

规则学习 Rule Learning

2020 年 4 月 19 日

1 15.1 基本概念 Basic Concepts

1.1 规则 If-then

$$\oplus \leftarrow f_1 \wedge f_2 \wedge \cdots \wedge f_L \quad (1)$$

其中, \oplus 称为规则头 (head), f_i 称为文字 (literal) 或选择子 (selector), L 是规则的长度。在一些论文当中右边的这个部分也称为 complex.

1.2 覆盖 Cover

规则 1 覆盖西瓜数据集 2.0 中的样本 1, 2, 3, 4, 5. 可以发现, 实际上覆盖是对整个规则都要成立, 也就是说属性值要符合, 结果也要符合。

例如规则: 好瓜 \leftarrow 色泽 = 青绿, 如果是仅考虑属性值的话, 那么覆盖了 1,4,6,10,13,17, 再把结果也考虑进去, 那么覆盖的是 1,4,6。

前面这种只考虑属性值的覆盖, 我们称之为 complex 的覆盖, 之后在算法里会用到, 而且大多数情况下用的是这种覆盖。

规则 1: 好瓜 \leftarrow 根蒂 = 蜷缩 \wedge 脐部 = 凹陷

规则 2: 坏瓜 \leftarrow 纹理 = 模糊

我们将上面两条规则组成的规则集记为 R

1.2.1 覆盖可能造成的两个问题

1. 冲突: 同一示例被不同规则覆盖, 且判别结果不同。冲突消解: 投票法、排序法、元规则法等

- 投票法：判别结果相同的规则数最多的为最终结果
- 排序法：定义一种排序，最靠前的为最终结果
- 元规则 (meta-rule) 法：根据领域知识设定一些“规则的规则”，即指导规则的使用
- 2. 不完全覆盖：规则集不能完全覆盖整个数据集
- 增加默认规则：“未被规则 1, 2 覆盖的都不是好瓜”

1.3 规则的表达能力

- 1. 命题规则：原子命题 + 逻辑连接词，例：R
- 2. 一阶规则：原子公式 + 量词 + 逻辑连接词，例：

$$\forall X(N(\sigma(X))) \leftarrow N(X), \text{ where } \sigma(X) = X + 1 \quad (2)$$

- 3. 命题规则是一阶规则的特殊形式，一阶规则表达能力比命题规则要强

2 15.2 序贯覆盖（以命题规则学习为例）

2.1 基本思想

逐条归纳，通过每次训练生成一条仅覆盖正例的规则，就去除掉那些覆盖的样例，用剩下的继续训练

2.2 以西瓜数据集 2.0 的训练集为例的序贯覆盖

- 1. 好瓜 \leftarrow 色泽 = 青绿 覆盖了 1、4、6、10、13、17, 并不是全部都是正例
 - 2. 好瓜 \leftarrow 色泽 = 青绿 \wedge 根蒂 = 蜷缩 覆盖了 1、4、17, 并不是全部都是正例
 - 3. 好瓜 \leftarrow 色泽 = 青绿 \wedge 根蒂 = 蜷缩 \wedge 敲声 = 浊响 覆盖了 1, 此时都是正例，就把这条规则加到规则集中，然后继续用剩下的样例进行训练
- 缺陷：基于穷尽搜索，在属性和样例数量较多的时候可能会组合爆炸

2.3 自顶向下与自底向上

自顶向下 top down: 也就是上面这个过程, 从一般到特殊, 覆盖范围从大到小, 其泛化性能更好 (理由是更容易产生能较短的能覆盖更多正例的规则)

自底向上 bottom up: 从特殊到一般, 覆盖范围从小到大, 适用于一阶学习这种假设空间比较复杂的情况 (理由是文字可能会很多, 从一般到特殊需要遍历很多可能)

2.4 以西瓜数据集 2.0 的训练集为例的自顶向下

详见 TopDown 函数的输出

3 15.3 剪枝优化

3.0.1 基本思想

通过剪枝前后发生的性能变化来判断是否进行剪枝, 可以缓解过拟合的风险

规则生成本质上是一个贪心搜索过程, 也就是说, 每一步都去寻找一个局部最优解, 分阶段去逼近最优解。例如上面的分析过程, 我们每次只是一些规则里面找到最好的规则 (而不是去找全部的可能的规则进行比较), 然后把它加进规则集里面, 最后我们得到的规则集不一定是最好的解, 但可能会是一个可行解。我们把这个局部最优解当成全局最优解来使用, 可能会造成过拟合, 于是通过剪枝来提高规则集的泛化性能。

预剪枝是指生长过程中剪枝, 后剪枝是指规则产生后剪枝。

3.0.2 CN2 算法

在预剪枝时, 假设用规则集预测必须显著优于直接用训练集的后验概率分布 (也就是训练集正反例数量的比率) 进行猜测。使用了似然率统计量 LRS 来表示两种预测方法之间的差别。

$$LRS = 2(\hat{m}_+ \log_2 \frac{(\frac{\hat{m}_+}{\hat{m}_+ + \hat{m}_-})}{(\frac{m_+}{m_+ + m_-})} + \hat{m}_- \log_2 \frac{(\frac{\hat{m}_-}{\hat{m}_+ + \hat{m}_-})}{(\frac{m_-}{m_+ + m_-})}) \quad (3)$$

当 LRS 越大的时候，说明规则集预测与直接用训练集分布进行猜测的差别越大。在数据量比较大的现实任务中，通常设置 LRS 很大（例如 0.99）时才停止。

通过查找 CN2 算法的论文发现，CN2 算法汲取了 ID3 算法和集束搜索的方法两种方法的优点。集束搜索每轮留下 b 个最优的选择，使得不会过于贪心，CN2 也这样做；而 ID3 算法则是使用信息熵来描述每个分类的混乱程度，越不混乱的越先分类，CN2 则也定义了一个描述规则预测结果混乱程度的“熵”：

$$Entropy = - \sum_{i=1}^n p_i \log_2(p_i) \quad (4)$$

用于衡量规则的好坏，其中 p_i 是指规则覆盖的样例当中某一类的数量占总数的比例，也就是某一类样例在结果中的频率。那么当然是如果某个 p_i 很接近 1 的话，那么熵接近 0，如果大家都差不多大的话，那么熵就比较大，这个在决策树里面讲过了我这里就不再赘述。

而 LRS 作为 CN2 算法的终止条件，论文也似乎和课本当中说的不太一样。论文中的 LRS 是这样定义的：

$$LRS = 2 \sum_i f_i \log_2\left(\frac{f_i}{e_i}\right) \quad (5)$$

其中 f_i 是指某一类样例在结果中的频率， e_i 是指某一类样例在整个训练集中的比例。如果两者差不多，那么 LRS 接近于 0，说明用这个规则判断跟用原本训练集的分布来猜差不多，那么结果也就不太可靠 (reliable) 了。这里代码是用论文里的定义。

CN2 算法的具体过程和 TopDown 方法类似，详见代码和论文。

3.0.3 REP, IREP 与 IREP*

减错剪枝 REP 是后剪枝常用的策略，其基本做法是：将样例集划分为训练集 T 和验证集 V ，从 T 上学得规则集 R 后进行多轮剪枝，每一轮穷举可能的剪枝操作，包括删除规则中的某个或多个文字，删除整条规则等，然后用 V 对剪枝产生的规则集进行评估，保留最好的规则集进行下一轮剪枝，直到无法通过剪枝提高验证集上的性能为止。（这应该是后剪枝能想到的最简单的做法，也就是去遍历所有的剪枝可能，找到最好的剪枝结果）

由于 REP 的算法复杂度是 $O(m^4)$ ，有效但复杂度太高了。IREP 把复杂度降到 $O(m \log^2 m)$ ，其做法是：在生成每条规则前，先将当前样例集划

分为训练集和验证集，在训练集上生成一条规则 r ，立即在验证集上对其进行 REP 剪枝，得到规则 r' ；将 r' 覆盖的样例去除，在更新后的样例集上重复上述过程。显然，REP 是针对规则集进行（遍历）剪枝，而 IREP 仅对单条规则进行（遍历）剪枝，因此后者比前者更加高效。

前面两个算法的性能度量指标都是准确率，而 IREP* 改用了

$$\frac{\hat{m}_+ + (m_- - \hat{m}_-)}{m_+ + m_-}$$

作为性能度量指标，在剪枝时删除规则尾部的多个文字，并在最终得到规则集之后再进行一次 IREP 剪枝。

3.0.4 RIPPER 算法

RIPPER 算法是将剪枝机制和后处理优化相结合的规则生成算法，其泛化性能也更好。首先是用 IREP* 算法在训练集 D 上生成规则集 R ，然后 $\text{PostOpt}(R)$ ，也就是遍历 R 中的每一条规则 r ，用 r 覆盖的样例和 IREP* 方法重新生成一条规则 r' ，成为替换规则；用 r 增加文字进行特化，然后再用 IREP* 方法重新生成一条规则 r'' ，成为修订规则然后分别用 r', r'' 替换 r ，形成 R', R'' ，保留最优的规则集。之后把没有覆盖到的样例再重新进行相同的训练。

RIPPER 算法就是将所有规则放在一起重新优化，恰恰是通过全局考虑缓解了之前贪心算法的局部性，也就是之前那种每条规则生成之后都没有对后面产生的规则加以考虑。