AdaBoost 学习笔记

李向阳　d1142845997@gmail.com

目录

1 引入

本次介绍我们进入一个新的领域, 通常称之为集成学习 (Ensemble Learning), 其实也就是通过构建并结合多个学习器来完成学习的任务. 一般情况下, 集成学习是将多个"弱学习器"进行结合, 从而获得比单一学习器更显著优越的泛化性能. 其中一种很重要的方法便是 Boosting 方法.

Boosting 是一族可将弱学习器提升为强学习器的算法. 这族算法的工作机制类似: 先从初始训练样本中训练出一个基学习器, 再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, 如此反复进行, 直至基学习器达到事先指定的值 T, 最终将这 T 个基学习器进行加权组合.

Boosting 族算法最著名的代表便是 AdaBoost(Adaptive Boosting), 它是针对二分类问题的, 下面我们就来介绍 AdaBoost 算法.

2 基本模型

沿用之前的记号, 假设我们的样本是 $(\boldsymbol{x}, y)$, 其中 y 是类标签, 用 $y = \pm 1$ 区分正负类. 也就是我们的数据集为 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)\}$, 其中 $y_i \in \{-1, 1\}$. 假设训练的轮数为 T, 即最终要训练出 T 个基分类器. 设最终的分类器为 $h(\boldsymbol{x})$, 第 t 步迭代产生的基分类器为 $h_t(\boldsymbol{x})$.

我们先来说一下算法的流程.

(1) 初始化训练数据的权值分布

$$\mathcal{D}_1(\boldsymbol{x}) = (w_{11}, \cdots, w_{1i}, \cdots, w_{1m}), w_{1i} = \frac{1}{m}, i = 1, 2, \cdots, m$$

(2) 接下来对 $t = 1, 2, \cdots, T$ 进行迭代:

(a) 使用具有权值分布 $\mathcal{D}_t(\boldsymbol{x})$ 的训练数据集学习, 得到基本分类器

$$h_t(\boldsymbol{x}) :\to \{-1, 1\}$$

(b) 计算 $h_t(\boldsymbol{x})$ 在训练数据集的分类误差率, 注意是加权计算

$$e_t = P(h_t(\boldsymbol{x}_i) \neq y_i) = \sum_{i=1}^{m} w_{ti} \mathbb{I}(h_t(\boldsymbol{x}_i) \neq y_i)$$

(c) 计算基分类器 $h_t(\boldsymbol{x})$ 的系数

$$\alpha_t = \frac{1}{2} \ln \frac{1 - e_t}{e_t}$$

(d) 更新训练数据集的权值分布

$$\mathcal{D}_{t+1}(\boldsymbol{x}) = (w_{t+1,1}, \cdots, w_{t+1,i}, \cdots, w_{t+1,m})$$
$$w_{t+1,i} = \frac{w_{ti}}{Z_t} \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$$

其中 Z_t 是规范化因子

$$Z_t = \sum_{i=1}^{m} w_{ti} \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$$

它使得 $\mathcal{D}_{t+1}(\boldsymbol{x})$ 成为一个概率分布.

(3) 构建基分类器的加权线性组合得到最终分类器

$$h(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$$

以上便是 AdaBoost 算法的大致流程. 我们稍微解释一下.

(1) 初始化时等均值分布, 这样可以保证第 1 步能够在原始数据上学习到基分类器 $h_1(\boldsymbol{x})$.

(2) 不断迭代学习出剩下的基分类器, 在每一轮 $t = 1, 2, \cdots, T$ 上顺次执行

(a) 使用当前的数据分布 $\mathcal{D}_t(\boldsymbol{x})$, 学习到基分类器 $h_t(\boldsymbol{x})$.

(b) 计算基分类器 $h_t(\boldsymbol{x})$ 在加权训练数据集上的分类误差率

$$e_t = P(h_t(\boldsymbol{x}_i) \neq y_i) = \sum_{h_t(\boldsymbol{x}_i) \neq y_i} w_{ti}$$

其中 w_{ti} 表示第 t 轮迭代中第 i 个样本的权值, 满足 $\sum\limits_{i=1}^{m} w_{ti} = 1$. 这表明, $h_t(\boldsymbol{x})$ 在加权的训练数据集上的分类误差率是被 $h_t(\boldsymbol{x})$ 误分类样本的权值之和, 由此可以看出数据权值分布 $\mathcal{D}_t(\boldsymbol{x})$ 与基分类器 $h_t(\boldsymbol{x})$ 的分类误差率的关系.

(c) 计算基分类器 $h_t(\boldsymbol{x})$ 的系数 α_t. 注意 e_t 表示 $h_t(\boldsymbol{x})$ 在最终分类器中的重要性, 而且当 $e_t \leqslant \frac{1}{2}$ 时, 才有 $\alpha_t \geqslant 0$, $e_t \geqslant \frac{1}{2}$ 的情况后面会补充说明. 可以看出, α_t 随着 e_t 的减小而增大, 所以分类误差率越小的基分类器在最终分类器中的作用越大.

(d) 更新训练数据的权值分布为下一轮迭代做准备. 由于 y_i 和 $h_t(\boldsymbol{x}_i)$ 的取值都是 1 或者 -1, 因此权值更新式可改写为

$$w_{t+1,i} = \begin{cases} \frac{w_{ti}}{Z_t} \exp(-\alpha_t), & h_t(\boldsymbol{x}_i) = y_i \\ \frac{w_{ti}}{Z_t} \exp(\alpha_t), & h_t(\boldsymbol{x}_i) \neq y_i \end{cases}$$

可以看出, 被基分类器 $h_t(\boldsymbol{x})$ 误分类样本的权值得以扩大, 而被正确分类的样本的权值得以缩小. 因此, 误分类样本在下一轮的学习中会起更大的作用. 不改变所给的训练数据, 而不断改变训练数据权值的分布, 使得训练数据在基分类器的学习中起不同的作用, 这正是 AdaBoost 的一个特点.

关于权值更新公式, 这里额外提一点, 权值更新式即为

$$w_{t+1,i} = w_{ti} \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$$

然后将 $w_{t+1,i}$ 归一化即可. 注意到

$$y_i \cdot h_t(\boldsymbol{x}_i)) = 1 - 2 \cdot \mathbb{I}(h_t(\boldsymbol{x}_i) \neq y_i)$$

代入可得

$$w_{t+1,i} = w_{ti} \cdot \exp(-\alpha_t) \cdot \exp(2\alpha_t \mathbb{I}(h_t(\boldsymbol{x}_i) \neq y_i)$$

由于因子 $\exp(-\alpha_t)$ 与 n 无关, 在归一化时不起作用, 因此权值更新公式可写为

$$w_{t+1,i} = w_{ti} \cdot \exp(2\alpha_t \mathbb{I}(h_t(\boldsymbol{x}_i) \neq y_i))$$

在有的资料中, 计算 α_t 时略去了 $\frac{1}{2}$, 即

$$\alpha_t = \ln \frac{1 - e_t}{e_t}$$

因此权值更新公式变成了

$$w_{t+1,i} = w_{ti} \cdot \exp(\alpha_t \mathbb{I}(h_t(\boldsymbol{x}_i) \neq y_i))$$

然后将 $w_{t+1,i}$ 归一化即可. 当然, 也有的资料中中不在这一步进行归一化, 而是在计算分类误差率 e_t 的时候进行归一化, 都是可以的.

(3) 最后的分类器 $h(\boldsymbol{x})$ 是基分类器的加权表决. 系数 α_t 表示了基分类器 $h_t(\boldsymbol{x})$ 的重要性. 注意一点, 所有的 α_t 之和并不一定是 1.

下面我们举一个简单的例子, 该例子来自李航的《统计学习方法》.

例 2.1. 我们的训练样本如表1所示, 为了方便, 把表横过来写了. 同时也只有一个特征变量, 所以就简记为 x 了. 假设弱分类器由 $x < v$ 或者 $x > v$ 产生, 其中阈值 v 使该分类器在训练数据集上的分类误差率最低, 试用 *AdaBoost* 算法学习出一个强分类器.

表 1: 训练数据表

ID	ID1	ID2	ID3	ID4	ID5	ID6	ID7	ID8	ID9	ID10
x	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1

解：初始化数据的权值分布

$$\mathcal{D}_1(\boldsymbol{x}) = (w_{11}, w_{12}, \cdots, w_{1,10})$$
$$w_{1,i} = 0.1, i = 1, 2 \cdots, 10$$

对 $t = 1$, 开始第一轮迭代

(a) 在权值分布为 $\mathcal{D}_1(\boldsymbol{x})$ 的训练数据上, 容易看出阈值 v 取 2.5 时分类误差率可达到最低, 比如取

$$h_1(\boldsymbol{x}) = \begin{cases} 1, & x < 2.5 \\ -1, & x > 2.5 \end{cases}$$

此时仅有 3 个误分类点, 实际上阈值取为 8.5 也可以, 即 $x < 8.5$ 时令 y 取 1, $x > 8.5$ 时令 y 取 -1, 此时也是有 3 个误分类点, 误差率均为 0.3. 这里任取一个, 不妨取为 2.5.

(b) 此时有 3 个误分类点, 误差率为 $e_1 = 3 \times 0.1 = 0.3$.

(c) 计算 $h_1(\boldsymbol{x})$ 的系数

$$\alpha_1 = \frac{1}{2} \ln \frac{1 - e_1}{e_1} = 0.4236$$

(d) 更新训练数据的权值分布

$$\mathcal{D}_2(\boldsymbol{x}) = (w_{21}, \cdots, w_{2i}, \cdots, w_{2,10})$$
$$w_{2i} = \frac{w_{1i}}{Z_1} \exp(-\alpha_1 y_i h_1(\boldsymbol{x}_i)), i = 1, 2, \cdots, 10$$
$$\mathcal{D}_2(\boldsymbol{x}) = (0.0715, 0.0715, 0.0715, 0.0715, 0.0715, 0.0715,$$
$$0.1666, 0.1666, 0.1666, 0.0715)$$

此时第一轮迭代结束, 可以看出, 取值为“6, 7, 8”的样本被分错了, 所以它们的权值由之前的 0.1 增大到 0.1666, 而其他样本被分类正确, 所以它们的权值由之前的 0.1 减小到 0.0715. 此时分类器 $\text{sign}[0.4236 h_1(\boldsymbol{x})]$ 在训练数据集上有 3 个误分类点.

对 $t = 2$, 开始第二轮迭代

(a) 在权值分布为 $\mathcal{D}_2(\boldsymbol{x})$ 的训练数据上, 可计算出阈值 v 取 8.5 时分类误差率达到最低, 即取

$$h_2(\boldsymbol{x}) = \begin{cases} 1, & x < 8.5 \\ -1, & x > 8.5 \end{cases}$$

事实上, 若阈值仍取为 2.5, 即 $x < 2.5$ 时取 1, $x > 2.5$ 时取 -1, 则还是"6, 7, 8"被分错, 误差率为 0.1666×3. 再比如将阈值取为 5.5, 即 $x > 5.5$ 时取 1, $x < 5.5$ 时取 -1, 则是"0, 1, 2, 9"被分错, 误差率为 $0.0715 \times 3 + 0.0715$. 而将阈值取为 8.5 时, 即 $x < 8.5$ 时取 1, $x > 8.5$ 时取 -1, 则是"3, 4, 5"被分错, 误差率为 0.0715×3, 此时的误差率最小.

(b) 此时有 3 个误分类点, 误差率为 $e_2 = 3 \times 0.0715 = 0.2143$.

(c) 计算 $h_2(\boldsymbol{x})$ 的系数

$$\alpha_2 = \frac{1}{2} \ln \frac{1 - e_2}{e_2} = 0.6496$$

(d) 更新训练数据的权值分布

$$\mathcal{D}_3(\boldsymbol{x}) = (0.0455, 0.0455, 0.0455, 0.1667, 0.1667, 0.1667,$$
$$0.1060, 0.1060, 0.1060, 0.0455)$$

此时第二轮迭代结束. 分类器 $\text{sign}[0.4236h_1(\boldsymbol{x}) + 0.6496h_2(\boldsymbol{x})]$ 在训练数据集上有 3 个误分类点.

对 $t = 3$, 开始第三轮迭代

(a) 在权值分布为 $\mathcal{D}_3(\boldsymbol{x})$ 的训练数据上, 可计算出阈值 v 取 5.5 时分类误差率达到最低, 即取

$$h_3(\boldsymbol{x}) = \begin{cases} 1, & x > 5.5 \\ -1, & x < 5.5 \end{cases}$$

事实上, 若阈值取为 2.5, 即 $x < 2.5$ 时取 1, $x > 2.5$ 时取 -1, 则还是"6, 7, 8"被分错, 误差率为 0.1060×3. 若将阈值取为 5.5, 即 $x > 5.5$ 时取 1, $x < 5.5$ 时取 -1, 则是"0, 1, 2, 9"被分错, 误差率为 $0.0455 \times 3 + 0.0455$. 而将阈值取为 8.5 时, 即 $x < 8.5$ 时取 1, $x > 8.5$ 时取 -1, 则是"3, 4, 5"被分错, 误差率为 0.1667×3, 因此阈值取为 5.5 误差率最低.

(b) 此时有 4 个误分类点, 误差率为 $e_3 = 4 \times 0.0455 = 0.1820$.

(c) 计算 $h_3(\boldsymbol{x})$ 的系数

$$\alpha_3 = \frac{1}{3} \ln \frac{1 - e_3}{e_3} = 0.7514$$

(d) 更新训练数据的权值分布

$$\mathcal{D}_3(\boldsymbol{x}) = (0.125, 0.125, 0.125, 0.102, 0.102, 0.102, 0.065, 0.065, 0.065, 0.125)$$

此时第三轮迭代结束, 而且此时分类器 $\text{sign}[0.4236h_1(\boldsymbol{x})+0.6496h_2(\boldsymbol{x})+0.7514h_3(\boldsymbol{x})]$ 在训练数据集上的误分类点的个数为 0.

于是可得最终的分类器为

$$h(\boldsymbol{x}) = \text{sign}[0.4236h_1(\boldsymbol{x}) + 0.6496h_2(\boldsymbol{x}) + 0.7514h_3(\boldsymbol{x})]$$

当然, 我们也可以一开始就规定训练的轮数.

□

上面我们介绍了 AdaBoost 算法. 其实一般的 Boosting 算法都是要求学习器能对特定的数据分布进行学习. 这里一般也是通过“重赋权法”(re-weighting) 实施的, 即在训练过程的每一轮中, 根据样本分布为每一个训练样本重新赋予一个权重. 我们知道, 很多算法是可以给样本赋予权重的, 但是对于那些无法接受带权样本的基学习算法呢?

这时我们可以通过“重采样法”(re-sampling) 来处理, 即在每一轮学习中, 根据样本分布对训练数据重新进行采样, 再用重采样而得的样本集对基学习器进行训练. 其实, 一般而言, “重赋权法”和“重采样法”没有显著的优劣差别.

下面解释一下上面的一个疑问之处, 就是在迭代中, 我们要求误差率 $e_t \leqslant \frac{1}{2}$, 这其实是检查当前的基学习器是否比随机猜测好. 一旦这个条件不满足, 则当前的基学习器即被抛弃, 并且迭代过程也就终止了. 在此种情形下, 可能初始设置的学习轮数 T 还远未达到, 这就可能导致最终集成中只包含很少的基学习器而性能不佳. 此时一个解决办法是采用上面的“重采样法”, 即在抛弃不满足条件的当前基学习器后, 根据当前分布重新对训练样本进行采样, 再基于新的采样结果重新训练出基学习器, 从而使得学习过程可以持续到预设的 T 轮完成.

AdaBoost 算法中采用的基分类器一般采用简单的决策树, 比如上面例子中的基分类器 $x < v$ 或 $x > v$ 就可以看做是一个根节点直接连接两个叶节点的简单决策树, 也就是所谓的决策树桩 (decision stump). 当然, 我们也可以采用其他分类器来作为 AdaBoost 算法的基分类器.

3 模型的推导

3.1 AdaBoost 变体

3.2 模型推导

4 关于 AdaBoost 的补充

4.1 前向分布算法

推导 AdaBoost 算法还有很多其他途径, 比如可将 AdaBoost 算法看成是模型为加法模型、损失函数为指数函数、学习算法为前向分布算法的二分类学习算法. 为了下面讲述梯度提升的方便, 这里我们介绍一下前向分布算法.

为了能够抽象的讨论, 我们将每一个分类器或者模型记为 $b(\boldsymbol{x};\boldsymbol{\gamma})$, 称为基函数, 其中 $\boldsymbol{\gamma}$ 为基函数的参数. 假设有 T 个基学习器, 考虑加权组合的加法模型 (additive model)

$$f(\boldsymbol{x}) = \sum_{t=1}^{T} \beta_t b(\boldsymbol{x};\boldsymbol{\gamma}_t)$$

其中用 $b(\boldsymbol{x};\boldsymbol{\gamma}_t)$ 表示第 t 个基学习器, 也可简记为 $b_t(\boldsymbol{x})$, $\boldsymbol{\gamma}_t$ 为基函数的模型参数, β_t 为基函数的权重系数.

在给定训练样本数据和损失函数 $L(y, f(\boldsymbol{x}))$ 的条件下, 学习加法模型 $f(\boldsymbol{x})$ 即要估计参数 β_t 和 $\boldsymbol{\gamma}_t$, 也即最小化如下损失函数

$$\min_{\beta_t,\boldsymbol{\gamma}_t} \sum_{i=1}^{m} L\left(y_i, \sum_{t=1}^{T} \beta_t b(\boldsymbol{x};\boldsymbol{\gamma}_t)\right)$$

这是一个非常复杂的优化问题, 可以采用启发式算法来解决, 具体的说, 可以采用贪心算法. 前向分布算法 (forward stagewise algorithm) 就是做这个的, 它的思想是: 每一步只学习一个基函数及其系数, 在当前状态下达到最优, 逐步逼近最终问题.

给定训练数据集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)\}$, 损失函数 $L(y, f(\boldsymbol{x}))$ 和基函数集合 $\{b(\boldsymbol{x};\boldsymbol{\gamma})\}$, 前向分布算法求解加法模型的过程如下:

(1) 初始化 $f_0(\boldsymbol{x}) = 0$

(2) 对 $t = 1, 2, \cdots, T$

(a) 极小化损失函数

$$(\beta_t, \boldsymbol{\gamma}_t) = \arg\min_{\beta,\boldsymbol{\gamma}} \sum_{i=1}^{m} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \beta b(\boldsymbol{x}_i;\boldsymbol{\gamma}))$$

得到参数 $\beta_t, \boldsymbol{\gamma}_t$

(b) 更新

$$f_t(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t b(\boldsymbol{x}; \boldsymbol{\gamma}_t)$$

(3) 最后得到加法模型

$$f(\boldsymbol{x}) = f_T(\boldsymbol{x}) = \sum_{t=1}^{T} \beta_t b(\boldsymbol{x}; \boldsymbol{\gamma}_t)$$

是不是很像优化中迭代求解得步骤? 我感觉真是太像了, 得到下降方向和最优步长, 继而可以更新值. 当然了, 本来这就是优化问题, 所以肯定很像啦.

前向分布算法的巧妙之处在于利用贪心思想, 将同时求解从 $t = 1$ 到 T 的所有参数 $\beta_t, \boldsymbol{\gamma}_t$ 的优化问题简化为了逐次求解各个 $\beta_t, \boldsymbol{\gamma}_t$ 的优化问题.

利用前向分布算法可以推导出 AdaBoost 算法, 这里就不多说了, 可以参见李航的《统计学习方法》.

4.2 梯度提升

前向分布算法只是给出了一个大体的框架

$$f_t(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t b(\boldsymbol{x}; \boldsymbol{\gamma}_t) = f_{t-1}(\boldsymbol{x}) + \beta_t b_t(\boldsymbol{x})$$

但没有给出如何具体求出参数 β_t 和基函数 $b_t(\boldsymbol{x})$ 的方法. 在一些具体的损失函数 (比如平方损失函数) 中, 这个优化还是比较容易的, 对于一般的损失函数, 我们可以采用类似于优化中最速下降法的算法, 即梯度提升 (Gradient Boosting) 方法.

关于梯度提升的一些详细资料可参考其他文献, 比如维基上https://en.wikipedia.org/wiki/Gradient_boosting就写的不错. 其关键思想是沿着损失函数的负梯度方向选取基函数 $b_t(\boldsymbol{x})$, 然后通过一维线性搜索选取系数 β_t. 这对于学过最优化的我们再熟悉不过了. 这里直接给出梯度提升的步骤.

给定训练数据集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)\}$, 损失函数 $L(y, f(\boldsymbol{x}))$, 迭代次数 T, 梯度提升的算法步骤是

(1) 初始化 $f_0(\boldsymbol{x})$, 比如可取

$$f_0(\boldsymbol{x}) = \arg\min_{\beta} \sum_{i=1}^{m} L(y_i, \beta)$$

(2) 对 $t = 1, 2, \cdots, T$

(a) 计算损失函数在当前模型的负梯度值作为伪残差值

$$r_{it} = -\left[\frac{\partial L(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}\right]_{f(\boldsymbol{x})=f_{t-1}(\boldsymbol{x})}, i = 1, 2, \cdots, m$$

(b) 找到最优的基函数 $b_t(\boldsymbol{x})$ 去拟合伪残差值, 也就是说拟合数据集 $\{(\boldsymbol{x}_i, r_{it})\}_{i=1}^m$

(c) 通过线性搜索寻找最优系数 β_t, 也就是解如下的一维最优化问题

$$\beta_t = \arg\min_{\beta} \sum_{i=1}^{m} L(y_i, f_{t-1}(\boldsymbol{x}_i) + \beta b_t(\boldsymbol{x}_i))$$

(d) 更新模型

$$f_t(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t b_t(\boldsymbol{x})$$

(3) 输出最终模型 $f(\boldsymbol{x}) = f_T(\boldsymbol{x})$

4.3 提升树

提升树是以分类树或者回归树为基分类器的提升方法. 目前已经很多次提到过“提升”这个词, 它到底是什么含义呢?

其实, 提升方法就是一种集成学习的方式而已, 它采用加法模型 (即基函数的线性组合) 与前向分布算法, 将几个弱分类器“提升”为一个强分类器. 以决策树为基函数的提升方法称之为提升树 (Boosting Tree).

梯度提升和决策树结合起来, 便称之为梯度提升决策树 (Gradient Boosting Decision Tree, 简记为 GBDT), 网上还有很多其他叫法, 比如 MART(Multiple Additive Regression Tree), GBRT(Gradient Boosting Regression Tree), Tree Net , Boosted Tree 等, 其实它们都是一个东西, 就是指把 (梯度) 提升方法和决策树这两个模型组合起来, 模型组合其实就是集成学习.

5 Bagging 方法与随机森林

上面我们介绍了集成学习的一个代表, 即 Boosting 方法, 并详述了 AdaBoost 算法的过程. 下面我们稍微提一下集成学习的另一个代表, 即 Bagging 方法, 并介绍一下随机森林.

Boosting 方法的特点是个体学习器之间存在强依赖关系, 是必须串行生成的序列化方法. Bagging 方法与之不同, 它是个体学习器之前不存在强依赖关系、可同时生成的并行化方法. Bagging 这个名字由 Bootstrap AGGregatING 缩写而来. 其中 Bootstrap 便是自助采样法.

给定包含 m 个样本的数据集, 我们先随机取出一个样本放入采样集中, 再把该样本放回初始数据集, 使得下次采样时该样本仍有可能被选中, 这样经过 m 次随机采样操作, 我们便得到了含有 m 个样本的采样集, 初始训练集中有的样本在采样集中多次出现, 有的则可能从未出现. 我们在这个采样集上训练出一个模型. 如此不断重复 T 次, 我们可采样出 T 个包含 m 个样本的采样集, 然后基于每个采样集训练出一个基学习器, 再将这些基学习器进行组合 (组合的方式有很多, 这里暂且不提), 这就是 Bagging 方法的基本流程.

随机森林 (Random Forest, 简称 RF) 是 Bagging 方法的一个扩展变体, 它在以决策树为基学习器构建 Bagging 集成的基础上, 进一步在决策树的训练过程中引入了随机属性选择.

具体来说, 传统决策树在选择划分属性时是在当前节点的属性集合 (假设有 d 个属性特征) 中选择一个最优属性, 而在随机森林中, 对基决策树的每个节点, 先从该节点的属性集合中随机选择一个包含 $k(k << n)$ 个属性的子集, 然后再从这个子集中选择一个最优属性用于划分, 这里的参数 k 控制了随机性的引入程度: 若令 $k = d$, 则基决策树的构建与传统决策树相同; 若令 $k = 1$, 则是随机选择一个属性用于划分. 一般情况下, 推荐值为 $k = \log_2 d$. 也有的资料上介绍, 假设样本有 n 个特征, 那么我们直接进行列采样, 也就是随机选取 k 个特征构建决策树, k 的可取值有 $\sqrt{n}, 1/2\sqrt{n}$ 和 $2\sqrt{n}$.

也就是说, 随机森林通过随机的方式建立了 T 棵树, 因此称为森林. 随机森林简单, 容易实现, 计算开销小, 令人惊奇的是, 它在很多现实任务中表现出强大的性能. 可以看出, 随机森林对 Bagging 方法只做了小改动, 但是与 Bagging 中基学习器的“多样性”仅通过样本扰动 (通过对初始训练集采样) 而来不同, 随机森林中基学习器的多样性不仅来自样本扰动, 还来自属性扰动, 这就使得最终集成的分类器的泛化性能可通过个体学习器之间的差异度的增加而进一步提升.

6 总结

6.1 参考资料

(1) 李航的《统计学习方法》, 算法步骤和例子均来自于此.

(2) 周志华的《机器学习》, 这个不多说了, 比较周志华是集成学习方面的大牛.

(3) 博客: http://blog.csdn.net/v_july_v/article/details/40718799,

其实内容就是李航的《统计学习方法》, 但是对例子解释的比较清楚. 参考李航的很多, 比如还有这篇: http://blog.csdn.net/dark_scope/article/details/14103983, 里面加入了自己的 Python 编程.

(4) 博客: http://blog.csdn.net/carson2005/article/details/41444289, 讲述了一些关系, 同时有形象的图形展示.

(5) 博客: http://www.36dsj.com/archives/21036, 对随机森林和梯度提升树作了介绍, 不过原文应该是这个: http://www.cnblogs.com/LeftNotEasy/archive/2011/03/07/1976562.html.

(6) 维基: https://en.wikipedia.org/wiki/Gradient_boosting, 对梯度提升做了详细介绍.

参考文献

[1] 李荣华. 偏微分方程数值解法. 高等教育出版社 (2010)

[2] Zhilin Li,Zhonghua Qiao,Tao Tang.*Numerical Solutions of Partial Differential Equations-An Introduction to Finite Difference and Finite Element Methods.*(2011)

[3] 孙志忠. 偏微分方程数值解法. 科学出版社 (2011)

[4] 陆金甫关治. 偏微分方程数值解法. 清华大学出版社 (2004)

附录