提升方法

> 在分类问题中，提升方法通过改变训练样本的权重，学习出多个分类器，并将这些分类器进行线性组合，提高分类的性能

8.1 提升方法AdaBoost算法

8.1.1 提升方法的基本思路

- **思路**：对于一个复杂任务，将多个专家的判断进行适当的综合所得的判断要比其中任何一个专家单独的判断好；
- **强可学习**（strongly learnable）：在概率近似正确（probably approximately correct，PAC）学习的框架中，一个概念（一个类）如果存在一个多项式的学习算法能够学习它，并且正确率很高，称这个概念是强可学习的；
- **弱可学习**（weakly learnable）：一个概念如果存在一个多项式的学习算法能够学习它，学习的正确率仅比随机猜测略好，那么称这个概念是弱可学习的；
- 已证明：*强可学习与弱可学习是等价的*，在PAC框架下，一个概念是强可学习的充要条件是它是弱可学习的；

- 对于分类问题，给定一个训练集，求比较粗糙的分类规则（弱分类器）比求精确的分类规则（强分类器）容易，提升方法反复学习得到一系列弱分类器，然后组合成一个强分类器；
- 大多数方法是改变*训练数据的概率分布*（训练数据的权值分布），然后调用弱学习算法；

- 两个问题：1）每一轮如何改变训练数据的权值或概率分布；2）如何把弱分类器组合成强分类器；
- AdaBoost：1）提高那些被前一轮弱分类器错误分类样本的权值，而降低正确分类的权值；2）采用*加权多数表决*方法，即加大分类误差率小的弱分类器的权值，使其在表决中发挥较大作用；

8.1.2 AdaBoost算法

- 假设给定一个二分类的训练数据集T；
- **AdaBoost算法**：
  1. **输入**训练数据集T，弱学习算法；
  2. 初始化训练数据的权值分布：

$$D_1 = (w_{11}, \dots, w_{1i}, \dots, w_{1N}), w_{1i} = \frac{1}{N}, i = 1, 2, \dots, N \tag{27}$$

  3. 对于$m = 1, 2, \dots, M$：
     1. 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：

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

     2. 计算$G_m(x)$在训练数据上的分类误差率：

$$e_m = P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i) \tag{2}$$

     3. 计算$G_m(x)$的系数，这里对数是自然对数：

$$\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m} \tag{3}$$

     4. 更新训练数据集的权值分布：

$$D_{m+1} = (w_{m+1,1}, \dots, w_{m+1,i}, \dots, w_{m+1,N}), \tag{4}$$
$$w_{m+1,i} = \frac{w_{mi} \exp(-\alpha_m y_i G_m(x_i))}{Z_m}, i = 1, 2, \dots, N$$

其中$Z_m = \sum_{i=1}^{N} w_{mi} \exp(-\alpha_m y_i G_m(x_i))$是归一化因子，使得$D_{m+1}$是一个概率分布；

4. 构建基本分类器的线性组合：

$$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x) \tag{5}$$

得到最终的分类器：

$$G(x) = \text{sign}(f(x)) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right) \tag{6}$$

- 算法解释：
  1. 初始化时假设具有均匀的权值分布，保证了在第一步中能够在*原始数据上学习*基本分类器；
  2. 计算基本分类器$G_m(x)$分类误差率时，$e_m = P(G_m(x_i) \neq y_i) = \sum_{G_m(x_i) \neq y_i} w_{mi}$实质上是被$G_m(x)$*误分类样本的权值之和*，因此权值分布与分类器误差率有关系；
  3. 基本分类器的系数α_m表示$G_m(x)$在最终分类器中的重要性：当$e_m \leq \frac{1}{2}$时，$\alpha_m \geq 0$，且α_m随着e_m的减少而增大，也即分类误差率越小的基本分类器在最终分类器的作用更大；
  4. 更新训练集的权值分布时，可以写成

$$w_{m+1,i} = \begin{cases} \frac{w_{mi}}{Z_m} e^{-\alpha_m}, & G_m(x_i) = y_i \\ \frac{w_{mi}}{Z_m} e^{\alpha_m}, & G_m(x_i) \neq y_i \end{cases} \tag{7}$$

  也就是说被基本分类器误分类的样本的权值得以扩大，而正确分类的却被缩小，相比较误分类样本的权值被放大$e^{2\alpha_m} = \frac{e_m}{1-e_m}$倍；
  5. 线性组合$f(x)$实现了M个基本分类器的加权表决，系数表示了各个基本分类器的重要性，但是所有α_m之和不等于1，$f(x)$的符号决定了实例x的类，$f(x)$的绝对值表示分类的确信度；

8.2 AdaBoost算法的训练误差分析

- AdaBoost最基本的性质是能够在学习过程中不断减少训练误差，即在训练数据上的分类误差率；
- **AdaBoost的训练误差界**（定理）：AdaBoost算法最终分类器的训练误差界为：

$$\frac{1}{N}\sum_{i=1}^{N} I(G(x_i) \neq y_i) \leq \frac{1}{N}\sum_{i} \exp(-y_i f(x_i)) = \prod_{m} Z_m \tag{8}$$

此定理说明，可以在每一轮选取适当的G_m使得Z_m最小，从而使得训练误差下降得最快；

- **二分类问题AdaBoost的训练误差界**（定理）：

$$\prod_{m=1}^{M} Z_m = \prod_{m=1}^{M} [2\sqrt{e_m(1-e_m)}] = \prod_{m=1}^{M} \sqrt{(1-4\gamma_m^2)} \leq \exp\left(-2\sum_{m=1}^{M}\gamma_m^2\right) \tag{9}$$

其中$\gamma_m = \frac{1}{2} - e_m$；

- **推论**：如果存在$\gamma > 0$，对所有的m有$\gamma_m \geq \gamma$，则

$$\frac{1}{N}\sum_{i=1}^{N} I(G(x_i) \neq y_i) \leq \exp(-2M\gamma^2) \tag{10}$$

则表明在此条件下，AdaBoost的训练误差是*以指数速率下降*的；

- AdaBoost不需要知道下界γ，其具有适应性，能够适应弱分类器各自的训练误差率，因此叫Adaptive Boosting；

8.3 AdaBoost算法的解释

> AdaBoost算法可以认为其模型是加法模型、损失函数为指数函数、学习算法为前向分步算法时的二分类学习方法

8.3.1 前向分步算法

- **加法模型**（additive model）：

$$f(x) = \sum_{m=1}^{M} \beta_m b(x;\gamma_m) \tag{11}$$

其中$b(x;\gamma_m)$为基函数，γ_m为基函数的参数，β_m为基函数的系数，AdaBoost算法对基本分类器的线性组合就是一个加法模型；

- 给定训练数据及损失函数$L(y,f(x))$条件下，学习加法模型成为经验风险极小化即损失函数极小化的问题：

$$\min_{\beta_m,\gamma_m}\sum_{i=1}^{N}L\left(y_i,\sum_{m=1}^{M}\beta_m b(x_i;\gamma_m)\right) \tag{12}$$

- 用**前向分步算法**（forward stagewise algorithm）进行求解的想法是：因为是加法模型，如果能够从前向后，*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，那就可以简化优化的复杂度，即每步只优化如下损失函数：

$$\min_{\beta,\gamma}\sum_{i=1}^{N}L\left(y_i,\beta b(x_i;\gamma)\right) \tag{13}$$

- **前向分步算法**：
  1. **输入**训练数据集T，损失函数$L(y,f(x))$，基函数集$\{b(x;\gamma)\}$；
  2. 初始化$f_0(x)=0$；
  3. 对于$m=1,2,\dots,M$：
     1. 极小化损失函数，得到参数β_m,γ_m：

$$(\beta_m,\gamma_m)=\arg\min_{\beta,\gamma}\sum_{i=1}^{N}L\left(y_i,f_{m-1}(x_i)+\beta b(x_i;\gamma)\right) \tag{14}$$

     2. 更新

$$f_m(x)=f_{m-1}(x)+\beta_m b(x;\gamma_m) \tag{15}$$

  4. 得到加法模型

$$f(x)=f_M(x)=\sum_{m=1}^{M}\beta_m b(x;\gamma_m) \tag{16}$$

- 因此，前向分步算法就把同时求解所有参数的优化为题简化为逐次求解各个β_m,γ_m的优化问题；

8.3.2 前向分步算法与AdaBoost

- **定理**：AdaBoost算法是前向分步加法算法的特例，这时，模型是由基本分类器组成的加法模型，损失函数是指数函数；（见证明）

8.4 提升树

> 提升树是以分类树或者回归树为基本分类器的提升方法，是统计学习中性能最好的方法之一

8.4.1 提升树模型

- 以决策树为基函数的提升方法称为**提升树**（boosting tree），对于分类问题决策树是二叉树分类树，对于回归问题决策树是二叉回归树；
- **决策树桩**（decision stump）：由一个根结点直接连接两个叶结点的简单决策树；
- 提升树模型可以表示为决策树的加法模型：

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m) \tag{17}$$

其中$T(x;\Theta_m)$表示决策树，Θ_m为决策树参数，M是树的个数；

8.4.2 提升树算法

- 提升树算法采用前向分步算法；
- 针对不同问题的提升树学习算法，主要区别在于使用**损失函数不同**：
  - 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；
  - 用一般损失函数的一般决策问题；
- 对于二分类问题，提升树算法只需将AdaBoost算法的基本分类器限制为二分类树即可；

- **回归问题的提升树**：给定训练集T，根据第五章的讨论，如果将输入空间$\mathcal{X}$划分为J个互不相交的区域$R_1, R_2, \ldots, R_J$，并且在每一个区域上确定输出常量c_j，那么树可以表示为：

$$T(x;\Theta) = \sum_{j=1}^{J} c_j I(x \in R_j) \tag{18}$$

其中参数$\Theta=\{(R_1,c_1),\dots,(R_J,c_J)\}$表示树的区域划分和各个区域的常数，$J$是回归树的复杂度也即叶结点个数；

- 在前向分步算法求解模型参数时，采用平方误差损失函数$L(y,f(x))=(y-f(x))^2$，在第m步时其损失变为：

$$\begin{aligned}&L(y,f_{m-1}+T(x;\Theta_m))\\&=(y-f_{m-1}(x)-T(x;\Theta_m))^2\\&=(r-T(x;\Theta_m))^2\end{aligned}\tag{19}$$

其中$r=y-f_{m-1}(x)$是*当前模型拟合数据的残差*（residual）；所有对于回归问题的提升树算法，只需简单地拟合当前模型的残差；

- **回归问题的提升树算法**：
  1. **输入**训练集T；
  2. 初始化$f_0(x)=0$；
  3. 对于$m=1,2,\dots,M$：
     1. 计算残差：

$$r_{mi}=y_i-f_{m-1}(x_i),i=1,2,\dots,N\tag{20}$$

     2. 拟合残差r_{mi}学习一个回归树，得到$T(x;\Theta_m)$；
     3. 更新$f_m(x)=f_{m-1}(x)+T(x;\Theta_m)$；
  4. 得到回归问题提升树

$$f_M(x)=\sum_{m=1}^{M}T(x;\Theta_m)\tag{21}$$

- **注意**：树的加法模型其实是在继续划分区域；

8.4.3 梯度提升

- 以上提升树算法使用*加法模型*与*前向分步算法*实现学习过程，对于一般损失函数来说，每一步的优化并不像平方损失和指数损失那么容易；
- **梯度提升**（gradient boosting）算法：利用最速下降法的近似方法，关键是利用*损失函数的负梯度在当前模型的值*

$$-\left[\frac{\partial L(y_i,f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}\tag{22}$$

作为回归问题提升树算法中的*残差的近似值*，拟合一个回归树；

- **梯度提升算法**：
  1. **输入**训练数据集T，损失函数$L(y,f(x))$；
  2. 初始化

$$f_0(x)=\arg\min_c\sum_{i=1}^N L(y_i,c) \tag{23}$$

  3. 对于$m=1,2,\dots,M$：
     1. 计算残差的近似值：

$$r_{mi}=-\left[\frac{\partial L(y_i,f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)},i=1,2,\dots,N \tag{24}$$

     2. 对r_{mi}拟合一个回归树，得到第m棵树的叶结点区域$R_{mj},j=1,2,\dots,J$；
     3. 计算

$$c_{mj}=\arg\min_c\sum_{x_i\in R_{mj}}L(y_i,f_{m-1}(x_i)+c),j=1,2,\dots,J \tag{25}$$

     4. 更新$f_m(x)=f_{m-1}(x)+\sum_{j=1}^J c_{mj}I(x\in R_{mj})$；
  4. 得到回归树

$$f(x)=f_M(x)=\sum_{m=1}^M\sum_{j=1}^J c_{mj}I(x\in R_{mj}) \tag{26}$$

- 算法解释：
  1. 初始化时估计使得损失函数极小化的常数值，它是一个根结点的树；
  2. 用损失函数的负梯度在当前模型的值作为残差的估计，对于平方误差就是残差，对于一般损失函数，就是残差的近似值；
  3. 对残差近似值拟合，估计得到回归树的叶结点区域；
  4. 利用线性搜索估计叶结点区域的值c_{mj}，使得损失函数极小化；