提升

cloud

2016.12.26

1 概述

提升（boosting）方法是一种思想，可以应用于分类，也可以应用于回归。它的基本思想是先从初始训练集训练出一个弱学习器，再根据弱学习器的表现对训练样本的分布进行调整，使得先前弱学习器做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弱学习器；如此重复进行，直至弱学习器数目达到事先指定的值，最终将这些弱学习器进行加权结合为一个强学习器，主要关注降低偏差。

2 AdaBoost 算法

2.1 基本原理

AdaBoost（Adaptive Boosting）算法就是利用提升思想的分类算法，对于不同的提升方法，需要解决两个问题。第一个是在每一轮如何改变训练数据的权值或概率分布；第二是如何将弱分类器组合成一个强分类器。

对于第一个问题，AdaBoost 是提高被前一轮弱分类器错误分类样本的权值，而降低被正确分类样本的权值，使得分类错误的样本在后一轮分类器中得到更多的关注。对于第二个问题，AdaBoost 采用加权多数表决的方法，即加大分类误差率小的弱分类器的权值，使其在表决中起较大的作用，减小分类误差率大的弱分类器权值，使其在表决中起较小的作用。

2.2 算法实现

假设给定一个二分类的训练数据集如下：

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

$$x^{(i)} \in \mathcal{X} = R^n, y^{(i)} \in \mathcal{Y} = \{+1, -1\}, i = 1, 2, ..., N$$

AdaBoost 利用以下算法从训练数据中学习一系列弱分类器，并将这些弱分类器组合成一个强分类器。

（1）初始化训练数据的权值分布：

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

解释：初始时假设每个训练样本权值相同，即在基本分类器的学习中作用相同。

（2）对 $m = 1, 2, ..., M$

（a）使用具有权值分布 D_m 的训练数据集学习，得到基本分类器：

$$G_m(x) : \mathcal{X} \to \{-1, +1\}$$

（b）计算 $G_m(x)$ 在训练集上的分类误差率：

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

解释：在这里误差率等于分错样本的权值之和，这里有 $\sum_{i=1}^{N} w_{mi} = 1$。

（c）计算 $G_m(x)$ 的系数，即对应分类器的权值：

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

解释：α_m 表示 $G_m(x)$ 在最终分类器的重要性，当 $e_m \leqslant \frac{1}{2}$ 时，$\alpha_m \geqslant 0$，并且 α_m 随着误差率的减小而增大。

（d）更新训练数据集的权值分布：

$$D_{m+1} = (w_{m+1,1}, ..., w_{m+1,i}, ..., w_{m+1,N})$$
$$w_{m+1,i} = \frac{w_{m,i}}{Z_m} exp(-\alpha_m y^{(i)} G_m(x^{(i)})), \quad i = 1, 2, ..., N$$

在这里 Z_m 是规范化因子，它使 D_{m+1} 称为一个概率分布。

$$Z_m = \sum_{i=1}^{N} w_{mi} exp(-\alpha_m y^{(i)} G_m(x^{(i)}))$$

解释：如果样本被分类正确，那么 $y^{(i)} G_m(x^{(i)})$ 为 $+1$，就有 $w_{m+1,i} = \frac{w_{m,i}}{Z_m} exp(-\alpha_m)$，这样被正确分类的样本权值降低。同理被误分类的样本权值会提高，使得误分类样本在下一轮学习中起更大的作用。不改变所给的训练数据，而不断改变训练数据权值的分布，使得训练样本在基本分类器的学习中起不同的作用。

（3）构建基本分类器的线性组合：

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

得到最终分类器：

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

解释：$f(x)$ 将 M 个基本分类器的加权表决，系数 α_m 表示了基本分类器的重要性，在这里所有 α_m 的和不为 1。$f(x)$ 的符号决定实例 x 的类，$f(x)$ 的绝对值表示了分类的确信度。

2.3 损失函数

AdaBoost 算法可以看成模型是加法模型，损失函数为指数函数，学习算法为前向分步算法的二类分类学习方法。

因为传统通过损失函数求加法模型最优参数的方法复杂度较高，所以可以通过前向分步算法把加法模型进行分解，每一步只学习一个基函数及其系数，逐步逼近优化目标函数式。简单来说就说把工作进行分解，逐步迭代加和求解最优分类器。

下面来证明在加法模型中使用前向分步算法求解损失函数最小化方法等价于 AdaBoost 算法。

（1）首先加法模型为

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

损失函数是指数损失函数：

$$L(y, f(x)) = exp[-yf(x)]$$

（2）假设经过 $m-1$ 轮迭代前向分步算法已经得到 $f_{m-1}(x)$：

$$f_{m-1}(x) = \alpha_1 G_1(x) + ... + \alpha_{m-1} G_{m-1}(x)$$

在第 m 轮迭代得到 $\alpha_m, G_m(x)$，由此可知：

$$f_m(x) = f_{m-1}(x) + \alpha_m G_m(x) \tag{1}$$

（3）目标是使用前向分步算法得到 $\alpha_m, G_m(x)$ 使得 $f(x)$ 在训练数据集上的指数损失最小：

$$\begin{aligned}(\alpha_m, G_m(x)) &= arg \min_{\alpha, G} \sum_{i=1}^{N} exp[-y^{(i)}(f_{m-1}(x^{(i)}) + \alpha G(x^{(i)}))] \\ &= arg \min_{\alpha, G} \sum_{i=1}^{N} \overline{w}_{mi} exp[-y^{(i)} \alpha G(x^{(i)})]\end{aligned} \tag{2}$$

其中 $\overline{w}_{mi} = exp[-y^{(i)} f_{m-1}(x^{(i)})]$，因为 $\overline{w}_{mi}$ 不依赖于 α 和 G，所以与最小化无关，但其依赖于 $f_{m-1}(x)$ 随着每一轮迭代而发生改变。

（4）现要证实使 (2) 达到最小的 $\alpha_m^*, G_m^*(x)$ 就是 AdaBoost 算法所得到的 $\alpha_m, G_m(x)$。首先求解 $G_m(x)$，对于任意的 $\alpha > 0$，使 (2) 最小的 $G(x)$ 由下式得到：

$$G_m^*(x) = arg \min_G \sum_{i=1}^{N} \overline{w}_{mi} I(y^{(i)} \neq G(x^{(i)}))$$

此分类器 $G_m^*(x)$ 是使得第 m 轮加权训练数据分类误差率最小的基本分类器。

然后再求 α_m^*，首先式 (2) 可以写成：

$$\begin{aligned}
&\min_{\alpha, G} \sum_{i=1}^{N} \overline{w}_{mi} exp[-y^{(i)} \alpha G(x^{(i)})] \\
&= \sum_{y^{(i)} = G_m(x^{(i)})} \overline{w}_{mi} e^{-\alpha} + \sum_{y^{(i)} \neq G_m(x^{(i)})} \overline{w}_{mi} e^{\alpha} \\
&= e^{-\alpha} \sum_{i=1}^{N} \overline{w}_{mi} - e^{-\alpha} \sum_{i=1}^{N} \overline{w}_{mi} I(y^{(i)} \neq G(x^{(i)})) + e^{\alpha} \sum_{i=1}^{N} \overline{w}_{mi} I(y^{(i)} \neq G(x^{(i)})) \\
&= (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{N} \overline{w}_{mi} I(y^{(i)} \neq G(x^{(i)})) + e^{-\alpha} \sum_{i=1}^{N} \overline{w}_{mi}
\end{aligned}$$

将 G_m^* 代入上式，对 α 求导为 0，即得到使上式最小的 α 如下。其中 $e_m = \sum_{i=1}^{N} w_{mi} I(G_m(x^{(i)}) \neq y^{(i)})$ 为分类误差率。

$$\alpha_m^* = \frac{1}{2} log \frac{1 - e_m}{e_m}$$

这里的 α_m^* 与前面 AdaBoost 的 α_m 是一致的。

（5）由式 (1) 以及 $\overline{w}_{mi} = exp[-y^{(i)} f_{m-1}(x^{(i)})]$ 可得：

$$\overline{w}_{m+1,i} = \overline{w}_{m,i} exp(-\alpha_m y^{(i)} G_m(x^{(i)}))$$

这与 AdaBoost 样本权值的更新只差规范化因子，因而是等价的。

3 提升树

3.1 基本定义

提升树是以分类树或回归树为基本分类器的提升方法，提升树可以表示为决策树的加法模型如下。在这里树就是基分类器，其中 $T(x; \Theta_m)$ 表示决策树，Θ_m 表示决策树的参数，M 为树的个数。

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

提升树算法采用前向分步算法，首先初始时提升树为 $f_0(x)=0$，第 m 步的模型是 $f_m(x)=f_{m-1}(x)+T(x;\Theta_m)$，其中当前模型是 $f_{m-1}(x)$，通过经验风险极小化确定下一棵决策树的参数 Θ_m。

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

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，包括用平方误差损失函数的回归问题，用指数损失函数的分类问题，以及用一般损失函数的一般决策问题。

3.2 回归问题的提升树算法

前面说到使用平方误差损失函数的回归问题，在这里我们用回归问题的提升树算法为例。当采用平方误差损失函数时：

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

在这里 $r=y-f_{m-1}(x)$ 表示当前模型拟合数据的残差，所以回归模型的提升树算法每一轮只需要拟合当前模型的残差。假设给定一个回归模型的训练数据集是

$T=\{(x^{(1)},y^{(1)}),(x^{(2)},y^{(2)}),...,(x^{(N)},y^{(N)})\}, x^{(i)}\in\mathcal{X}\subseteq R^n, y^{(i)}\in\mathcal{Y}\subseteq R, i=1,2,...,N$

下面给出回归问题的提升树算法。

（1）初始化 $f_0(x)=0$

解释：可以理解为初始的时候所有样本 x 对应的 y 值都是 0。

（2）对 $m=1,2,...,M$

（a）计算残差 $r_{mi}=y^{(i)}-f_{m-1}(x^{(i)}) \quad i=1,2,...,N$

解释：就是 x 的真实值和上一轮预测的结果之间的差值

（b）拟合残差 r_{mi} 学习一个回归树，得到 $T(x;\Theta_m)$

解释：用所有 x 对应的残差按照损失最小拟合一棵树。

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

解释：将新产生的树加和到原来的模型中。

（3）得到回归问题的提升树 $f_M(x)=\sum_{i=1}^{M}T(x;\Theta_m)$。

回归树是通过每一轮的残差表来划分训练集，每一轮的划分都要使得损失最小也即残差平方和最小，最终的回归树的形态相当于决策树每一层通过熵最大的属性划分变成 x 值的范围划分，根节点由类别变成了具体的值。例题可以参考统计学习方法 149 页，理解起来更加直观。

3.3 梯度提升树（GBDT）

对于一般损失函数而言，每一步的优化无法像上述的方法那么容易，所以针对这一问题梯度提升算法就出现了，在实际中 GBDT 就是使用梯度提升算法的提升树模型。梯度提升算法是利用最速下降法的近似方法，即利用损失函数的负梯度在当前模型的值

$$-[\frac{\partial L(y, f(x^{(i)}))}{\partial f(x^{(i)})}]_{f(x)=f_{m-1}(x)}$$

作为回归问题提升树算法中残差的近似值，拟合一棵回归树。在这里我想说在使用树模型的算法中，分类问题和回归问题是不分家的，因为回归可以看成每个 x 对应的 y 是一个类别，只不过是将分类问题里的类别标记变成了具体的数字。

假设给定一个回归模型的训练数据集是

$T = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}, x^{(i)} \in \mathcal{X} \subseteq R^n, y^{(i)} \in \mathcal{Y} \subseteq R, i = 1, 2, ..., N$

损失函数是 $L(y, f(x))$，下面给出回归问题的梯度提升算法。

（1）初始化 $f_0(x) = arg \min_c \sum_{i=1}^{N} L(y^{(i)}, c)$

解释：估计使损失函数极小化的常数值，它是只有一个根结点的树。

（2）对 $m = 1, 2, ..., M$

（a）计算残差 $r_{mi} = -[\frac{\partial L(y, f(x^{(i)}))}{\partial f(x^{(i)})}]_{f(x)=f_{m-1}(x)} \quad i = 1, 2, ..., N$

解释：对于一般损失函数，这个就是残差的近似值。

（b）拟合残差 r_{mi} 学习一个回归树，得到第 m 棵树的叶结点区域 $R_{mj} \quad j = 1, 2, ..., J$

解释：每一叶结点区域代表一个值。

（c）计算 $c_{mj} = arg \min_c \sum_{x^{(i)} \in R_{mj}} L(y^{(i)}, f_{m-1}(x^{(i)}) + c) \quad j = 1, 2, ..., J$

解释：估计这个叶结点区域代表的值使得损失极小。

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

（3）得到回归问题的提升树 $f_M(x) = \sum_{i=1}^{M} \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$。

在这里的梯度提升算法是将上节的提升树算法中 $T(x; \Theta_m)$ 详细化。

3.4 xgboost

xgboost 是 GBDT 的进一步延伸，它们之间有很多区别。详情和相关数学推导请参考以下网址。

https://www.zhihu.com/question/41354392

http://www.52cs.org/?p=429

因为重要性的原因决定将 xgboost 单独作为一章，在这里就不做赘述了。

4 参考文献

1. 李航博士的统计学习方法
2. 周志华的机器学习