On Symbolically Encoding the Behavior of Random Forests

Arthur Choi[1], Andy Shih[2], Anchal Goyanka[1], and Adnan Darwiche[1]

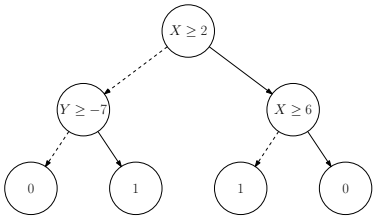
[1] Computer Science Department, UCLA
{aychoi, anchal, darwiche}@cs.ucla.edu
[2] Computer Science Department, Stanford University
andyshih@cs.stanford.edu

Abstract. Recent work has shown that the input-output behavior of some machine learning systems can be captured symbolically using Boolean expressions or tractable Boolean circuits, which facilitates reasoning about the behavior of these systems. While most of the focus has been on systems with Boolean inputs and outputs, we address systems with discrete inputs and outputs, including ones with discretized continuous variables as in systems based on decision trees. We also focus on the suitability of encodings for computing prime implicants, which have recently played a central role in explaining the decisions of machine learning systems. We show some key distinctions with encodings for satisfiability, and propose an encoding that is sound and complete for the given task.

Keywords: Explainable AI · Random Forests · Prime Implicants.

1 Introduction

Recent work has shown that the input-output behavior of some machine learning systems can be captured symbolically using Boolean expressions or tractable Boolean circuits [10, 12, 16, 25, 7, 8, 26, 23]. These encodings facilitate the reasoning about the behavior of these systems, including the explanation of their decisions, the quantification of their robustness and the verification of their properties. Most of the focus has been on systems with Boolean inputs and outputs, with some extensions to discrete inputs and outputs, including discretizations of continuous variables as in systems based on decision trees; see, e.g., [2, 15, 9]. This paper is concerned with the latter case of discrete/continuous systems but those that are encoded using Boolean variables, with the aim of utilizing the vast machinery available for reasoning with Boolean logic. Most prior studies of Boolean encodings have focused on the tasks of satisfiability and model counting [11, 27, 2]. In this paper, we focus instead on prime implicants which have recently played a central role in explaining the decisions of machine learning systems [25, 20, 7–9, 5]; cf. [21]. We first highlight how the prime implicants of a multi-valued expression are not immediately obtainable as prime implicants of a corresponding Boolean encoding. We reveal how to compute these prime implicants, by computing them instead on a Boolean expression derived from



value	interval
x_1	$(-\infty, 2)$
x_2	$[2, 6)$
x_3	$[6, +\infty)$
y_1	$(-\infty, -7)$
y_2	$[-7, +\infty)$

X	Y	$f(X,Y)$
x_1	y_1	0
x_1	y_2	1
x_2	y_1	1
x_2	y_2	1
x_3	y_1	0
x_3	y_2	0

Fig. 1. (Left) A decision tree of continuous variables X and Y, where a solid branch means the test is true, and a dashed branch means false. (Center) A discretization of X and Y into intervals. (Right) The discrete function represented by the decision tree.

the encoding. Our study is conducted in the context of encoding the behavior of random forests using majority voting, but our results apply more broadly.

This paper is structured as follows. We introduce the task in Section 2 as well as review related work. We discuss in Section 3 the problem of explaining the decisions of machine learning systems whose continuous features can be discretized into intervals. We follow in Section 4 by a discussion on encoding the input-output behavior of such systems, where we analyze three encodings from the viewpoint of computing explanations for decisions. We show that one of these encodings is suitable for this purpose, if employed carefully, while proving its soundness and completeness for the given task. We finally close in Section 5.

2 Boolean, Discrete and Continuous Behaviors

The simplest behaviors to encode are for systems with Boolean inputs and outputs. Consider a neural network whose inputs are Boolean and that has only step activation functions. Each neuron in this network computes a Boolean function and therefore each output of the network also computes a Boolean function. The input-output behavior of such networks can be immediately represented using Boolean expressions, or Boolean circuits as proposed in [3, 23].

Suppose now that the inputs to a machine learning system are discrete variables, say, variable A with values $1, 2, 3$, variable B with values r, b, g and variable C with values l, m, h. One can define a multi-valued propositional logic to capture the behavior of such a system. The atomic expressions in this case will be of the form $V{=}v$, indicating that discrete variable V has the value v. We can then construct more complex expressions using Boolean connectives. An example expression in this logic would be $(B{=}r \vee B{=}b) \implies (A{=}2 \wedge \neg C{=}h)$.

Some systems may have continuous variables as inputs, which get discretized during the learning process as is the case with systems based on decision trees. Consider for example the decision tree in Figure 1 (left) over continuous variables X and Y. The algorithm that learned this tree discretized its variables as follows: X to intervals $(-\infty, 2), [2, 6), [6, +\infty)$ and Y to intervals $(-\infty, -7), [-7, +\infty)$.

We can now think of variable X as a discrete variable with three values x_1, x_2, x_3, each corresponding to one of the intervals as shown in Figure 1

(center). Variable Y is binary in this case, with each value corresponding to one of the two intervals. According to this decision tree, the infinite number of input values for variables X and Y can be grouped into six equivalence classes as shown in Figure 1 (right). Hence, the input-output behavior of this decision tree can be captured using the multi-valued propositional expression $f(X, Y) = (X{=}x_1 \wedge Y{=}y_2) \vee X{=}x_2$, even though we have continuous variables.

Our goal is therefore to encode multi-valued expressions using Boolean expressions as we aim to exploit the vast machinery currently available for reasoning with propositional logic. This includes SAT-based and knowledge compilation tools, which have been used extensively recently to reason about the behavior of machine learning systems [10, 12, 16, 25, 7, 8, 26, 23].

Encoding multi-valued expressions using Boolean expressions has been of interest for a very long time and several methods have been proposed for this purpose; see, e.g., [11, 27, 2]. In some cases, different encodings have been compared in terms of the efficacy of applied SAT-based tools; see, e.g., [27]. In this paper, we consider another dimension for evaluating encodings, which is based on their suitability for computing prime implicants. This is motivated by the fundamental role that implicants have been playing recently in explaining the decisions of machine learning systems [25, 20, 7–9, 5]

The previous works use the notion of a *PI-explanation* when explaining the decision of a classifier on an instance. A PI-explanation, introduced in [25], is a minimal set of instance characteristics that are sufficient to trigger the decision. That is, if these characteristics are fixed, other instance characteristics can be changed freely without changing the decision. In an image, for example, a PI-explanation corresponds to a minimal set of pixels that guarantees the stability of a decision against any perturbation of the remaining pixels.[3]

PI-explanations are based on *prime implicants* of Boolean functions, which have been studied extensively in the literature [4, 17, 13, 18]. Consider the following Boolean function over variables A, B and C: $f = (A + \overline{C})(B + C)(A + B)$. A prime implicant of the function is a minimal setting of its variables that causes the function to trigger. This function has three prime implicants: AB, AC and $B\overline{C}$. Consider now the instance $AB\overline{C}$ leading to a positive decision $f(AB\overline{C}) = 1$. The PI-explanations for this decision are the prime implicants of function f that are compatible with the instance: AB and $B\overline{C}$. Explaining negative decisions requires working with the function's complement $\overline{f}$. Consider instance $\overline{A}BC$, which sets the function f to 0. The complement $\overline{f}$ has three prime implicants $\overline{A}C$, $\overline{B}\,\overline{C}$ and $\overline{A}\,\overline{B}$. Only one of these is compatible with the instance, $\overline{A}C$, so it is the only PI-explanation for the decision on this instance.[4]

[3] A PI-explanation can be viewed as a (minimally) *sufficient reason* for the decision [5].

[4] The popular Anchor system [22] can be viewed as computing approximations of PI-explanations. The quality of these approximations has been evaluated on some datasets and corresponding classifiers in [9], where an approximation is called *optimistic* if it is a strict subset of a PI-explanation and *pessimistic* if it is a strict superset of a PI-explanation. Anchor computes approximate explanations without having to abstract the machine learning system behavior into a symbolic representation.

When considering the encoding of multi-valued expressions using Boolean ones, we will be focusing on whether the prime implicants of multi-valued expressions can be soundly and completely obtained from the prime implicants of the corresponding Boolean expressions. This is motivated by the desire to exploit existing algorithms and tools for computing prime implicants of Boolean expressions (one may also consider developing a new set of algorithms and tools for operating directly on multi-valued expressions).

Before we propose and evaluate some encodings, we need to first define the notion of a prime implicant for multi-valued expressions and then examine explanations from that perspective. This is needed to settle the semantics of explanations in a multi-valued setting, which will then form the basis for deciding whether a particular encoding is satisfactory from the viewpoint of computing explanations. As the following discussion will reveal, the multi-valued setting leads to some new considerations that are preempted in a Boolean setting.

3 Explaining Decisions in a Multi-Valued Setting

Consider again the decision tree in Figure 1 whose behavior is captured by the multi-valued expression $(X{=}x_1 \wedge Y{=}y_2) \vee X{=}x_2$ as discussed earlier. Consider also the positive instance $X{=}3 \wedge Y{=}12$, which can be represented using the multi-valued expression $\alpha : X{=}x_2 \wedge Y{=}y_2$ as shown in Figure 1.

Instance α has two characteristics $X{=}x_2$ and $Y{=}y_2$, yet one of them $X{=}x_2$ is sufficient to trigger the positive decision. Hence, one explanation for the decision is that variable X takes a value in the interval $[2, 6)$, which justifies $X{=}x_2$ as a PI-explanation of this positive decision. In fact, if we stick to the literal definition of a PI-explanation from the Boolean setting, then this would be the only PI-explanation since $Y{=}y_2$ is the only characteristic that can be dropped from the instance while guaranteeing that the decision will stick.

Looking closer, this decision would also stick if the value of X were contained in the larger interval $(-\infty, 6)$ as long as characteristic $Y{=}y_2$ is maintained. The interval $(-\infty, 6)$ corresponds to $(X{=}x_1 \vee X{=}x_2)$, leading to the expression $(X{=}x_1 \vee X{=}x_2) \wedge Y{=}y_2$. This expression is the result of *weakening* literal $X{=}x_2$ in instance $X{=}x_2 \wedge Y{=}y_2$. It can be viewed as a candidate explanation of the decision on this instance, just like $X{=}x_2$, in the sense that it also represents an abstraction of the instance that preserves the corresponding decision.

For another example, consider the negative decision on instance $X{=}10 \wedge Y{=}{-}20$, and its corresponding multi-valued expression $\alpha : X{=}x_3 \wedge Y{=}y_1$. Recall that x_3 represents the interval $[6, +\infty)$ and y_1 represents the interval $(-\infty, -7)$. We can drop the characteristic $Y{=}y_1$ from this instance while guaranteeing that the negative decision will stick (i.e., regardless of what value variable Y takes). Hence, $X{=}x_3$ is a PI-explanation in this case. But again, if we maintain the characteristic $Y{=}y_1$, then this negative decision will stick as long as the value of X is in the larger, disconnected interval $(-\infty, 2] \cup [6, +\infty)$. This interval is represented by the expression $X{=}x_1 \vee X{=}x_3$ which is a weakening of characteristic $X{=}x_3$. This makes $(X{=}x_1 \vee X{=}x_3) \wedge Y{=}y_1$ a candidate explanation as well.

3.1 Multi-Valued Literals, Terms and Implicants

We will now formalize some notions on multi-valued variables and then use them to formally define PI-explanations in a multi-valued setting [19, 14]. We use three multi-valued variables for our running examples: Variable A with values $1, 2, 3$, variable B with values r, b, g and variable C with values l, m, h.

A *literal* is a non-trivial propositional expression that mentions a single variable. The following are literals: $B{=}r \vee B{=}b$, $A{=}2$ and $C{\neq}h$. The following are not literals as they are trivial: $B{=}r \vee B{=}b \vee B{=}g$ and $C{=}h \wedge C{\neq}h$. Intuitively, for a variable with n values, a literal specifies a set of values S where the cardinality of set S is in $\{1, \ldots, n-1\}$. A literal is *simple* if it specifies a single value (cardinality of set S is 1). When multi-valued variables correspond to the discretization of continuous variables, our treatment allows a literal to specify non-contiguous intervals of a continuous variable.

Consider two literals ℓ_i and ℓ_j for the same variable. We say ℓ_i is *stronger* than ℓ_j iff $\ell_i \models \ell_j$ and $\ell_i \not\equiv \ell_j$. In this case, ℓ_j is *weaker* than ℓ_i. For example, $B{=}r$ is stronger than $B{=}r \vee B{=}b$. It is possible to have two literals where neither is stronger or weaker than the other (e.g., $B{=}r \vee B{=}b$ and $B{=}g$).

A *term* is a conjunction of literals over distinct variables. The following is a term: $A{=}2 \wedge (B{=}r \vee B{=}b) \wedge C{\neq}h$. A term is *simple* if all of its literals are simple. The following term is simple: $A{=}2 \wedge B{=}r \wedge C{=}h$. The following terms are not simple: $A{\neq}2 \wedge B{=}r \wedge C{=}h$ and $A{=}2 \wedge (B{=}r \vee B{=}b) \wedge C{=}h$. A simple term that mentions every variable is called an *instance.*

Term τ_i *subsumes* term τ_j iff $\tau_j \models \tau_i$. If we also have $\tau_i \not\equiv \tau_j$, then τ_i *strictly subsumes* τ_j. For example, the term $A{=}2 \wedge (B{=}r \vee B{=}b) \wedge C{\neq}h$ is strictly subsumed by the terms $A{\neq}1 \wedge (B{=}r \vee B{=}b) \wedge C{\neq}h$ and $A{=}2 \wedge C{\neq}h$.

We stress two points now. First, if term τ_i strictly subsumes term τ_j that does not necessarily mean that τ_i mentions a fewer number of variables than τ_j. In fact, it is possible that the literals of τ_i and τ_j are over the same set of variables. Second, a term does not necessarily fix the values of its variables (unless it is a simple term), which is a departure from how terms are defined in Boolean logic.

In Boolean logic, the only way to get a term that strictly subsumes term τ is by dropping some literals from τ. In multi-valued logic, we can also do this by weakening some literals in term τ (i.e., without dropping any of its variables). This notion of *weakening* a literal generalizes the notion of *dropping* a literal in the Boolean setting. In particular, dropping a Boolean literal ℓ from a Boolean term can be viewed as weakening it into $\ell \vee \neg\ell$.

Term τ is an *implicant* of expression Δ iff $\tau \models \Delta$. Term τ is a *prime implicant* of Δ iff it is an implicant of Δ that is not strictly subsumed by another implicant of Δ. It is possible to have two terms over the same set of variables such that (a) the terms are compatible in that they admit some common instance, (b) both are implicants of some expression Δ, yet (c) only one of them is a prime implicant of Δ. We stress this possibility as it does not arise in a Boolean setting. We define the notions of *simple implicant* and *simple prime implicant* in the expected way.

3.2 Multi-Valued Explanations

Consider now a *classifier* specified using a multi-valued expression Δ. The variables of Δ will be called *features* so an *instance* α is a simple term that mentions all features. That is, an instance fixes a value for each feature of the classifier. A decision on instance α is positive iff the expression Δ evaluates to 1 on instance α, written $\Delta(\alpha) = 1$. Otherwise, the decision is negative (when $\Delta(\alpha) = 0$).

The notation Δ_α is crucial for defining explanations: Δ_α is defined as Δ if decision $\Delta(\alpha)$ is positive and Δ_α is defined as $\neg\Delta$ if decision $\Delta(\alpha)$ is negative. A *PI-explanation* for decision $\Delta(\alpha)$ is a prime implicant of Δ_α that is consistent with instance α. This basically generalizes the notion of PI-explanation introduced in [25] to a multi-valued setting.

The term *explanation* is somewhat too encompassing so any definition of this general notion is likely to draw criticism as being too narrow. The *PI-explanation* is indeed narrow as it is based on a syntactic restriction: it must be a conjunction of literals (i.e., a term) [25]. In the Boolean setting, a PI-explanation is a minimal subset of instance characteristics that is sufficient to trigger the same decision made on the instance. In the multi-valued setting, it can be more generally described as an *abstraction* of the instance that triggers the same decision made on the instance (still in the syntactic form of a term).

As an example, consider the following truth table representing the decision function of a classifier over two ternary variables X and Y:

X, Y	x_1y_1	x_1y_2	x_1y_3	x_2y_1	x_2y_2	x_2y_3	x_3y_1	x_3y_2	x_3y_3
$f(X, Y)$	1	0	0	1	0	0	1	1	1

Consider instance $X{=}x_3 \wedge Y{=}y_1$ leading to a positive decision. The sub-term $X{=}x_3$ is a PI-explanation for this decision: setting input X to x_3 suffices to trigger a positive decision. Similarly, the sub-term $Y{=}y_1$ is a second PI-explanation for this decision. Consider now instance $X{=}x_1 \wedge Y{=}y_2$ leading to a negative decision. This decision has a single PI-explanation: $X{\neq}x_3 \wedge Y{\neq}y_1$. Any instance consistent with this explanation will be decided negatively.

4 Encoding Multi-Valued Behavior

We next discuss three encodings that we tried for the purpose of symbolically representing the behavior of decision trees (and random forests). The first two encodings turned out unsuitable for computing prime implicants. Here, *suitability* refers to the ability of computing multi-valued prime implicants by processing Boolean prime implicants *locally* and *independently*. The third encoding, based on a classical encoding [11], was suitable for this purpose but required a usage that deviates from tradition. Using this encoding in a classical way makes it unsuitable as well. The summary of the findings below is that while an encoding may be appropriate for testing satisfiability or counting models, it may not be suitable for computing prime implicants (and, hence, explanations). While much attention was given to encodings in the context of satisfiability and model counting, we are not aware of similar treatments for computing prime implicants.

4.1 Prefix Encoding

Consider a multi-valued variable X with values $x_1, \ldots, x_n$. This encoding uses Boolean variables $x_2, \ldots, x_n$ to encode the values of variable X. Literal $X{=}x_i$ is encoded by setting the first $i-1$ Boolean variables to 1 and the rest to 0. For example, if $n = 3$, the values of X are encoded as $\bar{x}_2\bar{x}_3$, $x_2\bar{x}_3$ and x_2x_3. Some instantiations of these Boolean variables will not correspond to any value of variable X and are ruled out by enforcing the following constraint: all Boolean variables set to 1 must occur before all Boolean variables set to 0. We denote this constraint by Ψ_X: $\bigwedge_{i\in\{3,\ldots,n\}}(x_i \Rightarrow x_{i-1})$.

The fundamental problem with this encoding is that a multi-valued literal that represents non-contiguous values cannot be represented by a Boolean term. Hence, this encoding cannot generate prime implicants that include such literals. Consider the multi-valued expression $\Delta = (X{=}x_1 \vee X{=}x_3)$, where X has values $x_1, \ldots, x_4$, and its Boolean encoding $\Delta_b = \bar{x}_2\bar{x}_3\bar{x}_4 + x_2x_3\bar{x}_4$. There is only one prime implicant of Δ, which is $X{=}x_1 \vee X{=}x_3$, but this prime implicant cannot be represented by a Boolean term (that implies Δ_b) so it will never be generated.

4.2 Highest-Bit Encoding

Consider a multi-valued variable X with values $x_1, x_2, \ldots, x_n$. This encoding uses Boolean variables $x_2, x_3, \ldots, x_n$ to encode the values of variable X. Every instantiation of these Boolean variables will map to a value of variable X in the following way. If all Boolean variables are 0, then we map the instantiation to value x_1. Otherwise we map an instantiation to the maximum index whose variable is 1. The following table provides an example for $n = 4$.

$x_2x_3x_4$	000	001	010	011	100	101	110	111
highest 1-index	-	4	3	4	2	4	3	4
value	x_1	x_4	x_3	x_4	x_2	x_4	x_3	x_4

We can alternatively view this encoding as representing literal $X{=}x_1$ using the Boolean term $\bar{x}_2 \ldots \bar{x}_n$ and literal $X{=}x_i$, $i \geq 2$, using the term $x_i\bar{x}_{i+1} \ldots \bar{x}_n$. Literals over multiple values can also be represented with this encoding. For example, we can represent the literal $X{=}x_1 \vee X{=}x_2$ using the term $\bar{x}_3\bar{x}_4$.

This encoding also turned out to be unsuitable for computing prime implicants. Consider the multi-valued expression $\Delta = (X{=}x_1 \vee X{=}x_3)$, which has one prime implicant Δ. The Boolean encoding Δ_b is $\bar{x}_2\bar{x}_3\bar{x}_4 + x_3\bar{x}_4$ and has *two* prime implicants $\bar{x}_2\bar{x}_4$ and $x_3\bar{x}_4$. The term $x_3\bar{x}_4$ corresponds to the multi-valued implicant $X{=}x_3$, which is not prime. The term $\bar{x}_2\bar{x}_4$ does not even correspond to a multi-valued term. So in this encoding too, prime implicants of the original multi-valued expression Δ cannot be computed by locally and independently processing prime implicants of the encoded Boolean expression Δ_b.

4.3 One-Hot Encoding

The prefix and highest-bit encodings provide some insights into requirements that enable one to locally and independently map Boolean prime implicants into

multi-valued ones. The requirements are: (1) every multi-valued literal should be representable using a Boolean term, and (2) equivalence and subsumption relations over multi-valued literals should be preserved over their Boolean encodings. The next encoding satisfies these requirements. It is based on [11] but deviates from it in some significant ways that we explain later.

Suppose X is a multi-valued variable with values $x_1, \ldots, x_n$. This encoding uses a Boolean variable x_i for each value x_i of variable X. Suppose now that ℓ is a literal that specifies a subset S of these values. The literal will be encoded using the *negative* Boolean term $\bigwedge_{x_i \notin S} \bar{x}_i$. For example, if variable X has three values, then literal $X{=}x_2$ will be encoded using the negative Boolean term $\bar{x}_1\bar{x}_3$ and literal $X{=}x_1 \vee X{=}x_2$ will be encoded using the negative Boolean term $\bar{x}_3$. This encoding requires the employment of an exactly-one constraint for each variable X, which we denote by Ψ_X: $(\bigvee_i x_i) \wedge \bigwedge_{i \neq j} \neg(x_i \wedge x_j)$. We also use Ψ to denote the conjunction of all exactly-one constraints.

Using the encoding in [11], one typically represents literal $X{=}x_i$ by the Boolean term x_i which *asserts* value x_i. Our encoding, however, represents this literal by *eliminating* all other values of X. The following result reveals why we made this choice (proofs of results can be found in the appendix).

Proposition 1. *Multi-valued terms correspond one-to-one to negative Boolean terms that are consistent with Ψ. Equivalence and subsumption relations on multi-valued terms are preserved on their Boolean encodings.*

Exactly-one constraints are normally added to an encoding as done in [11]. We next show that this leads to unintended results when computing prime implicants, requiring another deviation from [11]. Consider two ternary variables X and Y, the expression $\Delta : X{=}x_1 \vee Y{=}y_1$ and its Boolean encoding $\Delta_b : \bar{x}_2\bar{x}_3 + \bar{y}_2\bar{y}_3$. If Ψ is the conjunction of all exactly-one constraints ($\Psi = \Psi_X \wedge \Psi_Y$), then Δ and $\Delta_b \wedge \Psi$ will each have five models:

Δ	$X{=}x_1, Y{=}y_1$	$X{=}x_1, Y{=}y_2$	$X{=}x_1, Y{=}y_3$	$X{=}x_2, Y{=}y_1$	$X{=}x_3, Y{=}y_1$
$\Delta_b \wedge \Psi$	$x_1\bar{x}_2\bar{x}_3y_1\bar{y}_2\bar{y}_3$	$x_1\bar{x}_2\bar{x}_3\bar{y}_1y_2\bar{y}_3$	$x_1\bar{x}_2\bar{x}_3\bar{y}_1\bar{y}_2y_3$	$\bar{x}_1x_2\bar{x}_3y_1\bar{y}_2\bar{y}_3$	$\bar{x}_1\bar{x}_2x_3y_1\bar{y}_2\bar{y}_3$

The term $X{=}x_1$ is an implicant of Δ. However, its corresponding Boolean encoding $\bar{x}_2\bar{x}_3$ is not an implicant of $\Delta_b \wedge \Psi$ (neither is $x_1\bar{x}_2\bar{x}_3$). For example, $x_1\bar{x}_2\bar{x}_3y_1y_2\bar{y}_3$ does not imply $\Delta_b \wedge \Psi$ since $y_1y_2\bar{y}_3$ does not satisfy the exactly-one constraint Ψ_Y. This motivates Definition 1 below and further results on handling exactly-one constraints, which we introduce after some notational conventions.

In what follows, we use Δ/τ to denote multi-valued expressions/terms, and Γ/ρ to denote Boolean expressions/terms. We also use Δ_b and τ_b to denote the Boolean encodings of Δ and τ. A *completion* of a term is a complete variable instantiation that is consistent with the term. We use α to denote completions. Finally, we use Ψ to denote the conjunction of all exactly-one constraints.

Definition 1. *We define $\rho \models_\Psi \Gamma$ iff $\alpha \models \Gamma$ for all completions α of Boolean term ρ that are consistent with constraint Ψ.*

Note that $\rho \models \Gamma$ implies $\rho \models_\Psi \Gamma$ but the converse is not true.

Proposition 2. *$\rho \models_\Psi \Gamma$ iff $\rho \models (\Psi \Rightarrow \Gamma)$.*

We now show how one-hot encodings can be used for computing prime implicants, particularly, how exactly-one constraints should be integrated.

Proposition 3. *If τ is a term, then $\tau \models \Delta$ iff $\tau_b \models (\Psi \Rightarrow \Delta_b)$.*

The proof is based on two lemmas that hold by construction and that use the notion of *full encoding* of an instance. Consider ternary variables X and Y. For instance $\tau : X{=}x_1 \wedge Y{=}y_1$ the full encoding is $\rho : x_1\bar{x}_2\bar{x}_3y_1\bar{y}_2\bar{y}_3$ (x_1 and y_1 are included). Note that $\rho \wedge \Psi = \rho$ since ρ is guaranteed to satisfy constraints Ψ.

Lemma 1. *If τ is an instance and ρ is its full encoding, then $\tau \models \Delta$ iff $\rho \models \Delta_b$.*

Lemma 2. *For term τ, there is a one-to-one correspondence between the completions of τ and the completions of τ_b that are consistent with Ψ.*

Term $\tau : X{=}x_1 \vee X{=}x_2$ has six completions: $X{=}x_1 \wedge Y{=}y_1$, $X{=}x_2 \wedge Y{=}y_1$, …, $X{=}x_2 \wedge Y{=}y_3$. Its Boolean encoding $\tau_b : \bar{x}_3$ also has six completions that are consistent with Ψ: $x_1\bar{x}_2\bar{x}_3y_1\bar{y}_2\bar{y}_3$, $\bar{x}_1x_2\bar{x}_3y_1\bar{y}_2\bar{y}_3$, …, $\bar{x}_1x_2\bar{x}_3\bar{y}_1\bar{y}_2y_3$. Each of these completions α is guaranteed to satisfy constraints Ψ leading to $\alpha \wedge \Psi = \alpha$. Next, we relate the prime implicants of multi-valued expressions and their encodings.

Proposition 4. *Consider a multi-valued expression Δ and its Boolean encoding Δ_b. If τ is a prime implicant of Δ, then τ_b is a negative term, consistent with Ψ and a prime implicant of $\Psi \Rightarrow \Delta_b$. If ρ is a prime implicant of $\Psi \Rightarrow \Delta_b$, negative and consistent with Ψ, then ρ encodes a prime implicant of Δ.*

This proposition suggests the following procedure for computing multi-valued prime implicants from Boolean prime implicants. Given a multi-valued expression Δ, we encode each literal in Δ using its negative Boolean term, leading to the Boolean expression Δ_b. We then construct the exactly-one constraints Ψ and compute prime implicants of $\Psi \Rightarrow \Delta_b$, keeping those that are negative and consistent with constraints Ψ.[5] Those Boolean prime implicants correspond precisely to the multi-valued prime implicants of Δ.[6]

The only system we are aware of that computes prime implicants of decision tree encodings (and forests) is Xplainer [9]. This system bypasses the encoding complications we alluded to earlier as it computes prime implicants in a specific manner [6, 7]. In particular, it encodes a multi-valued expression into a Boolean expression using the classical one-hot encoding. But rather than computing

[5] It is straightforward to augment the algorithm of [25] so that it only enumerates such prime implicants, by blocking the appropriate branches.

[6] Note that when computing PI-explanations, we are interested only in prime implicants that are consistent with a given instance. Any negative prime implicant which is consistent with an instance must also be consistent with constraints Ψ. The only way a negative Boolean term ρ can violate constraints Ψ is by setting all Boolean variables of some multi-valued variable to false. However, every instance α will set one of these Boolean variables to true so ρ cannot be consistent with α.

prime implicants of the Boolean encoding directly (which would lead to incorrect results), it reduces the problem of computing prime implicants of a multi-valued expression into one that requires only consistency testing of the Boolean encoding, which can be done using repeated calls to a SAT solver. The classical one-hot encoding is sound and complete for this purpose. Our treatment, however, is meant to be independent of the specific algorithm used to compute prime implicants. It would be needed, for example, when compiling the encoding into a tractable circuit and then computing prime implicants as done in [25, 5].

4.4 Encoding Decision Trees and Random Forests

Consider a decision tree, such as the one depicted in Figure 1. Each internal node in the tree represents a decision, which is either true or false. Each leaf is annotated with the predicted label. We can thus view a decision tree as a function whose inputs are all of the unique decisions that can be made in the tree, and whose output is the resulting label. Each leaf of the decision tree represents a simple term over the decisions made on the path to reach it, found by conjoining the appropriate literals. The Boolean function representing a particular class can then be found by simply disjoining the paths for all leaves of that class. That is, this Boolean function outputs true for all inputs that result in the corresponding class label, and false otherwise. We can also obtain this function for an ensemble of decision trees, such as a random forest. We first obtain the Boolean functions of each individual decision tree, and then aggregate them appropriately. For a random forest, we can use a simple majority gate whose inputs are the outputs of each decision tree; see also [1]. Finally, once we have the Boolean function of a classifier, we could apply a SAT or SMT solver to analyze it as proposed by [10, 16, 7]. We could also compile it into a tractable representation, such as an Ordered Binary Decision Diagram (OBDD), and then analyze it as proposed by [25, 24, 26, 23]. In the latter case, a representation such as an OBDD allows us to perform certain queries and transformation on a Boolean function efficiently, which facilitates the explanation and formal verification of the underlying machine learning classifier, as also shown more generally in [1].

5 Conclusion

We considered the encoding of input-output behavior of decision trees and random forests using Boolean expressions. Our focus has been on the suitability of encodings for computing prime implicants, which have recently played a central role in explaining the decisions of machine learning classifiers. Our findings have identified a particular encoding that is suitable for this purpose. Our encoding is based on a classical encoding that has been employed for the task of satisfiability but that can lead to incorrect results when computing prime implicants, which further emphasizes the merit of the investigation we conducted in this paper.

Ack. This work has been partially supported by grants from NSF IIS-1910317, ONR N00014-18-1-2561, DARPA N66001-17-2-4032 and a gift from JP Morgan.

References

1. Audemard, G., Koriche, F., Marquis, P.: On tractable XAI queries based on compiled representations. In: Proc. of KR'20 (2020), to appear
2. Bessiere, C., Hebrard, E., O'Sullivan, B.: Minimising decision tree size as combinatorial optimisation. In: CP. Lecture Notes in Computer Science, vol. 5732, pp. 173–187. Springer (2009)
3. Choi, A., Shi, W., Shih, A., Darwiche, A.: Compiling neural networks into tractable Boolean circuits. In: AAAI Spring Symposium on Verification of Neural Networks (VNN) (2019)
4. Crama, Y., Hammer, P.L.: Boolean Functions - Theory, Algorithms, and Applications, Encyclopedia of mathematics and its applications, vol. 142. Cambridge University Press (2011)
5. Darwiche, A., Hirth, A.: On the reasons behind decisions. In: Proceedings of the 24th European Conference on Artificial Intelligence (ECAI) (2020)
6. Ignatiev, A., Morgado, A., Marques-Silva, J.: Propositional abduction with implicit hitting sets. In: Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI). pp. 1327–1335 (2016)
7. Ignatiev, A., Narodytska, N., Marques-Silva, J.: Abduction-based explanations for machine learning models. In: Proceedings of the Thirty-Third Conference on Artificial Intelligence (AAAI). pp. 1511–1519 (2019)
8. Ignatiev, A., Narodytska, N., Marques-Silva, J.: On relating explanations and adversarial examples. In: Advances in Neural Information Processing Systems 32 (NeurIPS). pp. 15857–15867 (2019)
9. Ignatiev, A., Narodytska, N., Marques-Silva, J.: On validating, repairing and refining heuristic ML explanations. CoRR **abs/1907.02509** (2019)
10. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Computer Aided Verification CAV. pp. 97–117 (2017)
11. de Kleer, J.: A comparison of ATMS and CSP techniques. In: IJCAI. pp. 290–296. Morgan Kaufmann (1989)
12. Leofante, F., Narodytska, N., Pulina, L., Tacchella, A.: Automated verification of neural networks: Advances, challenges and perspectives. CoRR **abs/1805.09938** (2018)
13. McCluskey, E.J.: Minimization of boolean functions. The Bell System Technical Journal **35**(6), 1417–1444 (Nov 1956)
14. Miller, D.M., Thornton, M.A.: Multiple Valued Logic: Concepts and Representations, Synthesis lectures on digital circuits and systems, vol. 12. Morgan & Claypool Publishers (2008)
15. Narodytska, N., Ignatiev, A., Pereira, F., Marques-Silva, J.: Learning optimal decision trees with SAT. In: Lang, J. (ed.) Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI). pp. 1362–1368 (2018)
16. Narodytska, N., Kasiviswanathan, S.P., Ryzhyk, L., Sagiv, M., Walsh, T.: Verifying properties of binarized deep neural networks. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI) (2018)
17. Quine, W.V.: The problem of simplifying truth functions. The American Mathematical Monthly **59**(8), 521–531 (1952)
18. Quine, W.V.: On cores and prime implicants of truth functions. The American Mathematical Monthly **66**(9), 755–760 (1959)

19. Ramesh, A., Murray, N.V.: Computing prime implicants/implicates for regular logics. In: Proceedings of the 24th IEEE International Symposium on Multiple-Valued Logic (ISMVL). pp. 115–123 (1994)
20. Renooij, S.: Same-decision probability: Threshold robustness and application to explanation. In: Studeny, M., Kratochvil, V. (eds.) Proceedings of the International Conference on Probabilistic Graphical Models (PGM). Proceedings of Machine Learning Research, vol. 72, pp. 368–379. PMLR (2018)
21. Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: High-precision model-agnostic explanations. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI) (2018)
22. Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: High-precision model-agnostic explanations. In: AAAI. pp. 1527–1535. AAAI Press (2018)
23. Shi, W., Shih, A., Darwiche, A., Choi, A.: On tractable representations of binary neural networks. In: Proc. of KR'20 (2020), to appear
24. Shih, A., Choi, A., Darwiche, A.: Formal verification of bayesian network classifiers. In: PGM. Proceedings of Machine Learning Research, vol. 72, pp. 427–438. PMLR (2018)
25. Shih, A., Choi, A., Darwiche, A.: A symbolic approach to explaining bayesian network classifiers. In: IJCAI. pp. 5103–5111. ijcai.org (2018)
26. Shih, A., Darwiche, A., Choi, A.: Verifying binarized neural networks by angluin-style learning. In: SAT (2019)
27. Walsh, T.: SAT v CSP. In: CP. Lecture Notes in Computer Science, vol. 1894, pp. 441–456. Springer (2000)

A Proofs

Proof (of Proposition 1). For multi-valued term τ, the Boolean encoding τ_b is a negative term and consistent with Ψ by construction. Suppose now that ρ is a negative Boolean term that is consistent with Ψ. If ρ mentions a Boolean variable of multi-valued variable X, then ρ cannot mention all Boolean variables of X, otherwise ρ will be ruling out all possible values of X and hence inconsistent with Ψ. Hence, ρ encodes a literal over variable X when ρ mentions a Boolean variable for X. More generally, ρ encodes a term over multi-valued variables whose Boolean variables are mentioned in ρ. To prove the second part of the theorem, consider literals ℓ_1 and ℓ_2, which specify values S_1 and S_2 for variable X. The two literals are equivalent iff $S_1 = S_2$ iff $\bigwedge_{x_i \notin S_1} \bar{x}_i$ and $\bigwedge_{x_i \notin S_2} \bar{x}_i$ are equivalent. Moreover, $\ell_1 \models \ell_2$ iff $S_1 \subseteq S_2$ iff $\bigwedge_{x_i \notin S_1} \bar{x}_i \models \bigwedge_{x_i \notin S_2} \bar{x}_i$. Equivalence and subsumption relations are then preserved on literals, and on terms as well.

Proof (of Proposition 2). ($\Rightarrow$) Suppose $\rho \models_\Psi \Gamma$ and let α be a completion of ρ. If α is consistent with Ψ, then $\alpha \models \Gamma$ by Definition 1. If α is not consistent with Ψ, then $\alpha \models \neg\Psi$. Hence, $\rho \models \neg\Psi \vee \Gamma$. ($\Leftarrow$) Suppose $\rho \models \neg\Psi \vee \Gamma$ and let α be a completion of ρ that is consistent with Ψ. Then $\alpha \models \neg\Psi \vee \Gamma$ and, hence, $\alpha \wedge \Psi \models \Gamma$ and $\alpha \models \Gamma$. We then have $\rho \models_\Psi \Gamma$ by Definition 1.

Proof (of Proposition 3). ($\Rightarrow$) Suppose $\tau \models \Delta$. Then $\alpha \models \Delta$ for all completions α of τ. By Lemmas 1 and 2, $\alpha_b \models \Delta_b$ for all completions α_b of τ_b that are consistent with Ψ. Hence $\tau_b \models \neg\Psi \vee \Delta_b$. ($\Leftarrow$) Suppose $\tau_b \models \neg\Psi \vee \Delta_b$ and let

α_b be a completion of τ_b ($\alpha_b \models \neg\Psi \vee \Delta_b$). For each α_b consistent with Ψ, we have $\alpha_b \models \Psi$ and hence $\alpha_b \models \Delta_b$. By Lemmas 1 and 2, the completions α of τ correspond to these α_b (consistent with Ψ), leading to $\alpha \models \Delta$ and hence $\tau \models \Delta$.

Proof (of Proposition 4). ($\Rightarrow$) Suppose τ is a prime implicant of Δ. Then $\tau \models \Delta$. Moreover, $\tau_b \models (\Psi \Rightarrow \Delta_b)$ by Proposition 3 so τ_b is an implicant of $\Psi \Rightarrow \Delta_b$ (τ_b is negative and consistent with Ψ by construction). Suppose τ_b is not a prime implicant of $\Psi \Rightarrow \Delta_b$. Then $\rho \models (\Psi \Rightarrow \Delta_b)$ for a strict subset ρ of τ_b, which must be consistent with Ψ since $\tau_b \supset \rho$ is consistent with Ψ. Hence, ρ encodes a term $\tau^\star$ that is strictly weaker than term τ by Proposition 1. Moreover, $\tau^\star \models \Delta$ by Proposition 3 so τ is not a prime implicant of Δ, which is a contradiction. Therefore, τ_b is a prime implicant of $\Psi \Rightarrow \Delta_b$. ($\Leftarrow$) Suppose ρ is a prime implicant of $\Psi \Rightarrow \Delta_b$, negative and consistent with Ψ. Then ρ encodes a term τ by Proposition 1. Moreover, $\rho = \tau_b \models \Psi \Rightarrow \Delta_b$ so $\tau \models \Delta$ by Proposition 3. Hence, τ is an implicant of Δ. Suppose now that $\tau^\star \models \Delta$ for some term $\tau^\star$ that is strictly weaker than term τ. Then $\tau_b^\star \models \Psi \Rightarrow \Delta_b$ by Proposition 3. This means ρ is not a prime implicant of $\Psi \Rightarrow \Delta_b$ since $\tau_b^\star \subset \tau_b = \rho$ by Proposition 1, which is a contradiction. Hence, the term τ encoded by ρ is a prime implicant of Δ.