Feature Necessity & Relevancy in ML Classifier Explanations

Xuanxiang Huang[1], Martin C. Cooper[2], Antonio Morgado[3], Jordi Planes[4], and Joao Marques-Silva[5](✉)

[1] University of Toulouse, Toulouse, France xuanxiang.huang@univ-toulouse.fr
[2] Univ. Paul Sabatier, IRIT, Toulouse, France martin.cooper@irit.fr
[3] Universitat de Lleida, Lleida, Spain antonio.morgado@udl.cat
[4] Universitat de Lleida, Lleida, Spain jordi.planes@udl.cat
[5] IRIT, CNRS, Toulouse, France joao.marques-silva@irit.fr

Abstract. Given a machine learning (ML) model and a prediction, explanations can be defined as sets of features which are sufficient for the prediction. In some applications, and besides asking for an explanation, it is also critical to understand whether sensitive features can occur in some explanation, or whether a non-interesting feature must occur in all explanations. This paper starts by relating such queries respectively with the problems of relevancy and necessity in logic-based abduction. The paper then proves membership and hardness results for several families of ML classifiers. Afterwards the paper proposes concrete algorithms for two classes of classifiers. The experimental results confirm the scalability of the proposed algorithms.

Keywords: Formal Explainability · Abduction · Abstraction Refinement.

1 Introduction

The remarkable achievements in machine learning (ML) in recent years [12,32,47] are not matched by a comparable degree of trust. The most promising ML models are inscrutable in their operation. As a direct consequence, the opacity of ML models raises distrust in their use and deployment. Motivated by a critical need for helping human decision makers to grasp the decisions made by ML models, there has been extensive work on explainable AI (XAI). Well-known examples include so-called model agnostic explainers or alternatives based on saliency maps for neural networks [9,50,58,59]. While most XAI approaches do not offer guarantees of rigor, and so can produce explanations that are unsound given the underlying ML model, there have been efforts on developing rigorous XAI approaches over the last few years [40,54,63]. Rigorous explainability involves the computation of explanations, but also the ability to answer a wide range of related queries [7,8,36].

By building on the relationship between explainability and logic-based abduction [25,30,40,61], this paper analyzes two concrete queries, namely feature necessity and relevancy. Given an ML classifier, an instance (i.e. point in feature space and associated prediction) and a target feature, the goal of feature necessity is to decide whether the target feature occurs in *all* explanations of the given instance. Under the same assumptions, the goal of feature relevancy is to decide whether a feature occurs in *some* explanation of the given instance. This paper proves a number of complexity results regarding feature necessity and relevancy, focusing on well-known families of classifiers, some of which are widely used in ML. Moreover, the paper proposes novel algorithms for deciding relevancy for two families of classifiers. The experimental results demonstrate the scalability of the proposed algorithms.

The paper is organized as follows. The notation and definitions used throughout are presented in Section 2. The problems of feature necessity and relevancy are studied in Section 3, and example algorithms are proposed in Section 4. Section 5 presents experimental results for a sample of families of classifiers, Section 6 relates our contribution with earlier work and Section 7 concludes the paper.

2 Preliminaries

Complexity classes, propositional logic & quantification. The paper assumes basic knowledge of computational complexity, namely the classes of decision problems P, NP and Σ_2^P [6]. The paper also assumes basic knowledge of propositional logic, including the Boolean satisfiability (SAT) problem for propositional logic formulas in conjunctive normal form (CNF), and the use of SAT solvers as oracles for the complexity class NP. The interested reader is referred to textbooks on these topics [6, 13].

2.1 Classification Problems

Throughout the paper, we will consider classifiers as the underlying ML model. Classification problems are defined on a set of features (or attributes) $\mathcal{F} = \{1, \ldots, m\}$ and a set of classes $\mathcal{K} = \{c_1, c_2, \ldots, c_K\}$. Each feature $i \in \mathcal{F}$ takes values from a domain $\mathbb{D}_i$. Domains are categorical or ordinal, and each domain can be defined on boolean, integer/discrete or real values. Feature space is defined as $\mathbb{F} = \mathbb{D}_1 \times \mathbb{D}_2 \times \ldots \times \mathbb{D}_m$. The notation $\mathbf{x} = (x_1, \ldots, x_m)$ denotes an arbitrary point in feature space, where each x_i is a variable taking values from $\mathbb{D}_i$. The set of variables associated with the features is $X = \{x_1, \ldots, x_m\}$. Also the notation $\mathbf{v} = (v_1, \ldots, v_m)$ represents a specific point in feature space, where each v_i is a constant representing one concrete value from $\mathbb{D}_i$. A classifier $\mathbb{C}$ is characterized by a (non-constant) *classification function* κ that maps feature space $\mathbb{F}$ into the set of classes $\mathcal{K}$, i.e. $\kappa : \mathbb{F} \to \mathcal{K}$. An *instance* denotes a pair $(\mathbf{v}, c)$, where $\mathbf{v} \in \mathbb{F}$ and $c \in \mathcal{K}$, with $c = \kappa(\mathbf{v})$.

2.2 Examples of Classifiers

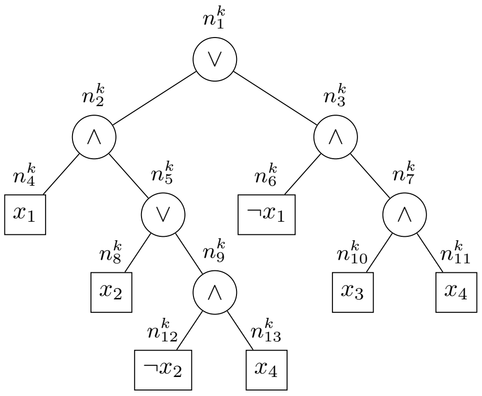
The results presented in the paper apply to a comprehensive range of widely used classifiers [28, 62]. These include, decision trees (DTs) [18, 42], decision graphs (DGs) [44] and diagrams (DDs) [1, 68], decision lists (DLs) [38, 60] and sets (DSs) [19, 41], tree ensembles (TEs) [37], including random forests (RFs) [17, 43] and boosted trees (BTs) [29], neural networks (NNs) [56], naive bayes classifiers (NBCs) [45, 52], classifiers represented with propositional languages, including deterministic decomposable negation normal form (d-DNNFs) [23, 35] and its proper subsets, e.g. sentential decision diagrams (SDDs) [22, 66] and free binary decision diagrams (FBDDs) [23, 31, 68], and also monotonic classifiers. In the rest of the paper, we will analyze some families of classifiers in more detail.

d-DNNF classifiers. Negation normal form (NNF) is a well-known propositional language, where the negation operators are restricted to atoms, or inputs. Any propositional formula can de reduced to NNF in polynomial time. Let the *support* of a node be the set of atoms associated with leaves reachable from the outgoing edges of the node. Decomposable NNF (DNNF) is a restriction of NNF where the children of AND nodes do not share atoms in their support. A DNNF circuit is *deterministic* (referred to as d-DNNF) if any two children of OR nodes cannot both take value 1 for any assignment to the inputs. Restrictions of NNF including DNNF and d-DNNF exhibit important tractability properties [23]. Besides, we briefly introduce FBDDs which is a proper subset of d-DNNFs. An FBDD over a set X of Boolean variables is a rooted, directed acyclic graph comprising two types of nodes: *nonterminal* and *terminal*. A nonterminal node is labeled by a variable $x_i \in X$, and has two outgoing edges, one labeled by 0 and the other by 1. A terminal node is labeled by a 1 or 0, and has no outgoing edges. For a subgraph rooted at a node labeled with a variable x_i, it represents a boolean function f which is defined by the *Shannon expansion*: $f = (x_i \wedge f|_{x_i=1}) \vee (\neg x_i \wedge f|_{x_i=0})$, where $f|_{x_i=1}$ ($f|_{x_i=0}$) denotes the *cofactor* [16] of f with respect to $x_i = 1$ ($x_i = 0$). Moreover, any FBDD is *read-once*, meaning that each variable is tested at most once on any path from the root node to a terminal node.

Monotonic classifiers. Monotonic classifiers find a number of important applications, and have been studied extensively in recent years [26, 48, 65, 70]. Let $\preccurlyeq$ denote a partial order on the set of classes $\mathcal{K}$. For example, we assume $c_1 \preccurlyeq c_2 \preccurlyeq \ldots c_K$. Furthermore, we assume that each domain D_i is ordered such that the value taken by feature i is between a lower bound $\lambda(i)$ and an upper bound $\mu(i)$. Given $\mathbf{v}_1 = (v_{11}, \ldots, v_{1i}, \ldots, v_{1m})$ and $\mathbf{v}_2 = (v_{21}, \ldots, v_{2i}, \ldots, v_{2m})$, we say that $\mathbf{v}_1 \leq \mathbf{v}_2$ if $\forall(i \in \mathcal{F}).(v_{1i} \leq v_{2i})$. Finally, a classifier is monotonic if whenever $\mathbf{v}_1 \leq \mathbf{v}_2$, then $\kappa(\mathbf{v}_1) \preccurlyeq \kappa(\mathbf{v}_2)$.

Running examples. As hinted above, throughout the paper, we will consider two fairly different families of classifiers, namely classifiers represented with d-DNNFs and monotonic classifiers.

Example 1. The first example is the d-DNNF classifier $\mathbb{C}_1$ shown in Fig. 1. It represents the boolean function $(x_1 \wedge (x_2 \vee x_4)) \vee (\neg x_1 \wedge x_3 \wedge x_4)$. The instance considered throughout the paper is $(\mathbf{v}_1, c_1) = ((0, 1, 0, 0), 0)$.



(a) Graphical representation of d-DDNF, i.e. κ_1

$$\mathcal{F}_1 = \{1, 2, 3, 4\}$$
$$\mathbb{D}_{1i} = \{0, 1\}, i = 1, \ldots, 4$$
$$\mathcal{K}_1 = \{0, 1\}$$

(b) Definition of $\mathcal{F}_1, \mathbb{D}_{1i}, \mathcal{K}_1$

```
IF      x_1 = 1 ∧ x_2 = 1 THEN 1
ELSE IF x_1 = 1 ∧ x_4 = 1 THEN 1
ELSE IF x_3 = 1 ∧ x_4 = 1 THEN 1
ELSE                           0
```

(c) Alternative representation of κ_1

Fig. 1: Example of d-DDNF classifier

$$\mathcal{F}_2 = \{1, 2, 3, 4\}$$
$$\mathbb{D}_{2i} = \{0, 1\}, i = 1, \ldots, 4$$
$$\mathcal{K}_2 = \{0, 1\}$$

(a) Definition of $\mathcal{F}_2, \mathbb{D}_{2i}, \mathcal{K}_2$

$$\kappa_2(\mathbf{x}) = \begin{cases} 1 & \text{if } x_1 + x_2 + x_3 \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

(b) Definition of κ_2

Fig. 2: Example of a monotonic classifier

Example 2. The second running example is the monotonic classifier $\mathbb{C}_2$ shown in Fig. 2. The instance that is considered throughout the paper is $(\mathbf{v}_2, c_2) = ((1, 1, 1, 1), 1)$.

2.3 Formal Explainability

Prime implicant (PI) explanations [63] represent a minimal set of literals (relating a feature value x_i and a constant $v_i \in \mathbb{D}_i$) that are logically sufficient for the prediction. PI-explanations are related with logic-based abduction, and so are also referred to as abductive explanations (AXp's) [54]. AXp's offer guarantees of rigor that are not offered by other alternative explanation approaches. More recently, AXp's have been studied in terms of their computational complexity [7, 10]. There is a growing body of recent work on formal explanations [3–5, 14, 15, 24, 27, 33, 51, 54, 67].

Formally, given $\mathbf{v} = (v_1, \ldots, v_m) \in \mathbb{F}$, with $\kappa(\mathbf{v}) = c$, an AXp is any subset-minimal set $\mathcal{X} \subseteq \mathcal{F}$ such that,

$$\mathsf{WAXp}(\mathcal{X}) \quad := \quad \forall(\mathbf{x} \in \mathbb{F}). \left[\bigwedge\nolimits_{i\in\mathcal{X}}(x_i = v_i)\right] \rightarrow (\kappa(\mathbf{x}) = c) \tag{1}$$

If a set $\mathcal{X} \subseteq \mathcal{F}$ is not minimal but (1) holds, then $\mathcal{X}$ is referred to as a *weak* AXp. Clearly, the predicate WAXp maps $2^{\mathcal{F}}$ into $\{\bot, \top\}$ (or $\{\textbf{false}, \textbf{true}\}$). Given $\mathbf{v} \in \mathbb{F}$, an AXp $\mathcal{X}$ represents an irreducible (or minimal) subset of the features which, if assigned the values dictated by $\mathbf{v}$, are sufficient for the prediction c, i.e. value changes to the features not in $\mathcal{X}$ will not change the prediction. We can use the definition of the predicate WAXp to formalize the definition of the predicate AXp, also defined on subsets $\mathcal{X}$ of $\mathcal{F}$:

$$\mathsf{AXp}(\mathcal{X}) \quad := \quad \mathsf{WAXp}(\mathcal{X}) \land \forall(\mathcal{X}' \subsetneq \mathcal{X}).\neg\mathsf{WAXp}(\mathcal{X}') \tag{2}$$

The definition of $\mathsf{WAXp}(\mathcal{X})$ ensures that the predicate is *monotone*. Indeed, if $\mathcal{X} \subseteq \mathcal{X}' \subseteq \mathcal{F}$, and if $\mathcal{X}$ is a weak AXp, then $\mathcal{X}'$ is also a weak AXp, as the fixing of more features will not change the prediction. Given the monotonicity of predicate WAXp, the definition of predicate AXp can be simplified as follows, with $\mathcal{X} \subseteq \mathcal{F}$:

$$\mathsf{AXp}(\mathcal{X}) := \mathsf{WAXp}(\mathcal{X}) \land \forall(j \in \mathcal{X}).\neg\mathsf{WAXp}(\mathcal{X} \setminus \{j\}) \tag{3}$$

This simpler but equivalent definition of AXp has important practical significance, in that only a linear number of subsets needs to be checked for, as opposed to exponentially many subsets in (2). As a result, the algorithms that compute one AXp are based on (3) [54].

Example 3. From Example 1, and given the instance $((0,1,0,0),0)$, we can conclude that the prediction will be 0 if features 1 and 3 take value 0, or if features 1 and 4 take value 0. Hence, the AXp's are $\{1,3\}$ and $\{1,4\}$. It is also apparent that the assignment $x_2 = 1$ bears no relevance on the fact that the prediction is 0.

Example 4. From Example 2, we can conclude that any sum of two variables assigned value 1 suffices for the prediction. Hence, given the instance $((1,1,1,1),1)$, the possible AXp's are $\{1,2\}$, $\{1,3\}$, and $\{2,3\}$. Observe that the definition of κ_2 does not depend on feature 4.

Besides abductive explanations, another commonly studied type of explanations are contrastive or counterfactual explanations [8,36,39,55]. As argued in related work [36], the duality between abductive and contrastive explanations implies that for the purpose of the queries studied in this paper, it suffices to study solely abductive explanations.

3 Feature Relevancy & Necessity: Theory

This section investigates the complexity of feature relevancy and necessity[6]. We are interested in membership results, which allow us to devise algorithms for the target problems. We are also interested in hardness results, which serve to confirm that the running time complexities of the proposed algorithms are within reason, given the problem's complexity.

3.1 Defining Necessity, Relevancy & Irrelevancy

Throughout this section, a classifier $\mathbb{C}$ is assumed, with features $\mathcal{F}$, domains $\mathbb{D}_i$, $i \in \mathcal{F}$, classes $\mathcal{K}$, a classification function $\kappa : \mathbb{F} \to \mathcal{K}$, and a concrete instance $(\mathbf{v}, c)$, $\mathbf{v} \in \mathbb{F}, c \in \mathcal{K}$.

[6] For the sake of brevity, we opt to only present sketches of some of the proofs.

Definition 1 (Feature Necessity, Relevancy & Irrelevancy)**.** Let $\mathbb{A}$ denote the set of all AXp's for a classifier given a concrete instance, i.e. $\mathbb{A} = \{\mathcal{X} \subseteq \mathcal{F} \,|\, \mathsf{AXp}(\mathcal{X})\}$, and let $t \in \mathcal{F}$ be a target feature. Then, (i) t is necessary if $t \in \cap_{\mathcal{X}\in\mathbb{A}}\mathcal{X}$; (ii) t is relevant if $t \in \cup_{\mathcal{X}\in\mathbb{A}}\mathcal{X}$; and (iii) t is irrelevant if $t \in \mathcal{F} \setminus \cup_{\mathcal{X}\in\mathbb{A}}\mathcal{X}$.

Throughout the remainder of the paper, the problem of deciding feature necessity is represented by the acronym FNP, and the problem of deciding feature relevancy is represented by the acronym FRP.

Example 5. As shown earlier, for the d-DNNF classifier of Fig. 1, and given the instance $(\mathbf{v}_1, c_1) = ((0,1,0,0),0)$, there exist two AXp's, i.e. $\{1,3\}$ and $\{1,4\}$. Clearly, feature 1 is necessary, and features 1, 3 and 4 are relevant. In contrast, feature 2 is irrelevant.

Example 6. For the monotonic classifier of Fig. 2, and given the instance $(\mathbf{v}_2, c_2) = ((1,1,1,1),1)$, we have argued earlier that there exist three AXp's, i.e. $\{1,2\}$, $\{1,3\}$ and $\{2,3\}$, which allows us to conclude that features 1, 2 and 3 are relevant, but that feature 4 is irrelevant. In this case, there are no necessary features.

The general complexity of necessity and (ir)relevancy has been studied in the context of logic-based abduction [25, 30, 61]. Recent uses in explainability are briefly overviewed in Section 6.

3.2 Feature Necessity

Proposition 2. If deciding $\mathsf{WAXp}(\mathcal{X})$ is in complexity class $\mathfrak{C}$, then FNP is in the complexity class co-$\mathfrak{C}$.

Given the known polynomial complexity of deciding whether a set is a weak AXp for several families of classifiers [54], we then have the following result:

Corollary 3. For DTs, XpG's[7], NBCs, d-DNNF classifiers and monotonic classifiers, FNP is in P.

3.3 Feature Relevancy: Membership Results

Proposition 4 (Feature Relevancy for DTs [36])**.** FRP for DTs is in P.

Proposition 5. If deciding $\mathsf{WAXp}(\mathcal{X})$ is in P, then FRP is in NP.

The argument above can also be used for proving the following results.

Corollary 6. For XpG's, NBCs, d-DNNF classifiers and monotonic classifiers, FRP is in NP.

Proposition 7. If deciding $\mathsf{WAXp}(\mathcal{X})$ is in NP, then FRP is in Σ_2^{P}.

Corollary 8. For DLs, DSs, RFs, BTs, and NNs, FRP is in Σ_2^{P}.

Additional results. The following result will prove useful in designing algorithms for FRP in practice.

Proposition 9. Let $\mathcal{X} \subseteq \mathcal{F}$, and let $t \in \mathcal{X}$ denote some target feature such that, $\mathsf{WAXp}(\mathcal{X})$ holds and $\mathsf{WAXp}(\mathcal{X} \setminus \{t\})$ does not hold. Then, for any AXp $\mathcal{Z} \subseteq \mathcal{X} \subseteq \mathcal{F}$, it must be the case that $t \in \mathcal{Z}$.

[7] Explanation graphs (XpG's) have been proposed to enable the computation of explanations for decision graphs, and (multi-valued) decision diagrams [36].

3.4 Feature Relevancy: Hardness Results

Proposition 10 (Relevancy for DNF Classifiers [36])**.** Feature relevancy for a DNF classifier is Σ_2^{P}-hard.

Proposition 11. Feature relevancy for monotonic classifiers is NP-hard.

Proof. We say that a CNF is trivially satisfiable if some literal occurs in all clauses. Clearly, SAT restricted to nontrivial CNFs is still NP-complete. Let Φ be a not trivially satisfiable CNF on variables $x_1, \ldots, x_k$. Let $N = 2k$. Let $\tilde{\Phi}$ be identical to Φ except that each occurrence of a negative literal x_i $(1 \leq i \leq k)$ is replaced by x_{i+k}. Thus $\tilde{\Phi}$ is a CNF on N variables each of which occur only positively. Define the boolean classifier κ (on $N+1$ features) by $\kappa(x_0, x_1, \ldots, x_N) = 1$ iff $x_i = x_{i+k} = 1$ for some $i \in \{1, \ldots, k\}$ or $x_0 \wedge \tilde{\Phi}(x_1, \ldots, x_N) = 1$. To show that Φ is monotonic we need to show that $\mathbf{a} \leq \mathbf{b} \Rightarrow \kappa(\mathbf{a}) \leq \kappa(\mathbf{b})$. This follows by examining the two cases in which $\kappa(\mathbf{a}) = 1$: if $a_i = a_{i+k} \wedge \mathbf{a} \leq \mathbf{b}$, then $b_i = b_{i+k}$, whereas, if $a_0 \wedge \tilde{\Phi}(a_1, \ldots, a_N) = 1$ and $\mathbf{a} \leq \mathbf{b}$, then $b_0 \wedge \tilde{\Phi}(b_1, \ldots, b_N) = 1$ (by positivity of $\tilde{\Phi}$), so in both cases $\kappa(\mathbf{b}) = 1 \geq \kappa(\mathbf{a})$.

Clearly $\kappa(\mathbf{1}_{N+1}) = 1$. There are k obvious AXp's of this prediction, namely $\{i, i+k\}$ $(1 \leq i \leq k)$. These are minimal by the assumption that Φ is not trivially satisfiable. This means that no other AXp contains both i and $i+k$ for any $i \in \{1, \ldots, k\}$. Suppose that $\Phi(\mathbf{u}) = 1$. Let $\mathcal{X}_u$ be $\{0\} \cup \{i \mid 1 \leq i \leq k \wedge u_i = 1\} \cup \{i+k \mid 1 \leq i \leq k \wedge u_i = 0\}$. Then $\mathcal{X}_u$ is a weak AXp of the prediction $\kappa(1) = 1$. Furthermore $\mathcal{X}_u$ does not contain any of the AXp's $\{i, i+k\}$. Therefore some subset of $\mathcal{X}$ is an AXp and clearly this subset must contain feature 0. Thus if Φ is satisfiable, then there is an AXp which contains 0.

We now show that the converse also holds. If $\mathcal{X}$ is an AXp of $\kappa(\mathbf{1}_{N+1}) = 1$ containing 0, then it cannot also contain any of the pairs $i, i+k$ $(1 \leq i \leq k)$, otherwise we could delete 0 and still have an AXp. We will show that this implies that we can build a satisfying assignment $\mathbf{u}$ for Φ. Consider first $\mathbf{v} = (v_0, \ldots, v_N)$ defined by $v_i = 1$ if $i \in \mathcal{X}$ $(0 \leq i \leq N)$ and $v_{i+k} = 1$ if neither i nor $i+k$ belongs to $\mathcal{X}$ $(1 \leq i \leq k)$, and $v_i = 0$ otherwise $(1 \leq i \leq N)$. Then $\kappa(\mathbf{v}) = 1$ by definition of an AXp, since $\mathbf{v}$ agrees with the vector 1 on all features in $\mathcal{X}$. We can also note that $v_0 = 1$ since $0 \in \mathcal{X}$. Since $\mathcal{X}$ does not contain i and $i+k$ $(1 \leq i \leq k)$, it follows that $v_i \neq v_{i+k}$. Now let $u_i = 1$ iff $i \in \mathcal{X} \wedge 1 \leq i \leq k$. It is easy to verify that $\Phi(\mathbf{u}) = \tilde{\Phi}(\mathbf{v}) = \kappa(\mathbf{v}) = 1$.

Thus, determining whether $\kappa(\mathbf{1}_{N+1}) = 1$ has an AXp containing the feature 0 is equivalent to testing the satisfiability of Φ. It follows that FRP is NP-hard for monotonic classifiers by this polynomial reduction from SAT. □

Proposition 12. Relevancy for FBDD classifiers is NP-hard.

Proof. Let ψ be a CNF formula defined on a variable set $X = \{x_1, \ldots, x_m\}$ and with clauses $\{\omega_1, \ldots, \omega_n\}$. We aim to construct an FBDD classifier $\mathcal{G}$ (representing a classification function κ) based on ψ and a target variable in polynomial time, such that: ψ is SAT iff for κ there is an AXp containing this target variable.

For any literal $l_j \in \omega_i$, replace l_j with l_j^i. Let $\psi' = \{\omega_1', \ldots, \omega_n'\}$ denote the resulting CNF formula defined on the new variables $\{x_1^1, \ldots, x_m^1, \ldots x_1^n, \ldots, x_m^n\}$. For each original variable x_j, let I_j^+ and I_j^- denote the indices of clauses containing literal x_j and $\neg x_j$, respectively. So if $i \in I_j^+$, then $x_j^i \in \omega_i'$, if $i \in I_j^-$, then $\neg x_j^i \in \omega_i'$. To build an FBDD D from ψ': 1) build an FBDD D_i for each ω_i'; 2) replace the terminal node 1 of D_i with the root node of D_{i+1}; D is read-once because each variable x_j^i occurs only once in ψ'. Satisfying a literal $x_j^i \in \omega_i'$ means $x_j = 1$, while satisfying a literal $\neg x_j^k \in \omega_k'$ means $x_j = 0$. If both x_j^i and $\neg x_j^k$ are satisfied, then it means we pick inconsistent values for the variable x_j, which is unacceptable. Let us define ϕ to capture inconsistent values for any variable x_j:

$$\phi := \bigvee\nolimits_{1 \leq j \leq m} \left(\left(\bigvee\nolimits_{i \in I_j^+} x_j^i \right) \wedge \left(\bigvee\nolimits_{k \in I_j^-} \neg x_j^k \right) \right) \tag{4}$$

If $I_j^+ = \emptyset$, then let $\left(\bigvee_{i \in I_j^+} x_j^i \right) = 0$. If $I_j^- = \emptyset$, then let $\left(\bigvee_{k \in I_j^-} \neg x_j^k \right) = 0$. Any true point of ϕ means we pick inconsistent values for some variable x_j, so it represents an unacceptable point of ψ. To avoid such inconsistency, one needs to at least falsify either $\bigvee_{i \in I_j^+} x_j^i$ or $\bigvee_{k \in I_j^-} \neg x_j^k$ for each variable x_j. To build an FBDD G from ϕ: 1) build FBDDs G_j^+ and G_j^- for $\bigvee_{i \in I_j^+} x_j^i$ and $\bigvee_{k \in I_j^-} \neg x_j^k$, respectively; 2) replace the terminal node 1 of G_j^+ with the root node of G_j^-, let G_j denote the resulting FBDD; 3) replace the terminal 0 of G_j with the root node of G_{j+1}; G is read-once because each variable x_j^i occurs only once in ϕ.

Create a root node labeled x_0^0, link its 1-edge to the root of D, 0-edge to the root of G. The resulting graph $\mathcal{G}$ is an FBDD representing $\kappa := (x_0^0 \wedge \psi') \vee (\neg x_0^0 \wedge \phi)$, κ is a boolean classifier defined on $\{x_0^0, x_1^1, \dots, x_m^n\}$ and x_0^0 is the target variable. The number of nodes of $\mathcal{G}$ is $O(n \times m)$. Let $\mathcal{I} = \{(0,0), (1,1), \dots (n,m)\}$ denote the set of variable indices, for variable x_j^i, $(i,j) \in \mathcal{I}$.

Pick an instance $\mathbf{v} = \{v_0^0, \dots, v_j^i, \dots\}$ satisfying every literal of ψ' (i.e. $v_j^i = 1$ and $v_j^k = 0$ for $x_j^i, \neg x_j^k \in \psi'$) and such that $v_0^0 = 1$, then $\psi'(\mathbf{v}) = 1$, and so $\kappa(\mathbf{v}) = 1$. Suppose $\mathcal{X} \subseteq \mathcal{I}$ is an AXp of $\mathbf{v}$: 1) If $\{(i,j),(k,j)\} \subseteq \mathcal{X}$ for some variable x_j, where $i \in I_j^+$ and $k \in I_j^-$, then for any point $\mathbf{u}$ of κ such that $u_j^i = v_j^i$ for any $(i,j) \in \mathcal{X}$, we have $\kappa(\mathbf{u}) = 1$ and $\phi(\mathbf{u}) = 1$. Moreover, if $\mathbf{u}$ sets $u_0^0 = 1$, then $\kappa(\mathbf{u}) = 1$ implies $\psi'(\mathbf{u}) = 1$, else if $\mathbf{u}$ sets $u_0^0 = 0$, then $\kappa(\mathbf{u}) = 1$ because of $\phi(\mathbf{u}) = 1$. $\kappa(\mathbf{u}) = 1$ regardless the value of u_0^0, so $(0,0) \notin \mathcal{X}$. 2) If $\{(i,j),(k,j)\} \not\subseteq \mathcal{X}$ for any variable x_j, where $i \in I_j^+$ and $k \in I_j^-$, then for some point $\mathbf{u}$ of κ such that $u_j^i = v_j^i$ for any $(i,j) \in \mathcal{X}$, we have $\phi(\mathbf{u}) \neq 1$, in this case $\kappa(\mathbf{u}) = 1$ implies $\psi'(\mathbf{u}) = 1$, besides, any such $\mathbf{u}$ must set $u_0^0 = 1$, so $(0,0) \in \mathcal{X}$.

If case 2) occurs, then ψ is satisfiable. (a satisfying assignment is $x_j = 1$ iff $\exists i \in I_j^+$ s.t. $(i,j) \in \mathcal{X}$). If case 2) never occurs, then ψ is unsatisfiable. It follows that FRP is NP-hard for FBDD classifiers by this polynomial reduction from SAT. □

Corollary 13. Relevancy for d-DNNF classifiers is NP-hard.

4 Feature Relevancy: Example Algorithms

This section details two methods for FRP. One method decides feature relevancy for d-DNNF classifiers, whereas the other method decides feature relevancy for

Table 1: Encoding for deciding whether there is a weak AXp including feature t.

Conditions	Constraints	Fml #
$\mathsf{Leaf}(j), \mathsf{Feat}(j,i), \mathsf{Sat}(\mathsf{Lit}(j), v_i)$	n_j^k	(1.1)
$\mathsf{Leaf}(j), \mathsf{Feat}(j,i), \neg\mathsf{Sat}(\mathsf{Lit}(j), v_i), i = k$	n_j^k	(1.2)
$\mathsf{Leaf}(j), \mathsf{Feat}(j,i), \neg\mathsf{Sat}(\mathsf{Lit}(j), v_i), i \neq k$	$n_j^k \leftrightarrow \neg s_i$	(1.3)
$\mathsf{NonLeaf}(j), \mathsf{Oper}(j) = \vee$	$n_j^k \leftrightarrow \bigvee_{l \in \mathsf{children}(j)} n_l^k$	(1.4)
$\mathsf{NonLeaf}(j), \mathsf{Oper}(j) = \wedge$	$n_j^k \leftrightarrow \bigwedge_{l \in \mathsf{children}(j)} n_l^k$	(1.5)
$\kappa(\mathbf{v}) = 0$	$\neg n_1^0$	(1.6)
$\kappa(\mathbf{v}) = 0$	$s_i \leftrightarrow n_1^i$	(1.7)
	s_t	(1.8)

arbitrary monotonic classifiers. Based on Proposition 2 and Corollary 3, existing algorithm for computing one AXp [35, 36, 52, 53] can be used to decide feature necessity. Hence, there is no need for devising new algorithms. Additionally, the weak AXp returned from the proposed methods (if it exist) can be fed (as a seed) into the algorithms of computing one AXp [35, 53] to extract one AXp in polynomial time.

4.1 Relevancy for d-DNNF Classifiers

This section details a propositional encoding that decides feature relevancy for d-DNNFs. The encoding follows the approach described in the proof of Proposition 9, and comprises two copies ($\mathbb{C}^0$ and $\mathbb{C}^t$) of the same d-DNNF classifier $\mathbb{C}$, $\mathbb{C}^0$ encodes $\mathsf{WAXp}(\mathcal{X})$ (i.e. the prediction of κ remains unchanged), $\mathbb{C}^t$ encodes $\neg\mathsf{WAXp}(\mathcal{X} \setminus \{t\})$ (i.e. the prediction of κ changes). The encoding is polynomial in the size of classifier's representation.

The encoding is applicable to the case $\kappa(\mathbf{x}) = 0$. The case $\kappa(\mathbf{x}) = 1$ can be transformed to $\neg\kappa(\mathbf{x}) = 0$, so we assume both d-DNNF $\mathbb{C}$ and its negation $\neg\mathbb{C}$ are given. To present the constraints included in this encoding, we need to introduce some auxiliary boolean variables and predicates.

1. s_i, $1 \leq i \leq m$. s_i is a selector such that $s_i = 1$ iff feature i is included in a weak AXp candidate $\mathcal{X}$.
2. n_j^k, $1 \leq j \leq |\mathbb{C}|$ and $0 \leq k \leq m$. n_j^k is the indicator of a node j of d-DNNF $\mathbb{C}$ for replica k. The indicator for the root node of k-th replica is n_1^k. Moreover, the semantics of n_j^k is $n_j^k = 1$ iff the sub-d-DNNF rooted at node j in k-th replica is consistent.
3. $\mathsf{Leaf}(j) = 1$ if the node j is a leaf node.
4. $\mathsf{NonLeaf}(j) = 1$ if the node j is a non-leaf node.
5. $\mathsf{Feat}(j, i) = 1$ if the leaf node j is labeled with feature i.
6. $\mathsf{Sat}(\mathsf{Lit}(j), v_i) = 1$ if for leaf node j, the literal on feature i is satisfied by v_i.

The encoding is summarized in Table 1. As literals are d-DNNF leafs, the values of the selector variables only affect the values of the indicator variables of leaf nodes. Constraint (1.1) states that for any leaf node j whose literal is consistent with the given instance, its indicator n_j^k is always consistent regardless of the value of s_i. On the contrary, constraint (1.3) states that for any leaf node j whose literal is inconsistent with the given instance, its indicator n_j^k is consistent iff feature i is not picked, in other words, feature i can take any value. Because replica k ($k > 0$) is used to check the necessity of including feature k in $\mathcal{X}$, we assume the value of the local copy of selector s_k is 0 in replica k. In this case, as defined in constraint (1.2), even though leaf node j labeled feature k has a literal that is inconsistent with the given instance, its indicator n_j^k is consistent. Constraint (1.4) defines the indicator for an arbitrary $\vee$ node j. Constraint (1.5) defines the indicator for an arbitrary $\wedge$ node j. Together, these constraints declare how the consistency is propagated through the entire d-DNNF. Constraint (1.6) states that the prediction of the d-DNNF classifier $\mathbb{C}$ remains 0 since the selected features form a weak AXp. Constraint (1.7) states that if feature i is selected, then removing it will change the prediction of $\mathbb{C}$. Finally, constraint (1.8) indicates that feature t must be included in $\mathcal{X}$.

Example 7. Given the d-DNNF classifier of Fig. 1 and the instance $(\mathbf{v}_1, c_1) = ((0,1,0,0),0)$, suppose that the target feature is 3. We have selectors $\mathbf{s} = \{s_1, s_2, s_3, s_4\}$, and the encoding is as follows:

1. $(n_1^0 \leftrightarrow n_2^0 \vee n_3^0) \wedge (n_2^0 \leftrightarrow n_4^0 \wedge n_5^0) \wedge (n_3^0 \leftrightarrow n_6^0 \wedge n_7^0) \wedge (n_5^0 \leftrightarrow n_8^0 \vee n_9^0) \wedge (n_7^0 \leftrightarrow n_{10}^0 \wedge n_{11}^0) \wedge (n_9^0 \leftrightarrow n_{12}^0 \wedge n_{13}^0) \wedge (n_4^0 \leftrightarrow \neg s_1) \wedge (n_6^0 \leftrightarrow 1) \wedge (n_8^0 \leftrightarrow 1) \wedge (n_{10}^0 \leftrightarrow \neg s_3) \wedge (n_{11}^0 \leftrightarrow \neg s_4) \wedge (n_{12}^0 \leftrightarrow \neg s_2) \wedge (n_{13}^0 \leftrightarrow \neg s_4) \wedge (\neg n_1^0) \wedge (s_3)$
2. $(n_1^3 \leftrightarrow n_2^3 \vee n_3^3) \wedge (n_2^3 \leftrightarrow n_4^3 \wedge n_5^3) \wedge (n_3^3 \leftrightarrow n_6^3 \wedge n_7^3) \wedge (n_5^3 \leftrightarrow n_8^3 \vee n_9^3) \wedge (n_7^3 \leftrightarrow n_{10}^3 \wedge n_{11}^3) \wedge (n_9^3 \leftrightarrow n_{12}^3 \wedge n_{13}^3) \wedge (n_4^3 \leftrightarrow \neg s_1) \wedge (n_6^3 \leftrightarrow 1) \wedge (n_8^3 \leftrightarrow 1) \wedge (n_{10}^3 \leftrightarrow 1) \wedge (n_{11}^3 \leftrightarrow \neg s_4) \wedge (n_{12}^3 \leftrightarrow \neg s_2) \wedge (n_{13}^3 \leftrightarrow \neg s_4) \wedge (s_3 \leftrightarrow n_1^3)$

Given the AXp's listed in Example 3, by solving these formulas we will either obtain $\{1,3\}$ or $\{1,4\}$ as the AXp.

4.2 Relevancy for Monotonic Classifiers

This section describes an algorithm for FRP in the case of monotonic classifiers. No assumption is made regarding the actual implementation of the monotonic classifier.

Abstraction refinement for relevancy. The algorithm proposed in this section iteratively refines an over-approximation (or abstraction) of all the subsets $\mathcal{S}$ of $\mathcal{F}$ such that: i) $\mathcal{S}$ is a weak AXp, and ii) any AXp included in $\mathcal{S}$ also includes the target feature t. Formally, the set of subsets of $\mathcal{F}$ that we are interested in is defined as follows:

$$\mathbb{H} = \{\mathcal{S} \subseteq \mathcal{F} \,|\, \mathsf{WAXp}(\mathcal{S}) \wedge \forall(\mathcal{X} \subseteq \mathcal{S}).\, [\mathsf{AXp}(\mathcal{X}) \rightarrow (t \in \mathcal{X})]\} \tag{5}$$

The proposed algorithm iteratively refines the over-approximation of set $\mathbb{H}$ until one can decide with certainty whether t is included in some AXp. The refinement step involves exploiting counterexamples as these are identified. (The approach is referred to as abstraction refinement FRP, since the use of abstraction refinement can be related with earlier work (with the same name) in model checking [20].) In practice, it will in general be impractical to manipulate such over-approximation of set $\mathbb{H}$ explicitly. As a result, we use a propositional formula (in fact a CNF formula) $\mathcal{H}$, such that the models of $\mathcal{H}$ encode the subsets of features about which we have yet to decide whether each of those subsets only contains AXp's that include t. (Formula $\mathcal{H}$ is defined on a set of Boolean variables $\{s_1, \ldots, s_m\}$, where each s_i is associated with feature i, and assigning $s_i = 1$ denotes that feature i is included in a given set, as described below.) The algorithm then iteratively refines the over-approximation by filtering out sets of sets that have been shown not to be included in $\mathbb{H}$, i.e. the so-called counterexamples.

Algorithm 1 summarizes the proposed approach[8]. Also, Algorithms 2 and 3 provide supporting functions. (For simplicity, the function calls of Algorithms 2 and 3 show the arguments, but not the parameterizations.) Algorithm 1 iteratively uses an NP oracle (in fact a SAT solver) to pick (or *guess*) a subset $\mathcal{P}$ of $\mathcal{F}$, such that any previously picked set is not repeated. Since we are interested in feature t, we enforce that the picked set must include t. (This step is shown in lines 4 to 7.) Now, the features not in $\mathcal{P}$ are deemed universal, and so we need to account for the range of possible values that these universal features can take. For that, we update lower and upper bounds on the predicted classes. For the features in $\mathcal{P}$ we must use the values dictated by $\mathbf{v}$. (This is shown in lines 8 and 9, and it is sound to do because we have monotonicity of prediction.) If the lower and upper bounds differ, then the picked set is not even a weak AXp, and so we can safely remove it from further consideration. This is achieved by enforcing that at least one of the non-picked elements is picked in the future. (As can be observed $\mathcal{H}$ is updated with a positive clause that captures this constraint, as shown in line 11.) If the lower and upper bounds do not differ (i.e. we picked a weak AXp), and if by allowing t to take any value causes the bounds to differ, then we know that any AXp in $\mathcal{P}$ must include t, and so the algorithm reports $\mathcal{P}$ as a weak AXp that is *guaranteed* to be included in $\mathbb{H}$. (This is shown in line 14.) It should be noted that $\mathcal{P}$ is not necessarily an AXp. However, by Proposition 9, $\mathcal{P}$ is guaranteed to be a weak AXp such that *any* of the AXp's contained in $\mathcal{P}$ *must* include feature t. From [53], we know that we can extract an AXp from a weak AXp in polynomial time, and in this case we are guaranteed to always pick one that includes t. Finally, the last case is when allowing t to take any value does not cause the lower and upper bounds to change. This means we picked a set $\mathcal{P}$ that is a weak AXp, but not all AXp's in $\mathcal{P}$ include the target feature t (again due to Proposition 9). As a result, we must prevent the same weak AXp from being re-picked. This is achieved by requiring that at least one of the picked features not be picked again in the feature set. (This is shown in line 16. As can be observed, $\mathcal{H}$ is updated with a negative clause that captures this constraint.)

As can be concluded from Algorithm 1 and from the discussion above, Proposition 9 is essential to enable us to use at most two classification queries per iter-

[8] Arguments can either represent actual arguments or some parameterization; these are separated by a semi-colon.

Algorithm 1 Deciding feature relevancy for a monotonic classifier

```
Input: Instance $\mathbf{v}$, Target feature $t$; Feature Set $\mathcal{F}$, Monotonic Classifier $\kappa$
 1: function DecideRelevant($\mathbf{v}, t; \mathcal{F}, \kappa$)
 2:     $\mathcal{H} \leftarrow \emptyset$                                  ▷ $\mathcal{H}$ overapproximates $\mathbb{H}$
 3:     repeat
 4:         $(\mathsf{outc}, \mathbf{s}) \leftarrow \mathsf{SAT}(\mathcal{H}, s_t)$       ▷ Pick candidate weak AXp containing $t$
 5:         if outc = true then
 6:             $\mathcal{P} \leftarrow \{i \in \mathcal{F} \,|\, s_i = 1\}$   ▷ $\mathcal{P}$ is the candidate weak AXp, and $t \in \mathcal{P}$
 7:             $\mathcal{D} \leftarrow \{i \in \mathcal{F} \,|\, s_i = 0\}$   ▷ $\mathcal{D}$ contains the features not included in $\mathcal{P}$
 8:             $\mathbf{v}_L \leftarrow (v_{L_1}, \ldots, v_{L_N})$, s.t. $v_{L_i} \leftarrow \text{ITE}(s_i, v_i, \lambda(i))$       ▷ $\mathbf{v}_L$: LB
 9:             $\mathbf{v}_U \leftarrow (v_{U_1}, \ldots, v_{U_N})$, s.t. $v_{U_i} \leftarrow \text{ITE}(s_i, v_i, \mu(i))$       ▷ $\mathbf{v}_U$: UB
10:             if $\kappa(\mathbf{v}_L) \neq \kappa(\mathbf{v}_U)$ then          ▷ More than one value possible?
11:                 $\mathcal{H} \leftarrow \mathcal{H} \cup \mathsf{newPosCl}(\mathcal{D}, t)$      ▷ $\mathcal{P}$ is *not* a weak AXp; block set
12:             else                                     ▷ $\mathcal{P}$ is a weak AXp
13:                 if $\kappa(\mathbf{v}_L[v_{L_t} \leftarrow \lambda(t)]) \neq \kappa(\mathbf{v}_U[v_{U_t} \leftarrow \mu(t)])$ then   ▷ $t$ needed?
14:                     $\mathsf{reportWeakAXp}(\mathcal{P})$      ▷ $t$ is included in any AXp $\mathcal{X} \subseteq \mathcal{P}$
15:                     return true
16:                 $\mathcal{H} \leftarrow \mathcal{H} \cup \mathsf{newNegCl}(\mathcal{P}, t)$            ▷ $t$ unneeded; block set
17:     until outc = false
18:     return false        ▷ If $\mathcal{H}$ becomes inconsistent, then *no* AXp contains $t$
```

Table 2: Example algorithm execution for $t = 4$

$\mathbf{s}$	$\mathcal{P}$	$\mathcal{D}$	$\kappa(\mathbf{v}_L)$	$\kappa(\mathbf{v}_U)$	Decision	New clause	Line
$(0,0,0,1)$	$\{4\}$	$\{1,2,3\}$	0	1	New pos clause	$(s_1 \vee s_2 \vee s_3)$	11
$(1,0,0,1)$	$\{1,4\}$	$\{2,3\}$	0	1	New pos clause	$(s_2 \vee s_3)$	11
$(1,1,0,1)$	$\{1,2,4\}$	$\{3\}$	1	1	New neg clause	$(\neg s_1 \vee \neg s_2)$	16
$(1,0,1,1)$	$\{1,3,4\}$	$\{2\}$	1	1	New neg clause	$(\neg s_1 \vee \neg s_3)$	16
$(0,1,1,1)$	$\{2,3,4\}$	$\{1\}$	1	1	New pos clause	(s_1)	11
—	—	—	–	–	$\mathcal{H}$ inconsistent	–	17

ation of the algorithm. If we were to use Proposition 5 instead, then the number of classification queries would be significantly larger.

Example 8. We consider the monotonic classifier of Fig. 2, with instance $(\mathbf{v}, c) = ((1,1,1,1), 1)$. Table 2 summarizes a possible execution of the algorithm when $t = 4$. Similarly, Table 3 summarizes a possible execution of the algorithm when $t = 1$. (As with the current implementation, and for both examples, the creation of clauses uses no optimizations.) In general, different executions will be determined by the models returned by the SAT solver.

With respect to the clauses that are added to $\mathcal{H}$ at each step, as shown in Algorithms 2 and 3, one can envision optimizations (shown lines 2 to 7 in both algorithms) that heuristically aim at removing features from the given sets, and so produce shorter (and so logically stronger) clauses. The insight is that any feature, which can be deemed irrelevant for the condition used for constructing

Algorithm 2 Create new pos. clause

```
Input: Set $\mathcal{D}$, $t$; $\kappa$, $\mathbf{v}_L$, $\mathbf{v}_U$
1: function newPosCl($\mathcal{D}, t; \kappa, \mathbf{v}_L, \mathbf{v}_U$)
2:     for all $i \in \mathcal{D}$ do
3:         $(v_{L_i}, v_{U_i}) \leftarrow (v_i, v_i)$
4:         if $\kappa(\mathbf{v}_L) \neq \kappa(\mathbf{v}_U)$ then
5:             $\mathcal{D} \leftarrow \mathcal{D} \setminus \{i\}$
6:         else
7:             $(v_{L_i}, v_{U_i}) \leftarrow (\lambda(i), \mu(i))$
8:     $\omega \leftarrow (\bigvee_{i \in \mathcal{D}} s_i)$
9:     return $\omega$
```

Algorithm 3 Create new neg. clause

```
Input: Set $\mathcal{P}$, $t$; $\kappa$, $\mathbf{v}_L$, $\mathbf{v}_U$
1: function newNegCl($\mathcal{P}, t; \kappa, \mathbf{v}_L, \mathbf{v}_U$)
2:     for all $i \in \mathcal{P} \setminus \{t\}$ do
3:         $(v_{L_i}, v_{U_i}) \leftarrow (\lambda(i), \mu(i))$
4:         if $\kappa(\mathbf{v}_L) = \kappa(\mathbf{v}_U)$ then
5:             $\mathcal{P} \leftarrow \mathcal{P} \setminus \{i\}$
6:         else
7:             $(v_{L_i}, v_{U_i}) \leftarrow (v_i, v_i)$
8:     $\omega \leftarrow (\bigvee_{i \in \mathcal{P} \setminus \{t\}} \neg s_i)$
9:     return $\omega$
```

Table 3: Example algorithm execution for $t = 1$

$\mathbf{s}$	$\mathcal{P}$	$\mathcal{D}$	$\kappa(\mathbf{v}_L)$	$\kappa(\mathbf{v}_U)$	Decision	New clause	Line
$(1,0,0,0)$	$\{1\}$	$\{2,3,4\}$	0	1	New pos clause	$(s_2 \vee s_3 \vee s_4)$	11
$(1,1,0,0)$	$\{1,2\}$	$\{3,4\}$	1	1	Weak AXp: $\{1,2\}$	–	14

the clause, can be safely removed from the set. (In practice, our experiments show that the time running the classifier is far larger than the time spent using the NP oracle to guess sets. Thus, we opted to use the simplest approach for constructing the clauses, and so reduce the number of classification queries.)

Given the above discussion, we can conclude that the proposed algorithm is sound, complete and terminating for deciding feature relevancy for monotonic classifiers. (The proof is straightforward, and it is omitted for the sake of brevity.)

Proposition 14. For a monotonic classifier $\mathbb{C}$, defined on set of features $\mathcal{F}$, with κ mapping $\mathbb{F}$ to $\mathcal{K}$, and an instance $(\mathbf{v}, c)$, $\mathbf{v} \in \mathbb{F}$, $c \in \mathcal{K}$, and a target feature $t \in \mathcal{F}$, Algorithm 1 returns a set $\mathcal{P} \subseteq \mathcal{F}$ iff $\mathcal{P}$ is a weak AXp for $(\mathbf{v}, c)$, with the property that any AXp $\mathcal{X} \subseteq \mathcal{P}$ is such that $t \in \mathcal{X}$ (i.e. $\mathcal{P}$ is a witness for the relevancy of t).

5 Experimental Results

This section reports the experimental results on FRP for the d-DNNF and monotonic classifiers. The goal is to show that FRP is practically feasible. We opt not to include experiments for FNP as the complexity of FNP is in P. Besides, to the best of our knowledges, there is no baseline to compare with. The experiments were performed on a MacBook Pro with a 6-Core Intel Core i7 2.6 GHz processor with 16 GByte RAM, running macOS Monterey.

d-DNNF classifiers. For d-DNNFs, we pick its subset SDDs as our target classifier. SDDs support polynomial time negation, so given a SDD $\mathbb{C}$, one can obtain its negation $\neg\mathbb{C}$ efficiently.

Table 4: Solving FRP for SDDs. Sub-Columns Avg. #var and Avg. #cls show, respectively, the average number of variables and clauses in a CNF encoding. Column Runtime reports maximum and average time in seconds for deciding FRP.

Dataset	SDD		%Y	CNF		Runtime (s)	
	#Features	#Nodes		Avg. #var	Avg. #cls	Max	Avg.
Accidents	415	8863	97	26513	78276	56.4	3.5
Audio	272	7224	88	31148	100972	663.1	22.0
DNA	513	8570	91	29155	91288	86.3	11.0
Jester	254	7857	85	35998	121508	362.1	22.7
KDD	306	8109	99	26402	83480	111.2	2.8
Mushrooms	248	7096	91	23874	82112	266.3	15.8
Netflix	292	7039	94	25520	83324	105.7	4.2
NLTCS	183	6661	100	19817	58494	1.4	0.5
Plants	244	6724	97	25356	84782	950.7	20.6
RCV-1	410	9472	90	33438	102500	153.6	11.2
Retail	341	3704	87	10601	28342	1.8	1.1

Monotonic classifiers. For monotonic classifiers, we consider the Deep Lattice Network (DLN) [70] as our target classifier. Since our approach for monotonic classifier is model-agnostic, it could also be used with other approaches for learning monotonic classifiers [48, 69] including Min-Max Network [21, 64] and COMET [65].

Prototype implementation. Prototype implementations of the proposed approaches were implemented in Python [9]. The PySAT toolkit [10] was used for propositional encodings. Besides, PySAT invokes the Glucose 4 [11] SAT solver to pick a weak AXp candidate. SDDs were loaded by using the PySDD [12]package.

Benchmarks & training. For SDDs, we selected 11 datasets from Density Estimation Benchmark Datasets[13]. [34, 46, 49]. 11 datasets were used to learn SDD using LearnSDD [11] (with parameter *maxEdges=20000*). The obtained SDDs were used as binary classifiers. For DLNs, we selected 5 publicly available datasets: *australian* (aus), *breast_ cancer* (b.c.), *heart_ c*, *nursery* [57] and *pima* [2]. We used the three-layer DLN architecture: Calibrators → Random Ensemble of Lattices → Linear Layer. All calibrators for all models used a fixed number of 20 keypoints. And the size of all lattices was set to 3.

Results for SDDs. For each SDD, 100 test instances were randomly generated. All tested instances have prediction 0. (We didn't pick instances predicted to class 1 as this requires the compilation of a new classifier which may have dif-

[9] https://github.com/XuanxiangHuang/frp-experiment
[10] https://github.com/pysathq/pysat
[11] https://www.labri.fr/perso/lsimon/glucose/
[12] https://github.com/wannesm/PySDD
[13] https://github.com/UCLA-StarAI/Density-Estimation-Datasets

Table 5: Solving FRP for DLN. Column Runtime reports maximum and average time in seconds for deciding FRP. Column SAT Time (resp. $\kappa(\mathbf{v})$ Time) reports maximum and average time in seconds for SAT solver (resp. calling DLN's predict function) to decide FRP. Column SAT Calls (resp. $\kappa(\mathbf{v})$ Calls) reports maximum and average number of calls to the SAT solver (resp. to the DLN's predict function) to decide FRP.

Dataset	%Y	Runtime (s)		SAT Time		SAT Calls		$\kappa(\mathbf{v})$ Time		$\kappa(\mathbf{v})$ Calls		$\frac{\kappa(\mathbf{v})\text{Time}}{\text{Runtime}}$
		Max	Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max	Avg.	
aus	61	40.4	8.31	0.02	0.01	291	65	40.0	8.15	424	98	97.8%
b.c.	45	5.4	1.93	0.00	0.00	53	20	5.3	1.89	78	30	98.0%
heart_c	35	31.5	6.67	0.02	0.00	171	54	31.1	6.52	249	80	97.7%
nursery	45	4.3	1.77	0.00	0.00	31	13	4.3	1.75	73	30	98.6%
pima	74	3.7	1.41	0.00	0.00	33	13	3.7	1.39	47	22	98.4%

ferent size). Besides, for each instance, we randomly picked a feature appearing in the model. Hence for each SDD, we solved 100 queries. Table 4 summarizes the results. It can be observed that the number of nodes of the tested SDD is in the range of 3704 and 9472, and the number of features of tested SDD is in the range of 183 and 513. Besides, the percentage of examples for which the answer is Y (i.e. target feature is in some AXp) ranges from 85% to 100%. Regarding the runtime, the largest running time for solving one query can exceed 15 minutes. But the average running time to solve a query is less than 25 seconds, this highlights the scalability of the proposed encoding.

Results for DLNs. For each DLN, we randomly picked 200 tested instances, and for each tested instance, we randomly pick a feature. Hence for each DLN, we solved 200 queries. Table 5 summarizes the results. The use of a SAT solver has a negligible contribution to the running time. Indeed, for all the examples shown, at least 97% of the running time is spent running the classifier. This should be unsurprising, since the number of the iterations of Algorithm 1 never exceeds a few hundred. (The fraction of a second reported in some cases should be divided by the number of calls to the SAT solver; hence the time spent in each call to the SAT solver is indeed negligible.) As can be observed, the percentage of examples for which the answer is Y (i.e. target feature is in some AXp and the algorithm returns **true**) ranges from 35% to 74%. There is no apparent correlation between the percentage of Y answers and the number of iterations. The large number of queries accounts for the number of times the DLN is queried by Algorithm 1, but it also accounts for the number of times the DLN is queried for extracting an AXp from set $\mathcal{P}$ (i.e. the witness) when the algorithm's answer is **true**. A loose upper bound on the number of queries to the classifier is $4 \times \text{NS} + 2 \times |\mathcal{F}|$, where NS is the number of SAT calls, and $|\mathcal{F}|$ is the number of features. Each iteration of Algorithm 1 can require at most 4 queries to the classifier. After reporting $\mathcal{P}$, at most 2 queries per feature will be required to extract the AXp (see Section 2.3). As can be observed this loose upper bound is respected by the reported results.

6 Related Work

The problems of necessity and relevancy have been studied in logic-based abduction since the early 90s [25, 30, 61]. However, this earlier work did not consider the classes of (classifier) functions that are considered in this paper.

There has been recent work on explainability queries [7, 8, 36]. Some of these queries can be related with feature relevancy and necessity. For example, relevancy and necessity have been studied with respect to a target class [7, 8], in contrast with our approach that studies a concrete instance, and so can be naturally related with earlier work on abduction. Recent work [36] studied feature relevancy under the name feature membership, but neither d-DNNF nor monotonic classifiers were discussed. Moreover, [36] only proved the hardness of deciding feature relevancy for DNF and DT classifiers and did not discuss the feature necessity problem. The results presented in this paper complement this work. Besides, the complexity results of FRP and FNP in this paper also complement the recent work [54] which summarizes the progress of formal explanations. [40] focused on the computation of one arbitrary AXp and one smallest AXp, which is orthogonal to our work. Computing one AXp does not guarantee that either FRP or FNP is decided, since the target feature t may not appear in the computed AXp. [53] studied the computation of one formal explanation and the enumeration of formal explanations in the case study of monotonic classifiers. However, neither FRP or FNP were identified and studied.

7 Conclusions

This paper studies the problems of feature necessity and relevancy in the context of formal explanations of ML classifiers. The paper proves several complexity results, some related with necessity, but most related with relevancy. Furthermore, the paper proposes two different approaches for solving relevancy for two families of classifiers, namely classifiers represented with the d-DNNF propositional language, and monotonic classifiers. The experimental results confirm the practical scalability of the proposed algorithms. Future work will seek to prove hardness results for the families of classifiers for which hardness is yet unknown.

Acknowledgements. This work was supported by the AI Interdisciplinary Institute ANITI, funded by the French program "Investing for the Future – PIA3" under Grant agreement no. ANR-19-PI3A-0004, and by the H2020-ICT38 project COALA "Cognitive Assisted agile manufacturing for a Labor force supported by trustworthy Artificial intelligence", and funded by the Spanish Ministry of Science and Innovation (MICIN) under project PID2019-111544GB-C22, and by a María Zambrano fellowship and a Requalification fellowship financed by Ministerio de Universidades of Spain and by European Union – NextGenerationEU.

References

1. Akers, S.B.: Binary decision diagrams. IEEE Transactions on computers **27**(06), 509–516 (1978)
2. Alcalá-Fdez, J., Fernández, A., Luengo, J., Derrac, J., García, S., Sánchez, L., Herrera, F.: Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. Journal of Multiple-Valued Logic & Soft Computing **17** (2011), https://sci2s.ugr.es/keel/dataset.php?cod=21
3. Amgoud, L., Ben-Naim, J.: Axiomatic foundations of explainability. In: IJCAI. pp. 636–642 (2022)
4. Arenas, M., Baez, D., Barceló, P., Pérez, J., Subercaseaux, B.: Foundations of symbolic languages for model interpretability. In: NeurIPS (2021)
5. Arenas, M., Barceló, P., Romero, M., Subercaseaux, B.: On computing probabilistic explanations for decision trees. CoRR **abs/2207.12213** (2022). https://doi.org/10.48550/arXiv.2207.12213, https://doi.org/10.48550/arXiv.2207.12213
6. Arora, S., Barak, B.: Computational Complexity - A Modern Approach. Cambridge University Press (2009), http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264
7. Audemard, G., Bellart, S., Bounia, L., Koriche, F., Lagniez, J., Marquis, P.: On the computational intelligibility of boolean classifiers. In: KR. pp. 74–86 (2021)
8. Audemard, G., Koriche, F., Marquis, P.: On tractable XAI queries based on compiled representations. In: KR. pp. 838–849 (2020)
9. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PloS one **10**(7), e0130140 (2015)
10. Barceló, P., Monet, M., Pérez, J., Subercaseaux, B.: Model interpretability through the lens of computational complexity. In: NeurIPS (2020)
11. Bekker, J., Davis, J., Choi, A., Darwiche, A., den Broeck, G.V.: Tractable learning for complex probability queries. In: NeurIPS. pp. 2242–2250 (2015), https://github.com/ML-KULeuven/LearnSDD
12. Bengio, Y., LeCun, Y., Hinton, G.E.: Deep learning for AI. Commun. ACM **64**(7), 58–65 (2021), https://doi.org/10.1145/3448250
13. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336. IOS Press (2021), https://doi.org/10.3233/FAIA336
14. Blanc, G., Lange, J., Tan, L.: Provably efficient, succinct, and precise explanations. In: NeurIPS (2021)
15. Boumazouza, R., Alili, F.C., Mazure, B., Tabia, K.: ASTERYX: A model-Agnostic SaT-basEd appRoach for sYmbolic and score-based eXplanations. In: CIKM. pp. 120–129 (2021)
16. Brayton, R.K., Hachtel, G.D., McMullen, C., Sangiovanni-Vincentelli, A.: Logic minimization algorithms for VLSI synthesis, vol. 2. Springer Science & Business Media (1984)
17. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324, https://doi.org/10.1023/A:1010933404324
18. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth (1984)
19. Clark, P., Boswell, R.: Rule induction with cn2: Some recent improvements. In: European Working Session on Learning. pp. 151–163. Springer (1991)

20. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643, https://doi.org/10.1145/876638.876643
21. Daniels, H., Velikova, M.: Monotone and partially monotone neural networks. IEEE Trans. Neural Networks **21**(6), 906–917 (2010)
22. Darwiche, A.: SDD: A new canonical representation of propositional knowledge bases. In: IJCAI. pp. 819–826 (2011)
23. Darwiche, A., Marquis, P.: A knowledge compilation map. J. Artif. Intell. Res. **17**, 229–264 (2002). https://doi.org/10.1613/jair.989
24. Darwiche, A., Marquis, P.: On quantifying literals in boolean logic and its applications to explainable AI. J. Artif. Intell. Res. **72**, 285–328 (2021)
25. Eiter, T., Gottlob, G.: The complexity of logic-based abduction. J. ACM **42**(1), 3–42 (1995), https://doi.org/10.1145/200836.200838
26. Fard, M.M., Canini, K.R., Cotter, A., Pfeifer, J., Gupta, M.R.: Fast and flexible monotonic functions with ensembles of lattices. In: NeurIPS. pp. 2919–2927 (2016)
27. Ferreira, J., de Sousa Ribeiro, M., Gonçalves, R., Leite, J.: Looking inside the black-box: Logic-based explanations for neural networks. In: KR. p. 432–442 (2022)
28. Flach, P.A.: Machine Learning - The Art and Science of Algorithms that Make Sense of Data. CUP (2012)
29. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
30. Friedrich, G., Gottlob, G., Nejdl, W.: Hypothesis classification, abductive diagnosis and therapy. In: ESE. pp. 69–78 (1990)
31. Gergov, J., Meinel, C.: Efficient boolean manipulation with OBDD's can be extended to FBDD's. IEEE Transactions on Computers **43**(10), 1197–1209 (1994). https://doi.org/10.1109/12.324545
32. Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive computation and machine learning, MIT Press (2016), http://www.deeplearningbook.org/
33. Gorji, N., Rubin, S.: Sufficient reasons for classifier decisions in the presence of domain constraints. In: AAAI (February 2022)
34. Haaren, J.V., Davis, J.: Markov network structure learning: A randomized feature generation approach. In: AAAI (2012)
35. Huang, X., Izza, Y., Ignatiev, A., Cooper, M.C., Asher, N., Marques-Silva, J.: Tractable explanations for d-DNNF classifiers. In: AAAI. pp. 5719–5728 (2022)
36. Huang, X., Izza, Y., Ignatiev, A., Marques-Silva, J.: On efficiently explaining graph-based classifiers. In: KR. pp. 356–367 (2021)
37. Ignatiev, A., Izza, Y., Stuckey, P.J., Marques-Silva, J.: Using MaxSAT for efficient explanations of tree ensembles. In: AAAI. pp. 3776–3785 (2022)
38. Ignatiev, A., Marques-Silva, J.: SAT-based rigorous explanations for decision lists. In: SAT. pp. 251–269 (2021)
39. Ignatiev, A., Narodytska, N., Asher, N., Marques-Silva, J.: From contrastive to abductive explanations and back again. In: AIxIA. pp. 335–355 (2020)
40. Ignatiev, A., Narodytska, N., Marques-Silva, J.: Abduction-based explanations for machine learning models. In: AAAI. pp. 1511–1519 (2019)
41. Ignatiev, A., Pereira, F., Narodytska, N., Marques-Silva, J.: A SAT-based approach to learn explainable decision sets. In: IJCAR. pp. 627–645 (2018)
42. Izza, Y., Ignatiev, A., Marques-Silva, J.: On tackling explanation redundancy in decision trees. J. Artif. Intell. Res. **75**, 261–321 (2022), https://doi.org/10.1613/jair.1.13575

43. Izza, Y., Marques-Silva, J.: On explaining random forests with SAT. In: IJCAI. pp. 2584–2591 (2021)
44. Kohavi, R.: Bottom-up induction of oblivious read-once decision graphs: strengths and limitations. In: AAAI. pp. 613–618 (1994)
45. Kohavi, R., et al.: Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In: Kdd. vol. 96, pp. 202–207 (1996)
46. Larochelle, H., Murray, I.: The neural autoregressive distribution estimator. In: AISTATS. pp. 29–37 (2011)
47. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
48. Liu, X., Han, X., Zhang, N., Liu, Q.: Certified monotonic neural networks. In: NeurIPS (2020)
49. Lowd, D., Davis, J.: Learning Markov network structure with decision trees. In: ICDM. pp. 334–343 (2010)
50. Lundberg, S.M., Lee, S.: A unified approach to interpreting model predictions. In: NeurIPS. pp. 4765–4774 (2017)
51. Malfa, E.L., Michelmore, R., Zbrzezny, A.M., Paoletti, N., Kwiatkowska, M.: On guaranteed optimal robust explanations for NLP models. In: IJCAI. pp. 2658–2665 (2021)
52. Marques-Silva, J., Gerspacher, T., Cooper, M.C., Ignatiev, A., Narodytska, N.: Explaining naive bayes and other linear classifiers with polynomial time and delay. In: NeurIPS (2020)
53. Marques-Silva, J., Gerspacher, T., Cooper, M.C., Ignatiev, A., Narodytska, N.: Explanations for monotonic classifiers. In: ICML. pp. 7469–7479 (2021)
54. Marques-Silva, J., Ignatiev, A.: Delivering trustworthy AI through formal XAI. In: AAAI. pp. 12342–12350 (2022)
55. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. Artif. Intell. **267**, 1–38 (2019)
56. Müller, B., Reinhardt, J., Strickland, M.T.: Neural networks: an introduction. Springer Science & Business Media (1995)
57. Olson, R.S., La Cava, W., Orzechowski, P., Urbanowicz, R.J., Moore, J.H.: PMLB: a large benchmark suite for machine learning evaluation and comparison. BioData Mining **10**(1), 36 (2017), https://epistasislab.github.io/pmlb/index.html
58. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why should I trust you?": Explaining the predictions of any classifier. In: KDD. pp. 1135–1144 (2016)
59. Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: High-precision model-agnostic explanations. In: AAAI. pp. 1527–1535 (2018)
60. Rivest, R.L.: Learning decision lists. Mach. Learn. **2**(3), 229–246 (1987)
61. Selman, B., Levesque, H.J.: Abductive and default reasoning: A computational core. In: AAAI. pp. 343–348 (1990)
62. Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning - From Theory to Algorithms. Cambridge University Press (2014)
63. Shih, A., Choi, A., Darwiche, A.: A symbolic approach to explaining bayesian network classifiers. In: IJCAI. pp. 5103–5111 (2018)
64. Sill, J.: Monotonic networks. In: NIPS. pp. 661–667 (1997)
65. Sivaraman, A., Farnadi, G., Millstein, T.D., den Broeck, G.V.: Counterexample-guided learning of monotonic neural networks. In: NeurIPS (2020)
66. Van den Broeck, G., Darwiche, A.: On the role of canonicity in knowledge compilation. In: AAAI. pp. 1641–1648 (2015)

67. Wäldchen, S., MacDonald, J., Hauch, S., Kutyniok, G.: The computational complexity of understanding binary classifier decisions. J. Artif. Intell. Res. **70**, 351–387 (2021), https://doi.org/10.1613/jair.1.12359
68. Wegener, I.: Branching Programs and Binary Decision Diagrams. SIAM (2000), http://ls2-www.cs.uni-dortmund.de/monographs/bdd/
69. Wehenkel, A., Louppe, G.: Unconstrained monotonic neural networks. In: NeurIPS. pp. 1543–1553 (2019)
70. You, S., Ding, D., Canini, K.R., Pfeifer, J., Gupta, M.R.: Deep lattice networks and partial monotonic functions. In: NeurIPS. pp. 2981–2989 (2017), https://github.com/tensorflow/lattice