集成学习

1 概述

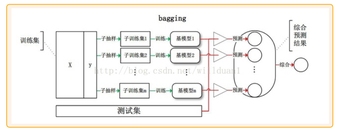
在机器学习的有监督学习算法中，我们的目标是学习出一个稳定的且在各个方面表现都较好的模型，但实际情况往往不这么理想，有时我们只能得到多个有偏好的模型（弱监督模型，在某些方面表现的比较好）。集成学习就是组合这里的多个弱监督模型以期得到一个更好更全面的强监督模型，集成学习潜在的思想是即便某一个弱分类器得到了错误的预测，其他的弱分类器也可以将错误纠正回来。

集成方法是将几种机器学习技术组合成一个预测模型的元算法，以达到减小方差（bagging）、偏差（boosting）或改进预测（stacking）的效果。

- 集成学习在各个规模的数据集上都有很好的策略
  - 数据集大：划分成多个小数据集，学习多个模型进行组合
  - 数据集小：利用Bootstrap方法进行抽样，得到多个数据集，分别训练多个模型再进行组合
- 特点
  - 1 将多个分类方法聚集在一起，以提高分类的准确率（这些算法可以是不同的算法，也可以是相同的算法。）
  - 2 集成学习法由训练数据构建一组基分类器，然后通过对每个基分类器的预测进行投票来进行分类
  - 3 严格来说，集成学习并不算是一种分类器，而是一种分类器结合的方法
  - 4 通常一个集成分类器的分类性能会好于单个分类器
  - 5 如果把单个分类器比作一个决策者的话，集成学习的方法就相当于多个决策者共同进行一项决策。
- 决策树与这些算法框架进行结合所得到的新的算法
  - Bagging + 决策树 = 随机森林
  - AdaBoost + 决策树 = 提升树
  - Gradient Boosting + 决策树 = GBDT

2 Bagging

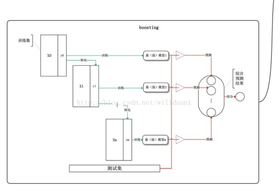
- Bagging即套袋法，先说一下bootstrap，bootstrap也称为自助法，它是一种有放回的抽样方法，目的为了得到统计量的分布以及置信区间，其算法过程如下：
  - 从原始样本集中抽取训练集。每轮从原始样本集中使用Bootstraping的方法抽取n个训练样本（在训练集中，有些样本可能被多次抽取到，而有些样本可能一次都没有被抽中）。共进行k轮抽取，得到k个训练集。（k个训练集之间是相互独立的）
  - 每次使用一个训练集得到一个模型，k个训练集共得到k个模型。（注：这里并没有具体的分类算法或回归方法，我们可以根据具体问题采用不同的分类或回归方法，如决策树、感知器等）
  - 对分类问题：将上步得到的k个模型采用投票的方式得到分类结果；对回归问题，计算上述模型的均值作为最后的结果。（所有模型的重要性相同）



- 特点
  - Bagging通过降低基分类器的方差，改善了泛化误差
  - 其性能依赖于基分类器的稳定性；如果基分类器不稳定，bagging有助于降低训练数据的随机波动导致的误差；如果稳定，则集成分类器的误差主要由基分类器的偏倚引起
  - 由于每个样本被选中的概率相同，因此bagging并不侧重于训练数据集中的任何特定实例
- 常用的集成算法类是随机森林。
  - 在随机森林中，集成中的每棵树都是由从训练集中抽取的样本（即bootstrap 样本）构建的。另外，与使用所有特征不同，这里随机选择特征子集，从而进一步达到对树的随机化目的。因此，随机森林产生的偏差略有增加，但是由于对相关性较小的树计算平均值，估计方差减小了，导致模型的整体效果更好。

3 Boosting

- 其主要思想是将弱分类器组装成一个强分类器。在PAC（probably approximately correct，概率近似正确）学习框架下，则一定可以将弱分类器组装成一个强分类器。
- 关于Boosting的两个核心问题
  - 在每一轮如何改变训练数据的权值或概率分布？
    - 通过提高那些在前一轮被弱分类器分错样例的权值，减小前一轮分对样例的权值，来使得分类器对误分的数据有较好的效果。
  - 通过什么方式来组合弱分类器？
    - 通过加法模型将弱分类器进行线性组合，比如：
      - AdaBoost（Adaptive boosting）算法：刚开始训练时对每一个训练例赋相等的权重，然后用该算法对训练集训练t轮，每次训练后，对训练失败的训练例赋以较大的权重，也就是让学习算法在每次学习以后更注意学错的样本，从而得到多个预测函数。通过拟合残差的方式逐步减小残差，将每一步生成的模型叠加得到最终模型
      - GBDT（Gradient Boost Decision Tree），每一次的计算是为了减少上一次的残差，GBDT在残差减少（负梯度）的方向上建立一个新的模型。



4 详解AdaBoost

提升方法 AdaBoost 算法

- 提升方法基于这样一种思想：对于一个复杂任务来说，将多个专家的判断进行适当的综合所得出的判断，要比其中任何一个专家单独的判断好。实际上，就是“三个臭皮匠顶个诸葛亮”的道理。
- 提升方法就是从弱学习算法出发，反复学习，得到一系列弱分类器（又称为基本分类器），然后组合这些弱分类器，构成一个强分类器。大多数的提升方法都是改变训练数据的概率分布（训练数据的权值分布），针对不同的训练数据分布调用弱学习算法学习一系列弱分类器
- 有两个问题需要回答：
  - 在每一轮如何改变训练数据的权值或概率分布
    - AdaBoost的做法是，提高那些被前一轮弱分类器错误分类样本的权值，而降低那些被正确分类样本的权值。这样一来，那些没有得到正确分类的数据，由于其权值的加大而受到后一轮的弱分类器的更大关注。于是，分类问题被一系列的弱分类器“分而治之”。
  - 如何将弱分类器组合成一个强分类器
    - 即弱分类器的组合，AdaBoost采取加权多数表决的方法。具体地，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的权值，使其在表决中起较小的作用。
- 叙述 AdaBoost 算法
  - 现在叙述 AdaBoost 算法。假设给定一个二类分类的训练数据集

    $$T=\{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$$

    其中，每个样本点由实例与标记组成。实例 $x_i\in X\subset R^n$，标记 $y_i\in Y=\{-1,+1\}$，X 是实例空间，Y 是标记集合。
    输出：最终分类器 $G(x)$
  - (1) 初始化训练数据的权值分布

    $$D_1=(\omega_{11},\ldots,\omega_{1i},\ldots,\omega_{1N}),\quad \omega_{1i}=\frac{1}{N},\quad i=1,2,\ldots,N$$
  - (2) 对 $M=1,2,\ldots,m$
    (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

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

    假设训练数据集具有均匀的权值分布，即每个训练样本在基本分类器的学习中作用相同。
    (b) 计算 $G_m(x)$ 在训练数据集上的分类错误率

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

    (c) 更新训练数据集的权值分布

    $$D_{m+1}=(\omega_{m+1,1},\ldots,\omega_{m+1,i},\ldots,\omega_{m+1,N})$$

    $$\omega_{m+1,i}=\frac{\omega_{mi}}{Z_m}exp(-\alpha_m y_i G_m(x_i)),\quad i=1,2,\ldots,N$$

    这里 Z_m 是规范化因子

    $$Z_m=\sum_{i=1}^{N}\omega_{mi}exp(-\alpha_m y_i G_m(x_i))$$

    $G_m(x)$ 在加权的训练数据集上的分类误差率是 $G_m(x)$ 提示错误样本的权值之和，由此可以看出数据权值分布 D_m 与基本分类器 $G_m(x)$ 的分类误差率有关。
    (d) 计算 $G_m(x)$ 的系数

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

    当 $e_m\le\frac{1}{2}$ 时，$\alpha_m\ge 0$，并且 α_m 随着 e_m 的减小而增大，
    所以分类误差率越小的基本分类器在最终分类器中的作用越大。

    它使 D_{m+1} 成为一个概率分布，
    $\omega_{m+1,i}$ 也可以写成

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

    因此，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，使得训练数据在基本分类器的学习中
    起不同的作用，这是AdaBoost的一个特点。
  - (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

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

    线性组合 $f(x)$ 实现 M 个基本分类器的加权表决。系数 α_m 表示了基本分类器 $G_m(x)$ 的重要性，这里，所有 α_m 之和并不为1。$f(x)$ 的符号决定实例 x 的类，$f(x)$ 的绝对值表示分类的确信度。利用基本分类器的线性组合构建最终分类器是 AdaBoost 的另一特点。
- 向前分布算法
  - 输入：训练数据集 $T=\{(x_1,\ y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$；损失函数 $L(Y,f(x))$；基函数集 $\{b(x;\gamma)\}$；
    输出：加法模型 $f(x)$。
    (1) 初始化 $f_0(x)=0$
    (2) 对 $m=1,2,\ldots,M$
    (a) 极小化损失函数

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

    得到参数 β_m,γ_m
    (b) 更新

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

    (3) 得到加法模型

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

    这样，前向分步算法将同时求解从 $m=1$ 到 M 所有参数 $\beta_m,\ \gamma_m$ 的优化问题简化为逐次求解各个 $\beta_m,\ \gamma_m$ 的优化问题。

AdaBoost算法的训练误差分析

- AdaBoost 最基本的性质是它能在学习过程中不断减少训练误差，即在训练数据集上的分类误差率。AdaBoost 算法最终分类器的训练误差界为

  $$\frac{1}{N}\sum_{i=1}^{N}I(G(x_i)\neq y_i)\le\frac{1}{N}\sum_i exp(-y_i f(x_i))=\prod_m Z_m$$

  - 当 $G(x_i)\neq y_i$ 时，$y_i f(x_i)<0$，因而 $exp(-y_i f(x_i))\ge 1$。因此可直接推导出前半部分。
    现推导后半部分如下

    $$\begin{aligned}\frac{1}{N}\sum_i exp(-y_i f(x_i)) &= \frac{1}{N}\sum_i exp(-\sum_{m=1}^{M}\alpha_m y_i G_m(x_i))\\ &= \frac{1}{N}\sum_i\prod_{m=1}^{M}exp(-\alpha_m y_i G_m(x_i))\\ &= \sum_i\omega_{1i}\prod_{m=1}^{M}exp(-\alpha_m y_i G_m(x_i))\\ &= \sum_i\frac{\omega_{2i}Z_m}{exp(-\alpha_1 y_i G_1(x_i))}\prod_{m=1}^{M}exp(-\alpha_m y_i G_m(x_i))\\ &= Z_1\sum_i\omega_{2i}\prod_{m=2}^{M}exp(-\alpha_m y_i G_m(x_i))\\ &= Z_1 Z_2\sum_i\omega_{3i}\prod_{m=3}^{M}exp(-\alpha_m y_i G_m(x_i))\\ &\cdots\\ &= Z_1 Z_2\cdots Z_{m-1}\sum_i\omega_{Mi}exp(\alpha_M y_i G_M(x_i))\\ &= \prod_{m=1}^{M}Z_m\end{aligned}$$

    这一定理说明，可以在每一轮选取适当的 G_m 使得 Z_m 最小，从而使训练误差下降最快。
- 二类分类问题 AdaBoost 的训练误差界

  $$\prod_{m=1}^{M}Z_m=\prod_{m=1}^{M}[2\sqrt{e_m(1-e_m)}]=\prod_{m=1}^{M}\sqrt{1-4\gamma_m^2}\le exp(-2\sum_{m=1}^{M}\gamma_m^2)$$

  这里 $\gamma_m=\frac{1}{2}-e_m$。

  证明：

  $$\begin{aligned}Z_m &= \sum_{i=1}^{N}\omega_{mi}exp(-\alpha_m y_i G_m(x_i))\\ &= \sum_{y_i=G_m(x_i)}\omega_{mi}e^{-\alpha_m}+\sum_{y_i\neq G_m(x_i)}\omega_{mi}e^{\alpha_m}\\ &= (1-e_m)e^{-\alpha_m}+e_m e^{\alpha_m}\\ &= \sqrt{(1-e_m)^2e^{-\frac{1-2e_m}{1-e_m}}+e_m^2e^{\frac{1-2e_m}{e_m}}+2e_m(1-e_m)}\\ &= 2\sqrt{e_m(1-e_m)}\\ &= \sqrt{1-4\gamma_m^2}\end{aligned}$$

  至于不等式

  $$\prod_{m=1}^{M}\sqrt{1-4\gamma_m^2}\le exp(-2\sum_{m=1}^{M}\gamma_m^2)$$

  可先由 e^x 和 $\sqrt{1-x}$ 在点 $x=0$ 的泰勒展开式推出 $\sqrt{1-4\gamma_m^2}\le exp(-2\gamma_m^2)$ 进而得到。
- 二类分类问题如果存在 $\gamma>0$，对所有 m 有 $\gamma_m>\gamma$，则

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

  这表明在此条件下 Adaboost 的训练误差是以指数速率下降的。

提升树

- 提升树是以分类树或回归树为基本分类器的提升方法。提升树被认为是统计学习中性能最好的方法之一。
- 以决策树为基函数的提升方法称为提升树（boosting tree）。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。
- 回归问题的提升树算法
  - 提升树模型可以表示为决策树的加法模型

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

    其中，$T(x;\Theta_m)$ 表示决策树；Θ_m 为决策树的参数；M 为树的个数。
  - 4 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

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

    其中，$f_{m-1}(x)$ 为当前模型，通过经验风险极小化确定下一棵决策树的参数 $\hat{\Theta}_m$，

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

    由于树的线性组合可以很好地拟合训练数据，即便数据中的输入与输出之间的关系很复杂也是
    如此，所以提升树是一个高性能的学习算法。
    - 5. 已知一个训练数据集 $T=\{(x_1,\ y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$，$x_i\in X\subset R^n$，
      $y_i\in Y\subset R$，在决策树部分我们已了解到树的结构，如果将输入空间 X 划分为 J 个互
      不相交的区域 $R_1,R_2,\ldots,R_J$，并且在每个区域上确定输出的常量 c_j，那么树可以表示
      为

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

      其中，参数 $\Theta=\{(R_1,c_1),(R_2,c_2),\ldots,(R_J,c_J)\}$ 表示树的区域划分和各区域上的
      常数。J 是回归树的复杂度即叶结点个数。
      回归问题提升树使用以下前向分步算法：

      $$\begin{aligned}&f_0(x)=0\\ &f_m(x)=f_{m-1}(x)+T(x;\Theta_m),\quad m=1,2,\ldots,M\\ &f_M(x)=\sum_{m=1}^{M}T(x;\Theta_m)\end{aligned}$$

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

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

      得到 $\hat{\Theta}_m$，即第 m 棵树的参数。
      当采用平方误差损失函数时，

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

      其损失变为

      $$L(y_i,f_{m-1}(x_i)+T(x_i;\Theta_m))=[y-f_{m-1}(x)-T(x;\Theta_m)]^2=[r-T(x;\Theta_m)]^2$$

      这里，$r=y-f_{m-1}(x)$ 是当前模型拟合数据的 **残差**（residual）。所以，对回归问题的
      提升树算法来说，只需简单地拟合当前模型的残差。这样，算法是相当简单的。
- 梯度提升算法
  - 输入：训练数据集 $T=\{(x_1,\ y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$，$x_i\in X\subset R^n$，
    $y_i\in Y\subset R$；损失函数 $L(Y,f(x))$；
    输出：回归树 $\hat{f}(x)$。
    (1) 初始化

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

    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 $m=1,2,\ldots,M$，计算
    (a) 对 $i=1,2,\ldots,N$，计算

    $$r_{mi}=-[\frac{\partial L(y_i,f(x_i))}{\partial f(x_i)}]_{f(x)=f_{m-1}(x)}$$

    计算损失函数的负梯度在当前模型的值，将它作为残差的估计。对于平方损失函数，它就是通
    常所说的残差；对于一般损失函数，它是残差的近似值。
    (b) 对 r_{mi} 拟合一个回归树，得到第 m 棵树的叶结点区域 R_{mj}，$j=1,2,\ldots,J$。
    估计回归树叶结点区域，以拟合残差的近似值。
    (c) 对 $j=1,2,\ldots,J$，计算

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

    利用线性搜索估计叶结点区域的值，使损失函数极小化。
    (d) 更新 $f_m(x)=f_{m-1}(x)+\sum_{j=1}^{J}c_{mj}I(x\in R_{mj})$
    (3) 得到回归树

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