frac{\sum_{x_i \in R_{mj}} r_{mj}}{\sum_{x_i \in R_{mj}} (y_i - r_{mj})(1 - y_i + r_{mi})}.$$

(d) 更新 $f_m(x)$:

$$f_m(x) = f_{m-1}(x) + \beta_m \sum_{j=1}^{J} c_{mj} I(x \in R_{mj}).$$

(3) 得到二分类问题的提升树:

$$f_M(x) = \sum_{m=1}^{M} \beta_m \sum_{j=1}^{J} c_{mj} I(x \in R_{mj}).$$

3. 二分类 GBDT 为什么初始化 $f_0(x) = \log \frac{p_1}{1-p_1}$?

$$
\begin{aligned}
J &= \sum_{i=1}^{N} L(y_i, f(x_i)) \\
&= \sum_{i=1}^{N} [-y_i \log \hat{y}_i - (1-y_i) \log(1-\hat{y}_i)] \\
&= \sum_{i=1}^{N} [-y_i \log \frac{1}{1+e^{-f(x_i)}} - (1-y_i) \log(1 - \frac{1}{1+e^{-f(x_i)}})].
\end{aligned}
$$

对 f_0 求导可得,

$$
\begin{aligned}
\frac{\partial J}{\partial f_0(x)} &= \frac{\partial \sum_{i=1}^{N} [-y_i \log \frac{1}{1+e^{-f_0(x_i)}} - (1-y_i) \log(1 - \frac{1}{1+e^{-f_0(x_i)}})]}{\partial f_0(x)} \\
&= \frac{\partial \sum_{i=1}^{N} [(1-y_i) f_0(x_i) + \log(1+e^{-f_0(x_i)})]}{\partial f_0(x)} \\
&= \sum_{i=1}^{N} (1 - y_i - \frac{e^{-f_0(x_i)}}{1+e^{-f_0(x_i)}}).
\end{aligned}
$$

令 $\frac{\partial J}{\partial f_0(x)} = 0$ 得到

$$
f_0(x) = \log \frac{\sum_{i=1}^{N} y_i}{N - \sum_{i=1}^{N} y_i} = \log \frac{(\sum_{i=1}^{N} y_i)/N}{(N - \sum_{i=1}^{N} y_i)/N} = \log \frac{p_1}{1-p_1}.
$$

3 极致梯度提升树 (XGBoost)

1. 梯度提升 GBDT 的损失函数利用自变量 f_{m-1} 的一阶导数信息进行梯度下降, 极致梯度提升则采用了二阶导数信息来指导弱学习器 h_m.

2. 对损失函数 $L(y, f_m(x))$ 关于变量 f_{m-1} 二阶展开可得:

$$
\sum_{i=1}^{N} L\Big(y_i, f_{m-1}(x_i) + h_m(x_i)\Big) \approx \sum_{i=1}^{N} \left(L\Big(y_i, f_{m-1}(x_i)\Big) + G_i h_m(x_i) + \frac{1}{2} H_i h_m(x_i)^2 \right),
$$

其中, G_i 是 $L(y, f_m(x))$ 关于变量 f_{m-1} 的一阶导数; H_i 是 $L(y, f_m(x))$ 关于变量 f_{m-1} 的二阶导数.

$$
G_i = \frac{\partial L(y_i, f_{m-1})}{\partial f_{m-1}}, \quad H_i = \frac{\partial^2 L(y_i, f_{m-1})}{\partial f_{m-1}^2}
$$

3. 为了让损失函数 $L(y, f_m)$ 最小化, 对 L 的二阶泰勒展开表达式中的 h_m 求导, 并且令导数值为, 则有

$$\frac{\partial L(y_i, f_{m_1}(x_i) + h_m(x_i))}{\partial h_m(x_i)} \approx G_i + H_i h_m(x_i) = 0,$$

因此,

$$h_m(x_i) = -\frac{G_i}{H_i}$$

4. 由第三点 XGBoost 是利用二阶导数的信息进行训练, 即训练集为:

$$S = \{(x_i, -\frac{G_i(x_i)}{H_i(x_i)}) \mid i = 1, 2, \cdots, N\}.$$

5. 除了使用二阶导数信息, XGboost 还对损失函数进行了正则化, 即加入了罚项表示模型复杂度, 防止过拟合, 同时, $L(y_i, f_{m-1}(x_i))$ 是常数项, 不影响最小值, 将其省去可得:

$$\begin{aligned}
&\sum_{i=1}^{N} L\Big(y_i, f_{m-1}(x_i) + h_m(x_i)\Big) \\
&\quad \approx \sum_{i=1}^{N} \left(G_i h_m(x_i) + \frac{1}{2} H_i h_m(x_i)^2\right) + \Omega(h_m) \\
&\quad \Omega(h_m) = \gamma T + \frac{1}{2}\lambda \|w\|^2.
\end{aligned}$$

6. 当使用 CART 作为树作为弱学习器 h_m 时,

$$h_m(x) = w_{q(x)},\ q : \mathbb{R}^n \to T, w \in \mathbb{R}^T,$$

其中:

(1) T 是树的叶节点数. q 将样本 x 映射成树的某个叶节点的索引（也就是某个叶节点的位置）.

(2) $w = (w_1, w_2, \cdots, w_T) \in \mathbb{R}^T$, 即 w 是树的叶节点对应的值. 所以 $h_m(x)$ 将 x 映射为树的某个叶节点的值. 所以必然存在某个整数 $1 \le j \le T$, 使得

$$h_m(x) = w_j.$$

(3) 将映射到同一个叶子节点的样本放在一起, 记为集合 I_j, I_j 表示将所有被 h_m 映射到叶节点 j 的样本点的集合:

$$I_j = \{x_i \mid h_m(x_i) = w_j\}.$$

即一个数据集 X, 就被划分成了 $I_1, I_2, \cdots, I_T$.

7. 因此, 从 $i=1$ 到 $i=N$ 的损失和可以写成每个叶节点的损失和:

$$\begin{aligned}
&\sum_{i=1}^{N} L\Big(y_i, f_{m-1}(x_i)+h_m(x_i)\Big) \\
&\quad\approx \sum_{i=1}^{N}\left(G_i h_m(x_i)+\frac{1}{2}H_i h_m(x_i)^2\right)+\gamma T+\frac{1}{2}\lambda\|w\|^2 \\
&\quad\approx \sum_{j=1}^{T}\left(\Big(\sum_{i\in I_j} G_i\Big)w_j+\frac{1}{2}\Big(\sum_{i\in I_j} H_i\Big)w_j^2\right)+\gamma T+\frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\
&\quad= \sum_{j=1}^{T}\left(\Big(\sum_{i\in I_j} G_i\Big)w_j+\frac{1}{2}\Big(\sum_{i\in I_j} H_i+\lambda\Big)w_j^2\right)+\gamma T
\end{aligned}$$

8. 对上面的损失函数 L 关于 w_j 求导, 并令 $L'=0$, 可得:

$$w_j^* = \frac{\sum_{i\in I_j} G_i}{\sum_{i\in I_j} H_i+\lambda}, \quad 1\le j\le T,$$

将 w_j^* 带入损失函数 L 的表达式, 得到 L 的取最小值表达式:

$$\min L = -\frac{1}{2}\sum_{j=1}^{T}\frac{\Big(\sum_{i\in I_j} G_i\Big)^2}{\sum_{i\in I_j} H_i+\lambda}+\gamma T,$$

9. 最优特征的选择 (最小化损失函数). XGBoost 采用贪心算法来选取分裂的特征和分裂节点. 设 I 是分裂前的某个节点, 进行分裂后得到两个叶节点 I_R, I_L, $I=I_L\cup I_R$. 损失函数的变化 ΔL:

$$\begin{aligned}
\Delta L &= L(I)-(L(I_R)+L(I_L)) \\
&= -\frac{1}{2}\frac{\Big(\sum_{i\in I} G_i\Big)^2}{\sum_{i\in I} H_i+\lambda}+\gamma-\left(-\frac{1}{2}\frac{\Big(\sum_{i\in I_L} G_i\Big)^2}{\sum_{i\in I_L} H_i+\lambda}+\gamma-\frac{1}{2}\frac{\Big(\sum_{i\in I_R} G_i\Big)^2}{\sum_{i\in I_R} H_i+\lambda}+\gamma\right) \\
&= \frac{1}{2}\left(\frac{\Big(\sum_{i\in I_L} G_i\Big)^2}{\sum_{i\in I_L} H_i+\lambda}+\frac{\Big(\sum_{i\in I_R} G_i\Big)^2}{\sum_{i\in I_R} H_i+\lambda}-\frac{\Big(\sum_{i\in I} G_i\Big)^2}{\sum_{i\in I} H_i+\lambda}\right)-\gamma,
\end{aligned}$$

10. 每次进行分裂时, 都计算该节点的 ΔL, 并取使得 ΔL 最大的特征为分裂点, 动态地生成一棵树.