使用答案集编程对决策树模型进行形式化解释

AkihiroTakemura Masayuki Otani国立信息学研究所，日本东京 东京工业大学，
日本东京 atakemura@nii.ac.jp otani.m.af@m.titech.ac.jp

Katsumi Inoue国立信息学研究所，
日本东京 inoue@nii.ac.jp

决策树模型（包括随机森林和梯度提升决策树）凭借出色的预测性能，在机器学习中得到了广泛应用。然而，其复杂的结构往往使其难以解释，尤其是在安全关键型应用中，模型决策需要得到正式论证。近期研究表明，可以通过自动推理技术推导出逻辑解释和溯因解释。本文提出一种方法，利用答案集编程（ASP）生成多种类型的解释，即充分解释、对比解释、多数解释和树特定解释。与基于 SAT 的方法相比，我们基于 ASP 的方法在编码用户偏好方面更加灵活，并支持枚举所有可能的解释。我们在多种数据集上对该方法进行了实证评估，并展示了其相较于现有方法的有效性和局限性。

1 引言

决策树模型在机器学习中的广泛应用，得益于其高准确率和计算效率。然而，更复杂的集成变体，例如随机森林和梯度提升树，往往以黑箱模型的形式运行，使其内部决策过程对用户而言不透明 [12]。在医疗保健、金融和刑事司法等高风险应用场景中，这种缺乏透明度的问题尤为突出，因为人们对模型决策的信任取决于能否清楚了解模型如何做出预测。

可解释性（常称为可解释人工智能，简称 XAI）旨在通过让模型行为变得易于理解来弥合这一差距 [12]。然而在实践中，由于现代机器学习模型十分复杂，要生成忠实的解释颇具挑战。简单或基于启发式的方法往往难以奏效，因为它们无法捕捉模型决策背后的完整逻辑结构 [23, 19]。

LIME [21]、SHAP [18] 和 Anchors [22] 等知名事后解释方法依赖启发式近似，因此生成的解释不一定与模型的真实行为一致 [20]。相比之下，基于逻辑的方法能够提供严格的保证，并已成功用于生成多种类型的解释，包括充分解释和对比解释 [15, 16, 14,4, 5, 3, 2]。这些基于逻辑的方法主要采用布尔可满足性（SAT）技术。然而，基于 SAT 的方法往往缺乏灵活性，尤其是在纳入用户定义的偏好或支持解释枚举等复杂推理任务方面。

在本研究中，我们提出了一种替代方法，即使用答案集编程（ASP）——一种适用于组合推理的声明式逻辑编程范式。我们展示了如何利用 ASP 来

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
第41届国际逻辑编程会议（ICLP 2025）EPTCS439，2026，第420–437页，doi:10.4204/EPTCS.439.29

© A. Takemura, M. Otani & K. Inoue
本作品根据知识共享署名许可协议授权。

为决策树模型编码并计算不同类型的解释，包括充分解释、对比解释、多数解释和树特定解释。ASP 的高度表达能力使我们能够简洁地描述各种解释类型，并灵活地纳入约束，因此它是形式化可解释性的有力工具。

我们的主要贡献如下：

1. 我们为多种解释类型开发了 ASP 编码，包括充分解释、对比解释、多数解释和树特定解释。2. 我们在 20 个数据集上实现并评估了我们的方法，并将其性能与现有的基于 SAT 的方法进行比较。3. 我们分析了基于 ASP 的方法的优势和局限，指出了其擅长之处（多数解释枚举）以及面临的挑战（扩展到大型森林的可扩展性）。

本文组织如下：第 2 节介绍决策树模型和 ASP 的必要背景。第 3 节介绍我们用于生成各种类型解释的基于 ASP 的编码。第 4 节详细说明实验设置和结果，并将我们的方法与基于 SAT 的方法进行比较。第 5 节讨论决策树模型可解释人工智能领域的相关工作。第 6 节给出结论。此外，附录提供了例题和可视化示例，展示我们的方法应用于决策树、随机森林和提升树模型。

2 背景

本节介绍本文其余部分所需的概念。我们首先回顾布尔函数及其与机器学习背景下逻辑推理的关系，以及决策树模型的结构。随后，我们简要概述 ASP——一种声明式框架，在我们的方法中用于编码和计算形式化解释。

2.1 布尔函数

设 $\mathbf{X}_n = \{\mathbf{x}1, \ldots, \mathbf{x}_n\}$ 为一组布尔变量。布尔函数是一个映射 $\mathbf{f}: \{0,1\}^n \to \{0,1\}$，它为每个布尔赋值 $\mathbf{xxx} \in \{0,1\}^n$ 指定一个布尔值。若 $\mathbf{f}(\mathbf{xxx}) = 1$，则称 xxx 为正实例；若 $\mathbf{f}(\mathbf{xxx}) = 0$，则称其为负实例。在 $\mathbf{X}_n$ 上构造命题公式时，使用逻辑连接词：合取（$\wedge$）、析取（$\vee$）和否定（$\neg$）。文字 L 可以是变量 $\mathbf{x}_i$（正文字），也可以是它的否定 $\neg\mathbf{x}_i$（负文字）。项是文字的合取。

对于赋值 $\mathbf{zzz} \in \{0,1\}^n$，$\mathbf{t_z} = \bigwedge_{i=1}^{n} \mathbf{x}^{z_i}_i$ 表示其对应的项，其中 $\mathbf{x}_i^0 = \neg\mathbf{x}_i$ and $\mathbf{x}_i^1 = \mathbf{x}_i$。如果项 $\mathbf{t}_1$ 中出现的文字也出现在项 $\mathbf{t}_2$ 中，则称 $\mathbf{t}_1$ 是项 $\mathbf{t}_2$ 的子项。例如，$\mathbf{x}_1 \wedge \mathbf{x}_3$ 是 $\mathbf{x}_1 \wedge \neg\mathbf{x}_2 \wedge \mathbf{x}_3$ 的子项。如果项 t 是 $\mathbf{t_z}$ 的子项，也就是说，在赋值 zzz [2]下 t 被满足，则称 t 覆盖 zzz。这体现了这样一种思想：只要赋值 zzz 满足 t 中的所有文字，t 就成立。如果一个项 t 蕴含命题公式 f，即 $\mathbf{t} \models \mathbf{f}$，那么称 t 是 f 的蕴含项；也就是说，每个满足 t 的赋值 $\mathbf{zzz} \in \{0,1\}^n$ 也满足 f。如果项 t 蕴含 f，即 $\mathbf{t} \models \mathbf{f}$，且 t 的任何真子项都不蕴含 f，那么称 t 是 f 的主蕴涵项。

2.2 树与森林

我们考虑有限个属性（或特征）的集合 $\mathscr{A} = \{\mathbf{A}1, \ldots, \mathbf{A}_n\}$，其中每个属性 $\mathbf{A}_i$ 的取值来自一个定义域 $\mathbf{D}_i$，该定义域可以是布尔型、分类型或数值型。决策树学习算法会创建布尔测试：针对数值属性的阈值测试（$\mathbf{A}_i \leq \mathbf{c}$）以及针对

类别属性的隶属测试。令 $\mathrm{L} = \{\mathrm{L1}, \ldots, \mathrm{L}_n\}$ 表示树节点测试中的布尔文字，其中每个 L_i 对应于 X_n 中的一个布尔变量。实例 $\mathrm{zzz} = (\mathrm{v1}, \ldots, \mathrm{v}_n)$，其中 $\mathrm{v}_i \in \mathrm{D}_i$，根据树节点测试对每个布尔变量 x_i 进行求值。对应的项为 $\mathrm{t_z} = \bigwedge_{ni=1} \mathrm{x}^{z_i}{}_i$，其中，如果针对 x_i 的布尔测试结果为真，则 $\mathrm{z}_i = 1$；如果结果为假，则 $\mathrm{z}_i = 0$。

决策树。针对 $\mathscr{A}$ 的决策树分类器是一棵二叉树 T，其中内部节点用于测试布尔条件，叶节点则标记为 $\{0,1\}$ 中的类别。树遍历从根节点开始，条件为真时沿左分支前进，条件为假时沿右分支前进。

随机森林。随机森林 $\mathrm{RF} = \{\mathrm{T1}, \ldots, \mathrm{T}_m\}$ 是由决策树 T_is 组成的集合，其输出由多数投票 [6] 决定：

$$RF(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{m} T_i(\boldsymbol{x}) > \frac{m}{2}, \\ 0 & \text{otherwise.} \end{cases}$$

回归树与森林。回归树与决策树类似，不同之处在于叶节点标记为实值输出。森林 $\mathrm{F} = \{\mathrm{T1}, \ldots, \mathrm{T}_m\}$ 由 m 棵回归树组成，其聚合权重为 $\mathrm{w(F, xxx)} = \sum_{k=1}^{m} \mathrm{w(T}_k\mathrm{, xxx)}$。

提升树。用于二分类的提升树 $\mathrm{BT} = \{\mathrm{T1}, \ldots, \mathrm{T}_m\}$ 是一个树序列 [9]（通常是回归树，例如 XGBoost [7] 等梯度提升树），其预测由累积权重的符号决定。

$$BT(\boldsymbol{x}) = \begin{cases} 1 & \text{if } w(BT,\boldsymbol{x}) > 0, \\ 0 & \text{if } w(BT,\boldsymbol{x}) < 0, \end{cases} \quad \text{where } w(BT,\boldsymbol{x}) = \sum_{k=1}^{m} w(T_k,\boldsymbol{x}).$$

我们将森林 F 和提升树 BT 的大小定义为 $|\mathrm{F}| = |\mathrm{BT}| = \sum_{k=1}^{m} |\mathrm{T}_k|$，其中 $|\mathrm{T}_k|$ 表示树 T_k 中的节点数。

2.3 答案集编程

答案集编程（ASP）是一种面向解决困难组合问题的声明式编程形式。它允许将问题编码为一组逻辑规则，并使用求解器根据稳定模型语义 [11]计算其解，这些解称为答案集。

典型的 ASP 规则写作如下：

$$\mathrm{a_1 \ :\!-\ a_2,\ \ldots,\ a_m,\ not\ a_{m+1},\ \ldots,\ not\ a_n.}$$

这里，每个 $\mathrm{a_i}$ 都是一个一阶原子，而 not 表示默认否定。只有头部 $\mathrm{a_1}$ 的规则称为事实；没有头部的规则则称为完整性约束，用于消除不期望的模型。

现代 ASP 系统（如 clingo [10], ）支持一些表达力强的构造，可简化问题建模，例如：

- 形式为 {a(X) :b(X)} 的条件文字，表示满足 b(X) 的所有 X 值所对应的原子 a(X) 集合。
- 形如 $\mathrm{s_1}$ {a(X) :b(X)} s2 的基数约束，用于限制集合中真文字的数量，使其介于 s1 和 s2 之间。

- 汇总 #count 和 #sum 等聚合函数分别用于对元素进行计数和求和 .

这些构造能够紧凑、清晰地编码复杂的搜索空间，使 ASP 特别适用于形式化解释。此外，#heuristic 指令可为求解器的原子选择提供偏好设置。更多详情请参阅 clingo 用户指南。[1]

3 决策树模型解释

本节介绍基于 ASP 的编码，用于从决策树模型中提取不同类型的解释。我们考察三种模型类型（决策树、随机森林和梯度提升树）中的四种解释类型（充分、对比、多数和树特定解释）。我们对每种解释类型进行形式化定义，并给出相应的 ASP 编码。

我们的 ASP 编码通过忠实表示决策树模型的结构和决策流程，保留其逻辑语义；完整性约束则确保只产生有效的解释。

表 1 总结了本文中每种模型类型支持的解释类型。

模型	充分	对比	多数	树专用
决策树	✓	✓	n/a	n/a
随机森林	✓	✓	✓	n/a
XGBoost	-	-	n/a	✓

表 1：我们的 ASP 编码针对每种模型类型所支持的解释类型。“-”表示尚未实现；“n/a”表示不适用。

3.1 决策树

我们先讨论决策树，其中每个模型都会被转换为一组 ASP 事实，用于表示其结构和预测逻辑。本小节将定义每种解释类型，并给出相应的 ASP 编码。预处理脚本引入的谓词列于表 2，其中 node/3 和 pre class/1 依赖于实例（真值取决于具体实例），而 leaf node/2 等结构谓词则独立于实例。在各种编码中，包括随机森林和提升树，我们都使用 next node/2 谓词来模拟从根节点到叶节点的路径遍历。该谓词根据选定文字有条件地重构决策流程，使求解器能够检查某个解释是否保留或改变预测，从而模拟决策树得出输出的过程。示例见附录 2 和程序 8。

谓词	含义
节点(X,L,B)	节点 X 检查当前实例中文字 L 的真值是否为 B。
叶节点(X,C)	节点 X 是一个预测类别 C 的叶节点。
左节点(X,LX)	节点 X 的左子节点为 LX。
right node(X,RX)	节点 X 的右子节点为 RX。
预类别(P)	该实例的预测类别为 P。

表 2：用于将决策树模型编码为 ASP 事实的谓词。

[1] https://github.com/potassco/guide/releases/

3.1.1 决策树的充分解释

定义 1（充分解释 [2]）设 T 是定义在布尔变量 $X_n = \{x_1, \ldots, x_n\}$ 上的一棵决策树，设 f 是 T 所表示的布尔函数，且 $xxx \in \{0,1\}^n$ 是一个项为 t_x 的实例。给定 f，将 xxx 分类为 1（分别为 0）的充分解释，是 t_x 的一个子项 t，且 t 是 f（分别为 $\neg$ f）的主蕴涵项。

根据定义，改变主蕴涵项中的任何文字，都会改变相应布尔函数的输出。基于此，可以通过找到不影响输出的可变化文字的极大集合，再取其补集，来获得充分解释。规则 node(X,L,0..1)（第 2 行）允许沿两个方向进行条件遍历，而无效谓词（第 8 行）则用于检测输出变化。

程序 1：决策树的充分解释

```
1  1 { selected_文字 (L): node (X ,L ,B)}.
2  node (X ,L ,0..1)      :- 选定_文字 (L)                , 节点 (X ,L ,B)。
3  下一个_节点 ( LX )  : - 节点 (0 , L ,1) , 左侧_节点 (0, LX )。
4  下一个_节点 ( RX )  :- 节点 (0 , L ,0) , 右侧_节点 (0 , RX )。
5  下一个_节点 (LX ) :- 下一个_节点 (X)      , 节点 (X ,L ,1) , 左_节点 (X , LX ).
6  下一个_节点 ( RX )  :- 下一个_节点 (X) , 节点 (X ,L ,0) , 右_节点 (X , RX ).
7  类别 (C):- 下一个_节点 (X) , 叶节点_节点 (X ,C).
8  无效        :- 类（0）     , 类（1）.
9  :- invalid .          %% 删除会改变预测的解
10 #heuristic selected_文字 (L).              [1 , true ]    %% 优先包含              更多文字
11 #显示 selected_文字 /1.
```

此编码将利用 clingo 中的 #heuristic 指令查找选定文字的极大集合，该指令会优先将选定文字赋值为真。示例请参见附录示例 1。 -

3.1.2 决策树的对比解释

与充分解释不同，我们要寻找的是一个极小文字集合，它能使布尔函数的输出发生变化。

定义 2（对比解释 [2]）设 T 是定义在布尔变量 $X_n = \{x_1, \ldots, x_n\}$ 上的决策树，设 f 是 T 表示的布尔函数，且 $xxx \in \{0,1\}^n$ 是一个具有项 t_x 的实例。对于 xxx，给定 f，将其分类为 1（分别为 0）的对比解释是 t_x 的一个子项：从 t_x 中移除该子项中的文字后，所得项不是 f（分别为 $\neg$ f）的蕴含项；并且对于每个文字 l $\in$t，从 t_x 中移除除 l 以外的所有文字后，不满足这一条件。

因此，我们通过引入有效谓词（第 8 行）来允许预测发生这种变化；根据下一行的完整性约束，该谓词不能为假。

程序 2：从决策树得到对比解释

```
1  1 { 所选文字 (L):  节点 (X ,L ,B)}.
2  节点 (X ,L ,0..1)      :- 所选文字 (L)              , 节点 (X ,L ,B).
3  下一个_节点 ( LX )  :- 节点 (0 , L ,1) , 左_节点 (0, LX ).
4  下一个_节点 ( RX )  :- 节点 (0 , L ,0) , 右_节点 (0 , RX ).
5  下一个_节点 (LX ) :- 下一个_节点 (X)      , 节点 (X ,L ,1) , 左_节点 (X , LX ).
6  下一个_节点 ( RX )  :- 下一个_节点 (X) , 节点 (X ,L ,0) , 右侧_节点 (X , RX ).
7  类别 (C)      :- 下一个_节点 (X) , 叶_节点 (X ,C).
```

```
有效    :- 类（0）    , 类（1）.
:- not valid .          %% the output must change
#heuristic 已选_文字 (L).                    [1 , false ]    %% 优先包含                    更少的文字
#show 已选_文字 /1.
```

此编码将利用 clingo 中的 #heuristic 指令，优先将选定文字赋值为假，从而找到最小的选定文字集合。示例请参见附录示例 2。 -

3.1.3 例题：决策树中的充分解释

考虑图 1 中的决策树，其中实例为 $\mathbf{xxx} = \{x_1 = 1, x_2 = 1, x_3 = 2\}$。

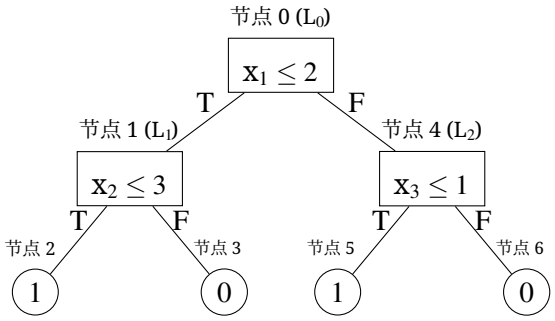

图 1：布尔属性的决策树示例。

预处理将这棵树转换为布尔文字：($x_1 \leq 2$) 变为文字 L_0 (true)，($x_2 \leq 3$) 变为文字 L_1 (true)，而 ($x_3 \leq 1$) 变为文字 L_2 (false)。该实例沿左-左路径前进，到达预测为 1 的叶节点 2。ASP 编码将此实例表示为一组事实：

```
pre_类别（1）。node（0，0，1）。node（1，1，1）。node（4，2，0）。leaf_node（2，1）。
leaf_node（3，0）。leaf_node（5，1）。leaf_node（6，0）。left_node（0，1）。right_
node（0，4）。% ... 其他事实
```

对于充分解释，程序 1 生成选定文字(2)，这意味着文字_L_0 和 L_1构成了充分解释 $\{x_1 \leq 2, x_2 \leq 3\}$。

3.2 随机森林

随机森林是由决策树组成的集成，其最终预测由多数投票决定。我们将介绍如何使用 ASP 从随机森林中推导出充分解释、对比解释和多数解释。每棵树的编码方式与前文所述相同，此外还加入了用于跟踪选定文字和树之间一致性的谓词。表3列出了用于表示森林结构和控制解释逻辑的谓词。

3.2.1 随机森林的充分解释

定义 3（充分解释 **[3]**）设 $\mathrm{RF} = \{T_1, \ldots\ldots, T_m\}$ 是定义在布尔变量 $\mathbf{X}_n = \{x_1, \ldots\ldots, x_n\}$上的随机森林，f 是 RF 表示的布尔函数。设 $\mathbf{xxx} \in \{0,1\}^n$ 是一个实例，其对应项为 $t_{\mathbf{x}}$。给定 f，要将 xxx 分类为 1（分别为 0）的充分解释是 $t_{\mathbf{x}}$ 的一个子项，且 t 是 f（分别为 $\neg$f）的主蕴涵项。

谓词	含义
node(T,X,L,B)	在树 T 中, 节点 X 检查文字 L 对此实例的真值是否为 B。
class(T,C)	树 T 预测此实例的类别为 C。
feature(L,B)	文字 L 对此实例的真值为 B。
new_node(T,X,L,B)	真值可能已翻转的节点副本。
下一个_节点(T,X)	在可能变化的真值下可到达的节点。
叶节点(T,X,C)	在树 T 中，节点 X 是一个预测类别 C 的叶节点。
左节点(T,X,LX)	树 T 中节点 X 的左子节点是 LX。
右节点(T,X,RX)	树 T 中节点 X 的右子节点是 RX。
原始_森林(P)	森林对 xxx 的预测类别是 P。
树阈值(TH)	充分解释中的多数阈值。
对比树阈值(TH)	对比解释中的多数阈值。
多数树阈值(TH)	多数解释的阈值（例如，$\lceil m/2 \rceil$）。

表3：用于将随机森林模型编码为 ASP 事实的谓词。

最小不可满足子集（MUS）是一组强制不存在答案集的原子。对于随机森林，充分解释需要固定一组最小原子，以阻断所有反事实。我们采用两阶段方法，使用 WASP [1] 进行不可满足核提取：阶段1搜索会改变森林预测的反事实；阶段2计算必须固定以阻止这种变化的一组文字的 MUS。

下面的第一个编码定义了森林的逻辑结构，并检测实例中的变化是否会导致预测结果不同。第二个编码使用 WASP的--mus模式提取固定文字的最小集合。接下来的编码通过识别一个极大文字集合来计算充分解释，使得足够多的树能够保持该预测。

程序 3：随机森林阶段1的充分解释

```
节点 (T ,X ,L)      :-  节点 (T ,X ,L    , _ ) .
new_节点 (T ,X ,L ,0)      :-  变化 (L) ,  节点 (T ,X ,L) ,  特征 (L ,1) 。
new_节点 (T ,X ,L ,1)      :-  变化 (L) , 节点 (T ,X ,L) , 特征 (L ,0) .
新_节点 (T ,X ,L ,B)        :-  节点 (T ,X ,L) , 特征 (L ,B) ,非                变化 (L).
下一个_节点 (T , LX ) :- 新_节点 (T ,0,L ,1) , 左_节点 (T ,0 , LX ).
下一个_节点 (T , RX )    :-  新建_节点 (T ,0 ,L ,0) , 右侧_节点 (T ,0 , RX ).
下一个_节点 (T , LX ) :- 下一个_节点 (T ,X), 新建_节点 (T,X ,L ,1) , 左侧_节点 (T ,X , LX ).
下一个_节点 (T , RX )    :-  下一个_节点 (T ,X ) , 新建_节点 (T ,X ,L ,0) , 右侧_节点 (T ,X , RX ).
类别 (T ,C) :- 下一个_节点 (T ,X) , 叶节点_节点 (T,X,C).
无效_树 (T)              :-  类别 (T ,C) , pre_森林 ( FC ) ,C !=FC .
森林_变化                :- VT =# count {T: invalid_树 (T)}, 树_阈值 (TH ) ,VT >TH .
:- not 森林_变化 .
```

第二阶段基于阶段1中定义的反事实结构，计算必须固定哪些文字，才能防止森林预测发生变化，从而得到这些文字的最小不可满足子集。具体做法是调用带有 --mus模式的 WASP，从猜测的固定文字子集中提取最小不可满足核心。

程序4：随机森林的充分解释——阶段2

```
特征 (I,B)。...                %% 列表   特征                参与其中的              森林预测
fix_lit (I)。...              %%  猜测          用于 MUS 提取的原子          提取          （参见   附录）
```

程序4展示了高层结构。feature/2事实表示原始实例，而 fix lit/1 用于猜测哪些文字必须保留其原始值。此方法使我们能够提取出一个

为保留原始预测所必需的最小文字集合，从而有效地计算出充分解释。示例请参见附录示例 3。

3.2.2 来自随机森林的对比解释

定义 4（对比解释 **[3]**）设 RF $=\{T_1、\ldots\ldots、T_m\}$ 为定义在布尔变量 $X_n=\{x_1、\ldots\ldots、x_n\}$ 上的随机森林，设 f 为 RF 所表示的布尔函数，并设 $xxx \in \{0,1\}^n$ 为一个关联项 t_x 的实例。对于给定的 f，要将 xxx 分类为 1（相应地为 0）的对比解释，是 t 的一个子项：从 t_x 中移除 t 中的文字后所得的项不是 f 的蕴含项（相应地，不是 $\neg$ f 的蕴含项），并且对于每个文字 $l \in t$，从 t 中移除 l 后都不满足这一条件。

此编码搜索一个最小文字集，移除这些文字后会使随机森林输出发生变化。selected literal/1 谓词确定发生变化的文字，而 con tree threshold/1 强制要求数量的树给出不一致结果。

程序 5：随机森林的对比解释

```
1  1 { selected_文字 (L): 节点 (T ,X ,L ,B)}.
2  new_节点 (T ,X ,L ,0)      :- selected_文字 (L)              , 节点 (T ,X ,L ,1) .
3  new_节点 (T ,X ,L ,1)      :- selected_文字 (L)              , 节点 (T ,X ,L ,0) .
4  new_节点 (T ,X ,L ,B) :- 节点 (T ,X ,L ,B),not selected_文字 (L).
5  next_节点 (T , LX ) :- new_节点 (T ,0,L ,1) , left_节点 (T ,0 , LX ).
6  next_节点 (T , RX )    : -  new_节点 (T ,0 ,L ,0) , right_节点 (T ,0 , RX ).
7  下一个_节点 (T , LX ) :- 下一个_节点 (T ,X), 新_节点 (T,X ,L ,1) , 左侧_节点 (T ,X , LX ).
8  下一个_节点 (T , RX )    : -  下一个_节点 (T ,X ) , 新_节点 (T ,X ,L ,0) , 右侧_节点 (T ,X , RX ).
9  类别 (T ,C):- 下一个_节点 (T ,X) , 叶_节点 (T ,X ,C).
10 有效_树 (T)          : -  类别 (T ,C) , 前置_森林 ( FC ) ,C != FC .
11 valid :- VT   =# count {T : valid_tree (T)}, con_tree_阈值 ( TH ),VT > TH .
12 : -  not  valid .
13 #启发式选择的_文字 (L)。                    [1 , false ]
14 #show selected_文字 /1.
```

3.2.3 来自随机森林的多数解释

定义 5（多数解释 **[3]**）设 RF $=\{T_1, \ldots, T_m\}$ 是一个定义在布尔型变量 $X_n=\{x_1, \ldots, x_n\}$ 上的随机森林，f 是 RF 表示的布尔函数，$xxx \in \{0,1\}^n$ 是一个关联项 t_x 的实例。对于给定的 f，将 xxx 分类为 1（分别为 0）的多数解释是 t_x 的一个子项 t，满足：若 RF(xxx) $=1$，则至少有 $\lfloor m/2 \rfloor + 1$ 棵树 T_i 的蕴含项包含 t（若 RF(xxx) $=0$，则至少有 $\neg T_i$ 棵树的蕴含项包含 t）；并且对于 $t \in$ 中的每个文字 l，从 t 中移除 l 后不再满足此条件。

此编码通过统计与森林原始预测一致的树，寻找一个在翻转后不会违反多数投票的极大项 t。该解释必须保留足够多的单棵树的一致性，以确保森林预测保持一致。

程序 6：随机森林的多数解释

```
1  1 { selected_文字 (L): 节点 (T ,X ,L ,B)}.
2  节点 (T ,X ,L ,0..1)       : -  已选_文字 (L)              , 节点 (T ,X ,L ,B)。
3  下一个_节点 (T , LX ) :- 节点 (T ,0,L ,1) , 左侧_节点 (T ,0 , LX )。
4  下一个_节点 (T , RX )    : -  节点 (T ,0 ,L ,0) , 右侧_节点 (T ,0 , RX )。
5  下一个_节点 (T , LX ) :- 下一个_节点 (T ,X),节点 (T,X,L ,1) , 左侧_节点 (T ,X , LX )。
6  下一个_节点 (T , RX )    : -  下一个_节点 (T ,X ) , 节点 (T ,X ,L ,0) , 右侧_节点 (T ,X , RX ).
7  类别 (T ,C) :- 下一个_节点 (T ,X) , 叶节点_ (T,X,C).
```

```
8 invalid_树 (T) :- 类别 (T ,C) , pre_森林 (FC ) ,C !=FC . 9 valid :- VT   =# count {T : invalid_树 (T)},
majo_树_阈值 ( TH ) ,VT <TH . 10 :- not valid . 11 # heuristic selected_文字 (L). [1 , true ]12 # sho
w selected_文字 /1.
```

该编码使用 majo tree threshold/1 指定为保留原始森林预测所需的一致树的最小数量。# heuristic 指令鼓励包含文字，从而生成一个极大项，再对其进行筛选以识别最小多数解释。由于该编码会找出可翻转文字的极大集合，因此实际的多数解释是其补集：为保留多数投票而必须保持不变的文字。示例请参见附录示例 5。

3.3 提升树

现在我们转向提升树模型，例如 XGBoost [7] 和 LightGBM [17], ，它们通过聚合回归树的输出进行预测。设 BT $=\{T1,\ldots,T_m\}(1\leq i\leq m)$ 为由 m 棵树组成的提升树模型，其中每个叶节点贡献一个实值权重，最终预测由权重和的符号决定。表 4 列出了用于将提升树编码为 ASP 事实的谓词。

谓词	含义
node(T,X,L,B)	在树中, T, 节点 X 检查文字 L 在当前实例中的真值是否为 B。
叶节点_node(T,X,W)	树 T 中的节点 X 是一个权重为 W 的叶节点。
左节点(T,X,LX)	树 T 中节点 X 的左子节点是 LX。
右节点(T,X,RX)	树 T 中节点 X 的右子节点是 RX。
原始森林(P)	森林对 X 的原始预测为 $P\in\{0,1\}$。
weight(T,W)	在树 T 中，当前选择所到达的叶节点权重。
best weight(T,BW)	树 T 中可达到的最大权重。
worst weight(T,WW)	树 T 中可达的最小权重。

表 4：用于将提升树模型编码为 ASP 事实的谓词。

3.3.1 提升树的树特定解释

提升树的树特定解释用于确定最小文字集，这些文字集可在最佳或最坏情况下保证某项预测。

定义6（最佳与最差实例 **[4]**）令 BT $=\{T_1,\ldots,T_m\}$ 为定义在布尔型变量 $X_n=\{x_1,\ldots,x_n\}$ 上的提升树，并令 xxx、xxx′、xxx″ $\in\{0,1\}^n$ 为对应项分别为 t_{xxx}、$t_{xxx'}$、$t_{xxx''}$ 的实例，且假设 BT (xxx) $=$BT (xxx′) $=$BT (xxx″)。

- 扩展 t 的最佳实例是任意满足以下条件的 xxx′： t 是 $t_{xxx'}$ 的子项，BT (xxx′) $=$BT (xxx)，且 xxx′ $=\arg\max_{xxx''}$:t 是 $t_{xxx''}$({w(BT, xxx″)})。$w\uparrow$(t, BT) 是任意最佳实例的权重，即 $w\uparrow$(t,BT) $=\max_{xxx''}$:t 是 $t_{xxx''}$ w(BT, xxx″)。
- 使 t 成为其子项的所有实例中，最差实例是任意一个 xxx′，满足 t 是 $t_{xxx'}$ 的子项，BT (xxx′) $=$ BT (xxx)，且 xxx′ $=\arg\min_{xxx''}$: t 是 $t_{xxx''}$({w(BT, xxx″)}) 的子项。$w\downarrow$(t, BT) 是任意最差实例的权重，即 $w\downarrow$(t,BT) $=\min_{xxx''}$: t 是 $t_{xxx''}$ w(BT, xxx″)。

最佳（分别为最差）实例代表最有利（分别为最不利）的情况，从而可以在悲观条件下推理预测保证。

定义 7（树特定解释 [4]）设 BT $= \{T_1、\ldots、T_m\} (1 \leq i \leq m)$ 为一棵提升树，xxx 为输入实例。形式化地，令 $w\downarrow(t, T_i)$ 表示树 T_i 在所有扩展 t 的实例中所能产生的最低（最坏情况）分数，$w\uparrow(t, T_i)$ 表示最高（最好情况）分数。那么，如果 t 满足以下条件，它就是实例 xxx 的树特定解释：

- 当 BT (xxx) $= 1$：t 是 txxx 的子项，并且 $\Sigma mi_{=1} w\downarrow(t,Ti) > 0$，且 t 没有满足后一条件的真子项。
- 当 BT (xxx) $= 0$：t 是 txxx 的子项，并且 $\Sigma mi_{=1} w\uparrow(t,Ti) \leq 0$，且 t 没有满足后一条件的真子项。

程序 7：来自提升树的树特定解释

```
1 1 { selected_文字 (L) : node (T ,X ,L ,B) }. 2 node (T ,X ,L ,0..1) :- node (T ,X ,L ,B), not
selected_文字 (L). 3 next_node (T , LX ) :- node (T ,0 ,L ,1) , left_node (T ,0 , LX ). 4 next_node (T ,
RX ) :- node (T ,0 ,L ,0) , right_node (T ,0 , RX ). 5 next_node (T , LX ) :- next_node (T ,X), node
(T,X,L ,1) , left_node (T ,X , LX ). 6 next_node (T , RX ) :- next_node (T ,X), node (T,X,L ,0) , right_
node (T ,X , RX ). 7 weight (T ,W) :- next_node (T ,X) , leaf_node (T,X,W ). 8 best_weight (T , BW ) :-
weight (T , _), BW    =# max {W : weight (T ,W) }. 9 worst_weight (T , WW ) :- weight (T , _),
WW    =# min { W: weight (T ,W) }. 10 valid :- SW    =# sum { BW : best_weight (_ ,BW )}, SW  <=0,
pre_forest (0) . 11 valid :- SW    =# sum { WW : worst_weight (_ ,WW )}, SW  >0, pre_forest (1) .
12 :- not valid . 13 # heuristic selected_文字 (L).  [1 , false ]14 # show selected_文字 /1.
```

该编码通过使用 #min 和 #max 聚合（第 8 行和第 9 行）对树的极值行为进行建模，并使用 #sum 聚合（第 10 行和第 11 行）处理森林级决策，从而在悲观聚合下保留决策。示例参见附录示例 6。

4 实验

为评估我们的方法，我们在来自 OpenML、UCI 和 Kaggle 的 20 个公开数据集上进行了实验。每个数据集使用 100 个测试实例，这些实例从 20% 的测试集划分中抽取，以确保评估能够反映模型在未见数据上的泛化性能。我们为每个实例设置了 100 秒的超时时间、6 的最大深度，以及集成模型的 100 个估计器。

我们将基于 ASP 的方法与用于形式化解释的 PyXAI [2], 基于 SAT 的框架进行比较。LIME [21], SHAP [18] 和 Anchors [22] 未纳入比较，因为它们提供的是启发式近似，无法提供形式化保证 [13]。实验在配备 Intel Core i9-9900K 和 64 GB 内存的 Ubuntu 22.04 系统上进行。

表5展示了在超时时间内计算解释的成功率，并区分计算全部解释与计算一个解释的情况。对于计算一个解释的情况，只要至少生成了一个解释，就视为计算成功；对于计算全部解释的情况，则必须枚举出所有解释。我们提出的 ASP 方法在决策树上取得了完美的成功率，而在计算单个解释时，在随机森林上也具有竞争力。然而，在为随机森林枚举解释时，ASP 表现不佳，尤其是在枚举对比解释和多数解释时。相比之下，在为随机森林枚举解释时，PyXAI 在枚举对比解释方面表现更好，而 ASP 在枚举多数解释方面表现更好。

Type	解释数量	模型	PyXAI 完成率 (%)	ASP 完成率 (%)
对比	All	DT	100.0	100.0
对比	All	RF	42.6	6.4
对比	One	DT	100.0	100.0
对比	One	RF	60.8	100.0
多数	All	RF	0.4	6.4
多数	One	RF	100.0	98.7
充分	All	DT	100.0	100.0
充分	One	DT	100.0	100.0
充分	One	RF	100.0	57.7
树特定	One	XGB	100.0	20.8

表 5：跨解释类型和模型计算解释的成功率。

表6比较了随机森林模型中解释的平均数量和长度，仅展示已完成实例的结果。请注意，平均值仅根据成功处理的实例计算，而 PyXAI 和 ASP方法可能分别在不同的子集上成功。

在执行时间未超过超时限制的实例子集上，ASP 能够枚举出比 PyXAI 更多的对比解释和多数解释（#N 列）。这一趋势在多数解释的情况下尤为明显：尽管总体完成率较低，ASP 仍在多个数据集（例如 startup、泰坦尼克号、乳房）上显著优于 PyXAI。然而，对于复杂模型以及资源受限条件下的穷举枚举，ASP 面临可扩展性挑战。

5 相关工作

解释基于决策树的模型这一问题受到了广泛关注，尤其是通过逻辑和符号方法进行解释。Audemard 等人 [2] 提出了一个基于 SAT 的框架，通过将模型转换为合取范式，从决策树生成形式化解释，支持直接解释、充分解释和对比解释，这些解释分别对应于本文所采用的解释概念。

Izza 等人 [15]通过将溯因解释提取为文字子集，解决了基于路径的解释中的冗余问题，相较于路径追踪，提供了更为简洁的解释。

一些基于 SAT 的扩展方法以集成模型为目标。Audemard 等人扩展了其框架，用于计算随机森林的多数解释 [3] 和对比解释 [5]，证明了其可行性，同时也指出了解释枚举所面临的计算挑战。

Izza 等人 [16] 提出了一种高效的基于 SAT 的方法，用于为随机森林生成主蕴含项（PI）解释。其结果表明，与 Anchors[22], 等启发式方法相比，该方法具有更好的性能，凸显了逻辑方法在生成忠实解释方面的优势。

Takemura 等人 [24]提出了一种基于 ASP 的方法，用于从决策树模型中推导基于规则的解释。他们的方法着重于通过挖掘决策树和集成树的结构中的模式，提取全局和局部规则。然而，他们的解释缺少我们的方法所提供的逻辑保证。

除了树模型之外，Dhurandhar 等人 [8]为深度神经网络开发了对比解释方法（CEM），引入了对比范式，并强调通过信号的存在或缺失，提供符合人类直觉的解释。

与这些研究不同，我们基于 ASP 的方法提供了一个统一且可扩展的框架，可为决策树模型和集成生成多种类型的形式化解释。通过利用

	数据集	Con #N		Maj #N		Con #L		Maj #L	
		PyXAI	ASP	PyXAI	ASP	PyXAI	ASP	PyXAI	ASP
1	广告数据	19.7	-	-	133277.5	1.7	-	-	181.1
2	成年人	11.8	-	-	72112.0	2.2	-	-	77.3
3	Arcene	179.0	179.0	-	-	1.0	1.0	-	-
4	澳大利亚	7.9	-	-	-	1.8	-	-	-
5	纸币	8.0	31.0	-	-	1.7	1.0	-	-
6	乳房	8.3	10.0	27.7	48202.8	2.7	1.0	33.8	30.6
7	Christine	2.7	157.0	-	-	2.4	1.0	-	-
8	compas	3.9	3.8	-	22085.6	1.5	1.5	-	13.9
9	德国信用	7.5	-	-	1392.0	2.6	-	-	82.6
10	德克斯特	50.7	172.8	-	-	2.2	1.0	-	-
11	糖尿病	11.7	-	-	-	1.7	-	-	-
12	离婚	20.6	-	-	-	4.4	-	-	-
13	吉娜不可知论者	7.2	-	-	-	2.3	-	-	-
14	心脏	6.7	-	-	5768.0	2.0	-	-	81.1
15	印度肝病患者	24.5	-	-	-	1.5	-	-	-
16	qsar	9.9	-	-	-	2.0	-	-	-
17	垃圾邮件数据集	1.7	-	-	-	1.2	-	-	-
18	创业公司	14.0	476.1	-	-	2.1	1.0	-	-
19	泰坦尼克号	13.9	208.0	-	127778.0	2.5	1.0	-	212.9
20	wine	6.0	-	-	-	1.0	-	-	-

表6：随机森林结果：对比解释和多数解释的平均数量（#N）及长度（#L）。破折号（-）表示超时（100秒）。仅统计已完成的实例（全部解释）。

ASP 的模型枚举能力，我们能够在单一编码环境中支持各种解释类型，从而提高灵活性，并简化与领域特定约束的集成。

6 结论

本文提出了一种基于 ASP 的方法，用于生成决策树模型的形式化解释。我们实现了充分解释、对比解释、多数解释和树特定解释的编码，并证明 ASP 能够复现基于 SAT 的方法，有时甚至优于这些方法；尤其是在随机森林中生成单个多数解释和对比解释时。

ASP 的枚举能力在生成多个解释时具有实际优势，可通过一次求解器调用探索搜索空间。不过，对于大型模型，可扩展性方面仍面临挑战，计算时间过长限制了其实用性。主要瓶颈包括：（1）实例化所有可能组合时，实例化过程中的内存需求呈指数级增长；（2）低效的两阶段流程需要使用两个不同的 ASP 系统。

未来的工作将着重通过更高效的编码、剪枝、更好的启发式方法和优化来提升运行性能，同时拓展至提升树解释，并开展更广泛的比较分析。

致谢

本研究得到了日本 JSPS KAKENHI Grant Numbers JP21H04905、JP25K03190 以及 JST CREST Grant Number JPMJCR22D3 的资助。

参考文献

[1] Mario Alviano、Carmine Dodaro、Wolfgang Faber、Nicola Leone 和 Francesco Ricca（2013）：WASP：一种基于约束学习的原生 ASP 求解器。载于：LPNMR，第 54–66 页，doi:10.1007/978-3-642-40564-8 6。[2] Gilles Audemard、Steve Bellart、Louenas Bounia、Frederic Koriche、Jean-Marie Lagniez 和 Pierre Marquis（2022）：布尔决策树的解释能力。数据与知识工程 142，p. 102088，doi:10.1016/j.datak.2022.102088。[3] Gilles Audemard、Steve Bellart、Louenas Bounia、Frederic Koriche、Jean-Marie Lagniez 和 Pierre Marquis（2022）：以复杂度换取随机森林解释的稀疏性。载于：AAAI，36，第 5461–5469 页，doi:10.1609/aaai.v36i5.20484。[4] Gilles Audemard、Jean-Marie Lagniez、Pierre Marquis 和 Nicolas Szczepanski（2023）：计算提升树的溯因解释。载于：AISTATS，第 4699–4711 页。[5] Gilles Audemard、Jean-Marie Lagniez、Pierre Marquis 和 Nicolas Szczepanski（2023）：基于树的分类器的对比解释。载于：ECAI，《人工智能与应用前沿》372，第 117–124 页，doi:10.3233/faia230261。[6] Leo Breiman（2001）：随机森林。机器学习 45(1)，第 5–32 页，doi:10.1023/A:1010933404324。[7] Tianqi Chen 和 Carlos Guestrin（2016）：XGBoost：一种可扩展的树提升系统。载于：KDD，第 785–794 页，doi:10.1145/2939672.2939785。[8] Amit Dhurandhar、Pin-Yu Chen、Ronny Luss、Chun-Chen Tu、Pai-Shun Ting、Karthikeyan Shanmugam 和 Payel Das（2018）：基于缺失特征的解释：通过相关负例实现对比解释。载于：NeurIPS，第 590–601 页。[9] Jerome H. Friedman（2001）：贪心函数逼近：一种梯度提升机。《统计年鉴》，第 1189–1232 页，doi:10.1214/aos/1013203451。[10] Martin Gebser、Roland Kaminski、Benjamin Kaufmann 和 Torsten Schaub（2014）：Clingo = ASP + Control：初步报告。CoRRabs/1405.3694。[11] Michael Gelfond 和 Vladimir Lifschitz（1988）：逻辑编程的稳定模型语义。载于：ICLP/SLP，88，第 1070–1080 页。[12] Riccardo Guidotti、Anna Monreale、Salvatore Ruggieri、Franco Turini、Fosca Giannotti 和 Dino Pedreschi（2018）：黑盒模型解释方法综述。《ACM 计算调查》51(5)，p. Article 93，doi:10.1145/3236009。[13] Alexey Ignatiev（2020）：迈向可信赖的可解释人工智能。载于：IJCAI，5，第 5154–5158 页，doi:10.24963/ijcai.2020/726。[14] Alexey Ignatiev、Nina Narodytska 和 Jo˜ao Marques-Silva（2019）：机器学习模型的基于溯因的解释。载于：AAAI，第 1511–1519 页，doi:10.1609/aaai.v33i01.33011511。[15] Yacine Izza、Alexey Ignatiev 和 Jo˜ao Marques-Silva（2020）：决策树解释。arXiv:2010. 11034。[16] Yacine Izza 和 Jo˜ao Marques-Silva（2021）：使用 SAT 解释随机森林。载于：IJCAI，第 2584–2591 页，doi:10.24963/ijcai.2021/356。[17] Guolin Ke、Qi Meng、Thomas Finley、Taifeng Wang、Wei Chen、Weidong Ma、Qiwei Ye 和 Tie-Yan Liu（2017）：LightGBM：一种高效的梯度提升决策树。载于：NeurIPS，第 3146–3154 页。[18] Scott M Lundberg 和 Su-In Lee（2017）：解释模型预测的统一方法。载于：NeurIPS，第 4765–4774 页。[19] Jo˜ao Marques-Silva 和 Alexey Ignatiev（2022）：通过形式化可解释人工智能打造可信赖的人工智能。载于：AAAI，第 12342–12350 页，doi:10.1609/aaai.v36i11.21499。[20] Nina Narodytska、Aditya Shrotri、Kuldeep S. Meel、Alexey Ignatiev 和 Jo˜ao Marques-Silva（2019）：使用模型计数评估启发式机器学习解释。载于：SAT，第 267–278 页，doi:10.1007/ 978-3-030-24258-9 19。

[21] Marco Tulio Ribeiro、Sameer Singh 和 Carlos Guestrin（2016年）：《我为什么应该相信你？》：解释任意分类器的预测。载于：KDD，第 1135–1144 页，doi:10.1145/2939672.2939778。[22] Marco Tulio Ribeiro、Sameer Singh 和 Carlos Guestrin（2018年）：Anchors：高精度、模型无关的解释。载于：AAAI，第 1527–1535 页，doi:10.1609/aaai.v32i1.11491。[23] Cynthia Rudin、Chaofan Chen、Zhi Chen、Haiyang Huang、Lesia Semenova 和 Chudi Zhong（2022年）：可解释机器学习：基本原则与 10 项重大挑战。Statistics Surveys16，第 1–85 页，doi:10.1214/21-ss133。[24] Akihiro Takemura 和 Katsumi Inoue（2024年）：通过答案集编程为树集成学习方法生成全局和局部解释。逻辑编程理论与实践 24(5)，第 973–1010 页，doi:10.1017/s1471068424000401。

A 解释示例

本附录提供了可视化示例，说明如何从决策树模型中推导出充分解释、对比解释、多数解释和树特定解释。每幅图都附有简要说明，介绍模型结构和文字所代表的含义。

A.1 决策树

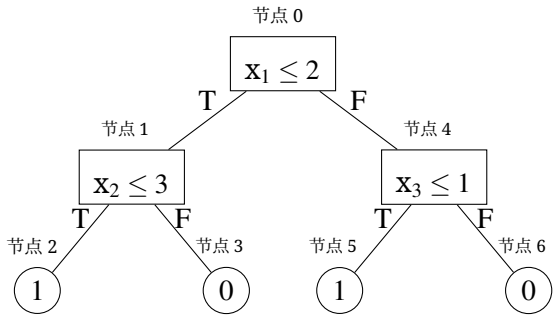

图 2：一棵包含三个内部节点的决策树。

设输入实例为 $xxx = \{x_1 = 1, x_2 = 1, x_3 = 2\}$。此实例满足 $(x_1 \leq 2)$ 和 $(x_2 \leq 3)$，因此树沿左-左路径前进，并预测为 1。该树被编码为一组 ASP 事实，其中我们使用从 0 开始的深度优先索引。例如，node(0,1,0) 对应于满足 $(x_1 \leq 2)$，node(1,1,1) 对应于满足 $(x_2 \leq 3)$，等等。

程序 8：决策树：实例

```
%% node status
node (0 ,0 ,1) .    节点 (1 ,1 ,1) .    节点 (4 ,1 ,0) .

%% 树结构
leaf_节点 (2 ,1) .   leaf_节点 (3 ,0) .   leaf_节点 (5 ,1) .   leaf_节点（6，0）。
左侧_节点（0，1）。 右侧_节点（0，4）。  左侧_节点（1，2）。 右侧_节点（1，3）。
左侧_节点（4，5）。 右侧_节点（4，6）。
```

示例 1（充分解释）使用此实例执行程序 1 后，会得到包含 *selected_literal(2)* 的答案集，它是充分解释的补集。因此，此案例的充分解释为：*selected_literal(0)*和 *selected_literal(1)*。这些对应于文字 $\{x_1 \leq 2、x_2 \leq 3\}$，改变任一文字都会使预测翻转，因此该项是充分解释。

示例 2（对比解释）执行程序 2 得到 *selected_literal(1)*，对应于 $\{x_2 \leq 3\}$。翻转此条件，即 $x_2 > 3$，会导致输出从 1 变为 0：这就是一种对比解释。

A.2 随机森林

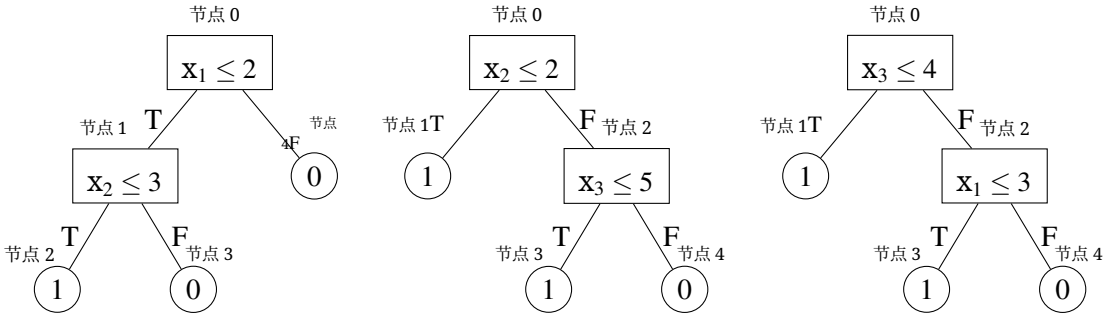

图3：一片包含三棵树的随机森林。

考虑实例 xxx $= \{x_1 = 1、x_2 = 2、x_3 = 6\}$，三个树对该实例的预测类别均为类别1。该随机森林和该实例产生了以下编码。

程序 9：随机森林：实例

```
%% 实例和节点状态
特征 (1 ,1) 。
特征 (2 ,1) 。
特征 (3 ,1) 。
特征 (4 ,0) 。
特征 (5 ,0) 。
特征 (6 ,1) 。

%% 森林预测和阈值
pre_森林 (1) 。        con_树_阈值 (1) 。              majo_树_阈值 (2) 。

%% 树 1
节点 (1  ,0 ,1 ,1) .  节点 (1  ,1 ,2 ,1) .
leaf_节点 (1 ,2 ,1) .    leaf_节点 (1 ,3 ,0) .    leaf_节点 (1 ,4 ,0) .
左侧_节点 (1 ,0 ,1) .    右侧_节点 (1 ,0 ,4) .    左侧_节点 (1 ,1 ,2) .    右侧_节点 (1 ,1 ,3) .

%% 树 2
节点 (2 ,0 ,3 ,1) .    节点 (2 ,2 ,4 ,0) 。
leaf_节点 (2 ,1 ,1) 。  leaf_节点 (2 ,3 ,1) 。  leaf_节点 (2 ,4 ,0) 。
左_节点 (2 ,0 ,1) 。    右_节点 (2 ,0 ,2) 。    左侧_节点 (2 ,2 ,3) 。    右侧_节点 (2 ,2 ,4) 。

%% 树 3
节点 (3 ,0 ,5 ,0) 。    节点 (3 ,2 ,6 ,1) 。
叶节点_节点 (3 ,1 ,1) 。  叶节点_节点 (3 ,3 ,1) .  叶节点_节点 (3 ,4 ,0) .
左侧_节点 (3 ,0 ,1) .    右侧_节点 (3 ,0 ,2) .    左侧_节点 (3 ,2 ,3) .    右侧_节点 (3 ,2 ,4) .
```

程序 10：随机森林充分解释：编码

```

%% 将此行添加到程序 3 和 9 前面,                    并将其保存为 enc . lp
%% MUS 提取生成器。                          请注意，1..6 取决于森林。
{ 变化(L)      : 非   fix_lit (L) ,L =1..6}。

%% 程序 3
节点 (T ,X ,L)   :-  节点 (T ,X ,L  , _) .
```

```
8 new_node (T ,X ,L ,0) :- change (L) , node (T ,X,L) ,feature (L ,1) . 9 new_
node (T ,X ,L ,1) :- change (L) , node (T ,X,L) ,feature (L ,0) . 10 ... 11 12 %% 程序
9 13 %% 实例和节点状态 14 feature (1 ,1) . 15 feature (2 ,1) . 16 ...
```

程序 11：随机森林充分解释：实例

```
%% 使用 WASP 进行最小不可满足子集提取， 将其保存为 ins .lp
fix_lit (1) 。
fix_字面量（2）。
fix_字面量（3）。
fix_字面量（4）。
fix_字面量（5）。
fix_字面量（6）。
```

程序12：随机森林充分解释：命令

```
python wasp_rewriter . py enc . lp ins .lp | clingo -- 输出 =smodels | ./ wasp -- mus =__调试__ -n0
```

示例3（充分解释）使用 WASP 执行程序12中的命令会输出*fix_lit(3)*, *fix_lit(6)*，这对应于 $\{x_2 \leq 2, x_1 \leq 3\}$。固定这些文字可确保足够多的树保留原始预测。

示例4（对比解释）执行程序 5 会输出
selected_literal(1), *selected_literal(3)*，这对应于 $\{x_1 \leq 2, x_2 \leq 2\}$。如果翻转这两个文字，多数树的预测会发生变化，森林预测也会改变。

示例5（多数解释）执行程序 6 输出*selected_literal(3)*, *selected_literal(4)*, *selected_literal(5)*，从文字集合中减去这些输出后，得到*selected_literal(1)*, *selected_literal(2)*, *selected_literal(6)*。它们对应于$\{x_1 \leq 2$、$x_2 \leq 3$、$x_1 \leq 3\}$，确保至少有两棵树维持该预测。

A.3 提升树

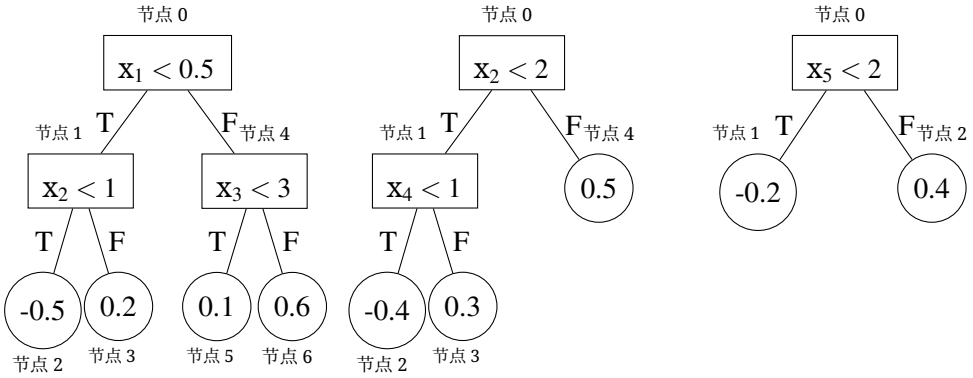

图4：包含三棵回归树的提升树模型。

考虑输入实例 xxx $= \{x_1 = 1、x_2 = 2、x_3 = 3、x_4 = 4、x_5 = 5\}$。提升树被编码为以下 ASP 事实：

程序13：提升树：实例

```
pre_森林（1）。
%% 树 1
节点（0，0，1，0）。节点（0，1，2，1）。节点（0，4，3，0）。
叶节点_节点 (0 ,2 , -500) .    叶节点_节点 (0 ,3 ,200) .    叶节点_节点 (0 ,5 ,100) .    叶节点_节点 (0 ,6 ,600) .
左侧_节点 (0 ,0 ,1) .          右侧_节点 (0 ,0 ,4) .
左侧_节点 (0 ,1 ,2) .          右侧_节点 (0 ,1 ,3) .
左侧_节点 (0 ,4 ,5) .          右侧_节点 (0 ,4 ,6) .

%% 树 2
节点 (1 ,0 ,4 ,0) .      节点 (1 ,1 ,5 ,0) .
叶节点_节点 (1 ,2 , -400) .    叶节点_节点 (1 ,3 ,300) .    叶节点_节点 (1 ,4 ,500) .
左_节点 (1 ,0 ,1) .            右_节点 (1 ,0 ,4) .
左_节点 (1 ,1 ,2) 。           右_节点 (1 ,1 ,3) 。

%% 树 3
节点 (2 ,0 ,6 ,0) 。    叶节点_节点 (2 ,1 , -200) 。    叶节点_节点 (2 ,2 ,400) 。
左侧_节点 (2 ,0 ,1) 。          右侧_节点 (2 ,0 ,2) 。
```

示例 6（树特定解释）执行程序 7 输出 *selected_literal(5),selected_literal(6)*，对应于 $\{x_4 < 1, x_5 < 2\}$。固定这些文字可确保最坏情况实例的总和仍大于 0。