集成学习

张伟平

zwp@ustc.edu.cn

Office: 东区管理科研楼 1006

Phone: 63600565

课件 http://staff.ustc.edu.cn/~zwp/

论坛 http://fisher.stat.ustc.edu.cn

简介

1.1 简介

- 集成学习方法
  - 假设你有许多基础分类器—“**简单的**分类规则”，则组合这些分类规则可能是一个好主意, 可能会比单个规则获得更高的精度
  - 选择基础分类器时候往往主要考虑的是其简单性，而非精度
  - 基础分类器应该对总体中的一部分不同个体是精确的，他们组合起来可以有效处理所有个体 (即互为补充)
  - 基础分类器之间的差异
    * 不同基础分类器有着不同的假设
    * 同一分类器有着不同的超参数

    * 同一输入对象或事件的不同表达：例如语音识别里面嘴唇运动的声音特征和形状变化过程都是输入
    * 不同的训练集：使用不同训练集并行或者依次训练基础分类器
    * 不同的子任务：主要任务通过使用一些基础分类器解决一系列子任务而完成

– 组合基础分类器

    * 多专家综合方法 (并行方式)
        · 基础分类器并行方式运行
        · 综合每个分类器的结果给出最终结果
        · 例如：投票制
    * 多阶段综合方法 (依序方式)
        · 基础分类器依序进行

· 根据复杂性将基础分类器从小到大排序：不使用复杂的分类器，除非都不满意前面的简单分类器

- 一个集成学习方法比单个学习方法的推广能力更强，原因在于
  - 训练集没有包含可以选择一个最佳分类器的充分信息
  - 学习算法的学习过程可能不完美
  - 搜索的空间可能没有包含真正的目标函数，而集成学习可以给出更好的近似
- **模型选择**和**模型平均**(集成学习)
  - 当一个模型比其他模型显著的更精确，则模型选择方法更好
  - 如果所有模型预测精度类似，则模型平均方法更好

1.2 投票

- 投票 (voting) 方法使用所有基础学习器结果的凸组合作为最终决策:

$$y = f(d_1, \ldots, d_L|\Theta) = \sum_{i=1}^{L} w_i d_i$$

其中 w_i 为权重, d_i 为第 i 个学习器的预测结果; $\Theta = (\theta_1, \ldots, \theta_L)$ 为所有学习器的参数.

- 对类别 C_j :

$$y_j = \sum_{i=1}^{L} w_i d_{ij}$$

其中 d_{ij} 表示第 i 个分类器对类别 C_j 的结果, w_i 为权重

- **简单投票**:

$$w_i = \frac{1}{L}$$

- **Bayes**模型选择:

$$P(C_j|\mathbf{x}) = \sum_{all\ \ models\ \ M_i} P(C_j|\mathbf{x}, M_i)P(M_i)$$

因此权重 w_i 可以视为是对模型先验概率的一个近似.

- 记 $y = \frac{1}{L}\sum d_i$， $d_1, \ldots, d_L$ 为预测精度类似的分类器. 则

$$Ey = \frac{1}{L}\sum_i E[d_i] = E[d_i]$$

$$Var(y) = \frac{1}{L^2}[\sum_i Var(d_i) + 2\sum_{i<j} cov(d_i, d_j)]$$

因此当 L 增加时候，期望值没有变化，而方差降低，从而导致精确度提高.

1.3 Bagging

Bagging=Bootstrap AGGregatING

- 给定一组训练集 $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  - 使用 Bootstrap 方法从 D 中有放回的取出 N 个作为训练集 D_1, 重复此过程 T 次, 得到 T 个训练集 $D_1, \ldots, D_T$
  - 使用每个训练集 D_i $(i = 1, \ldots, T)$ 训练一个分类器或者回归函数, 得到一列输出结果 $f_1(\mathbf{x}), \ldots, f_T(\mathbf{x})$.
- 最终聚合所有的分类器
  - 回归问题:

$$\bar{f}(\mathbf{x}) = \frac{1}{T} \sum f_i(\mathbf{x})$$

  - 分类问题:

$$\bar{f}(\mathbf{x}) = \theta(f_1(\mathbf{x}), \ldots, f_T(\mathbf{x}))$$

$\theta()$ 为基于 $f_1, \ldots, f_T$ 的投票制。

- Bagging 可以视为是一种特殊的模型平均方法，可以降低方差和提高精度
- Bagging 可以显著提高不稳定学习算法 (训练集的小幅度变化导致学习器的结果发生较大变化) 的精度
- 分类器下的 Bagging 算法:

- **Input**: Data set $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$
  Base learning algorithm $\mathcal{L}$
  Number of learning rounds T
- **Process**:
  For $t = 1, \ldots, T$
  $\mathcal{D}_t = Bootstrap(\mathcal{D})$ %Generate a bootstrap sample from $\mathcal{D}$
  $h_t = \mathcal{L}(\mathcal{D}_t)$ %Train a base learner h_t from the bootstrap sample
  End
- **Output**: $H(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} 1(y = h_t(\mathbf{x}))$

- Bootstrap 样本一般比交叉验证样本重合较多，因此他们的估计是相依的
- N 次随机有放回抽取中一个个体不被取出的概率为

$$\left(1-\frac{1}{N}\right)^{N} \approx e^{-1} = 0.368$$

  因此每个 Bootstrap 样本大约包含 63.2% 的个体
- 使用多组 Bootstrap 样本以最大可能使得系统在所有个体上进行训练
- 随机森林 (Random Forest) 是一种 bagging 的变种方法，其是一种高效的集成学习方法

1.4 Boosting

- Bagging 方法中训练学习器时候不考虑不同基础学习器是否互补，以及它们各自的稳定性
- Boosting 方法中通过在前一个学习器出错的个体上训练下一个学习器
- Boosting 方法通过综合较差的学习器 (比随机猜效果能好一点的学习器)，来得到一个较强的学习器（较弱学习器的加权和)
- 有许多不同的 boosting 算法: AdaBoost(Adaptive boosting, Freund & Schapire, 1996) 是第一个
  - LPBoost: 线性规划 boosting
  - BrownBoost: 增加稳健性
  - LogitBoost 等

AdaBoost

- 在 Bootstrap 抽样时候，先给所有个体同样的权重，得到训练集 D_1 后训练第一个分类器
- 使用训练好的分类器对原始数据集 D 进行预测
- 对第一个分类器预测错误的个体，增加其权重。使用此权重进行 Bootstrap 抽样，得到第二个训练集 D_2，再训练第二个分类器
- 再使用第二个分类器对原始数据集 D 进行预测，增加分类错误的个体权重。
- 再在新的权重下进行 Bootstrap 抽样，训练分类器。依次下去，直至所有分类器都被训练好。
- 最后对所有的分类器加权求和，得到最终的集成分类器。

AdaBoost.M1(两分类) 算法:

1. Training data: (x_i, y_i), $i = 1, \ldots, n$, $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y} = \{-1, 1\}$.
2. Let $w_{1,i} = \frac{1}{n}$, $i = 1, \ldots, n$.
3. For $t = 1, \ldots, T$:
   - 3.1 Learn classifier $h_t : \mathcal{X} \to \mathcal{Y}$ from a set of weak classifiers called *hypotheses*, $\mathcal{H} = \{h(\cdot, \omega) | \omega \in \Omega\}$, that minimizes the error rate with respect to distribution $w_{t,i}$ over x_i's.
   - 3.2 Let $r_t = \sum_{i=1}^{n} w_{t,i} I(y_i \neq h_t(x_i))$. If $r_t > 0.5$, stop.
   - 3.3 Choose $\alpha_t \in \mathcal{R}$. Usually set $\alpha_t = \frac{1}{2} \log \frac{1-r_t}{r_t}$.
   - 3.4 Update $w_{t+1,i} = \frac{w_{t,i} e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$, where Z_t is a normalization factor to ensure $\sum_i w_{t+1,i} = 1$.
4. Output the final classifier: $f(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.

1.5 随机森林

随机森林树 (Random Forests) 算法 (Breiman,2001) 是一种组合多个树分类器进行分类的方法, 其基本思想是每次随机选取一些特征, 独立建立树, 重复这个过程, 保证每次建立树时变量选取的可能性一致, 从而得到许多彼此不相关的树, 最终的分类结果通过类似 Bagging 的方法来得到. 在许多问题中, 随机森林的性能非常类似于提升算法的性能, 可以非常简单的进行训练和调整. 因此, 随机森林是一种非常流行的算法.

在概率论中我们已经知道 B 个 i.i.d 随机变量 (每个方差为 σ^2) 的平均值具有方差 σ^2/B, 这正是 Bagging 方法希望降低方差的来由. 但是当 B 个随机变量不相互独立时候 (仅是同分布), 比如具有正的两两相关系数 ρ, 则平均值的方差为

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

当 B 增加时候, 第二项会趋于零, 但是第一项会一直保留, 因此此时使用 Bagging 方法的话, 方差就不会得到显著降低. 随机森林的想法就是希望通过减少树之间的相关性, 然后通过 Bagging 方法来降低方差. 随机森林通过在每次 Bootstrap 抽样后随机选取少量变量来作为生成树的变量. 具体算法可以表述为

Input: 训练集$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$

for $b = 1, \ldots, B$ **do**

1. 从训练集中得到Bootstrap抽样Z^*
2. 对Bootstrap样本Z^*, 生成树T_b: 对树的每个终端节点重复下面步骤, 直至达到最小节点大小(节点内包含的点数目):
   (a) 从p个变量中随机选择m个变量(常取$m \approx \sqrt{p}$)
   (b) 在m个变量中选出最佳的拆分变量集和划分点
   (c) 将节点分割为两个子节点

end

Output: 输出树$\{T_b\}_{b=1}^B$,对新的点$\mathbf{x}$的预测通过

- 回归树: $\widehat{f}^B(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^B T_b(\mathbf{x})$
- 分类树: $\widehat{f}^B(\mathbf{x}) = majorityvote\{T_b(\mathbf{x}), b = 1, \ldots, B\}$

可以看出, 随机森林和 Bagging 方法的最大区别在于每次拆分时

候随机从 p 个自变量中选择 m 个变量, 因此即便自变量中某些变量是最主要的, 随机森林算法也是不考虑这一点的. 这种想法很聪明. 事实上, 如果自变量中存在一个非常强的自变量, 则在 Bagging 方法中几乎所有的树都会选择这个变量作为拆分变量, 从而导致生成的树非常相似, 因此高度正相关. 此时, 平均所有生成的树就不会达到降低方差的目的. 而随机森林的做法强制每次拆分时候选择变量的子集作为拆分变量, 恰好避免这一情况的发生. 此外, 当 $m = p$ 时候, 则随机森林算法就是 Bagging 方法.

R包 **randomForest** 可以对回归和分类问题实施 Breiman 的随机森林算法.