




Explaining SAT Benchmark Classifiers with Prime Implicants

Markus Iser   

Karlsruhe Institute of Technology (KIT), Institute of Theoretical Informatics, Algorithm Engineering, Germany

Abstract

Taxonomies for benchmark instances can be derived in several ways. For example, this can be based on their theoretical properties, or it can be based on their origin, e.g., a concrete application. Some algorithm selectors even generate instance classes based on a set of features in an unsupervised manner. We see a gap in the explainability of such approaches. In this paper, we present a SAT encoding for decision tree and random forest classifiers that we can use to generate prime implicants. We then apply this encoding to generate minimal explanations for two types of classifiers. The first type of classifier was trained to map SAT instances to their instance family. The second type of classifier was trained as an instance-specific SAT solver selector on a small portfolio. We show that the explainability of machine learning methods is useful for interpreting experiments as it provides valuable feedback to the algorithm developer.

2012 ACM Subject Classification Theory of computation → Logic and verification; Computing methodologies → Supervised learning by classification

Keywords and phrases Benchmarks, Explainable Algorithm Selection, Prime Implicants

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Models created by inductive learning algorithms are useful in practice. The explainability of such automatically generated models is the subject of current research projects [4, 11]. In particular, for many application scenarios where accountability concerns arise, it is imperative that what is learned can be explained [22].

Automated methods for generating explanations for machine-learned models include formalizing and encoding them in formal logic. The symbolic representation of what is learned enables the application of deductive methods for reasoning about what is learned. Deductive reasoning about inductively generated models is usually referred to by the term eXplainable AI (XAI) [3].

The term *explanation* is ambiguous. There is a plethora of possible formalizations leading to different definitions of explainability. Audemard et al. analyze the complexity of different types of XAI queries [5]. One of the explanatory queries they analyze is prime implicants.

Choi et al. present a coding of decision trees and random forests for computing prime implicants [10]. They report limitations in encoding cardinality constraints for multivalued variables.

Inductive learning methods have been used in various ways to solve SAT problems more efficiently. Cherif et al. successfully used reinforcement learning to control the switching of branching heuristics in their solver Kissat MAB [9]. Clause forgetting heuristics based on classification and regression using solver runtime data were presented by Soos et al. with their system Crystal Ball [23].

Prediction models have also been used to create algorithm selectors for portfolios of SAT solvers (cf. SATzilla [24]). In their approach known as instance-specific algorithm configuration (ISAC), Kadioglu et al. use unsupervised learning to create clusters in the



© Markus Iser;

licensed under Creative Commons License CC-BY 4.0

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

feature space of SAT instances [19]. For each cluster, a configurator optimizes a SAT solver for the instances in that cluster.

Prediction models for algorithm selection induce, in a sense, a taxonomy for SAT instances. This differs from the classes used in common practice, where instances are usually assigned to an instance family [13]. The instance family may for example describe a category of SAT applications (such as *hardware verification*), or refer to a specific class of instances that is of theoretical interest (such as *pigeon hole*).

Elffers et al. analyze SAT solver configurations for instance classes of theoretical interest [12]. Audemard and Simon devise a taxonomy which is purely based on solver runtime parameters and deduce class specific solver configurations [6].

Given the circumstances sketched so far, we conclude that explanations of prediction models for SAT benchmarks can provide valuable feedback to the algorithm engineer about what was learned. Such explanations allow the algorithm engineer to reason about the induced instance taxonomy, and to improve their algorithm or configuration choices.

In this paper, we present a simple encoding of decision trees classifiers and random forests as a monotone combinatorial circuit. We show how prime implicants can be computed for the resulting formula using an off-the-shelf incremental SAT solver. Moreover, we present a tool which can encode decision tree classifiers generated with the Python package `scikit-learn` [21] into propositional formulas. Our tool uses the SAT solver `CaDiCaL` [8] via its `IPASIR` [7] interface to generate prime implicants. The prime implicants are then decoded with respect to the represented case-distinctions in the features space. We evaluate the tool on a set of 26794 SAT instances, with a large set of features including their instance families.

The document is structured as follows. In Section 2, we present a formalization of decision tree classifiers and random forest classifiers as they are implemented in the Python package `scikit-learn`. The section concludes with the required fundamentals on propositional logic, incremental SAT solvers and prime implicants. In Section 3, we present a propositional encoding of the previously formalized classifiers and an algorithm to compute prime implicants for them. We present our implementation in Section 4, where we also evaluate our approach on a classifier which was trained to associate SAT instances families for a set of 26794 SAT instances based on a set of instance features which we also describe in that section. We also evaluate our approach on a classifier which was trained to select SAT solvers from small portfolios of solvers drawn from SAT competition 2020. We conclude with Section 5.

2 Preliminaries

In the following sections, we formally describe the data structures created by *decision tree* (Section 2.1.1) and *random forest* (Section 2.1.2) classifiers which realize the learned prediction function as they are implemented in the Python package `scikit-learn` [21]. In Section 2.2, we introduce some basic notions of propositional logic in particular regarding prime implicants. The formalizations will later serve as a fundament to the propositional encoding of the prediction models under consideration (Section 3).

2.1 Classification

The *classification problem* under consideration is specified as follows. Given a set of training samples $T \subset \mathbb{R}^n$, a set of classes K , and a functional ground-truth relation $G \subset T \times K$, devise a prediction function $c : \mathbb{R}^n \rightarrow K$ which maximizes the cardinality of correctly classified training samples $|\{t \in T \mid (t, c(t)) \in G\}|$. The challenge is to devise a prediction function which generalizes to yet unseen samples of the feature space. In theory, decision trees can

grow until they classify each training sample correctly, i.e., the classification result equals ground truth. In practice however, pruning techniques such as maximum depth or minimum leaf size are used in order to improve the generalization of a classifier to yet unseen samples. An introduction to methods and challenges of classification tasks and supervised learning in general can be found in Hastie et al. [14]. The following formalization depends on the concrete implementations `tree.DecisionTreeClassifier` and `ensemble.RandomForestClassifier` in the Python package `scikit-learn` [21].

2.1.1 Decision Tree Classifiers

Let a classification instance (T, K, G) and its solution in form of a *decision tree* \mathcal{D} be given. A decision tree $\mathcal{D} = (V, E, f, t)$ is specified by a *binary tree* with nodes V and edges $E \subset V \times V$. The nodes are partitioned into *inner nodes* $V^I := \{v \in V \mid \exists x, (v, x) \in E\}$, and *leaf nodes* $V^L = V \setminus V^I$. The *root node* r is the special inner node with $\nexists x, (x, r) \in E$. The set of edges E is partitioned into *positive edges* E^+ and *negative edges* E^- such that each inner node $v \in V^I$ has exactly one *positive successor* $\text{hi}(v) \in V$ with $(v, \text{hi}(v)) \in E^+$ and exactly one *negative successor* $\text{lo}(v) \in V$ with $(v, \text{lo}(v)) \in E^-$. Associated with each inner node $v \in V^I$ is a feature index $f : V^I \rightarrow \{1, 2, \dots, n\}$ and a threshold $t : V^I \rightarrow \mathbb{R}$.

Given a set of samples $S \subseteq \mathbb{R}^n$, the decision tree \mathcal{D} induces a partitioning $P_S : V \rightarrow 2^S$ of samples over nodes in V . Starting with $P_S(r) = S$ for root node r , the partitioning is recursively defined as in Equation 1.

$$P_S(v) = \begin{cases} S & \text{iff } v \text{ is the root node} \\ \{(s_0, \dots, s_n)^\top \in P_S(x) \mid s_{f(x)} \leq t(x)\} & \text{iff } v = \text{hi}(x) \\ \{(s_0, \dots, s_n)^\top \in P_S(x) \mid s_{f(x)} > t(x)\} & \text{iff } v = \text{lo}(x) \end{cases} \quad (1)$$

The partitioning P_T of the set of training samples T together with ground truth G induces for each node v a probability distribution p_v over classes $k \in K$ as in Equation 2.

$$p_v(k) = \frac{|\{s \in P_T(v) \mid (s, k) \in G\}|}{|P_T(v)|} \quad (2)$$

The decision tree \mathcal{D} specifies a classification function $c(s) : \mathbb{R}^n \rightarrow K$. Given a sample $s \in \mathbb{R}^n$, there exists exactly one leaf node $v \in V^L$ such that $s \in P_{\{s\}}(v)$. The classification result for sample s is the one with the highest probability in that node, i.e., $c(s) = \arg \max_{k \in K} p_v(k)$.

2.1.2 Random Forest Classifiers

In the context of machine learning, *bagging* is the idea to combine several weak learners to one stronger learner by using some kind of voting to determine one collective prediction [14]. A *random forest* $\mathcal{R}^d = \{(T_i, \mathcal{D}_i) \mid 1 \leq i \leq d\}$ combines a set of d decision trees. Each decision tree \mathcal{D}_i is independently trained on randomly selected subsets of the training samples $T_i \subset T$.

Let a classification problem (T, K, G) and a random forest $\mathcal{R}^d = \{(T_i, \mathcal{D}_i) \mid 1 \leq i \leq d\}$ be given. For each decision tree $\mathcal{D}_i = (V_i, E_i, f_i, t_i)$, the training samples T_i induce a probability distribution $p_v(k)$ for nodes $v \in V_i$ (cf. Section 2.1.1). Given a sample $s \in \mathbb{R}^n$, for each decision tree \mathcal{D}_i there exists exactly one leaf node $v_i \in V_i^L$ such that $s \in P_{\{s\}}(v_i)$. The collective class probabilities $p'(k)$ for a sample s are determined by averaging over the class probabilities in each tree as in Equation 3. The classification result for sample s is then given by $c'(s) = \arg \max_{k \in K} p'(k)$.

$$p'(k) = \frac{1}{d} \sum_{i=1}^d p_{v_i}(k) \quad (3)$$

2.2 Prime Implicants

In this document, propositional formulas are given in conjunctive normal form (CNF). A propositional *formula* F is defined over a finite set of Boolean variables V . Each formula is a conjunction of clauses, a clause is a disjunction of literals and a literal is either a variable or its negation. A set of (non-contradictory) literals over V is a *model* M of a formula F , iff its intersection with any clause in F is non-empty. A model is *complete* iff $|M| = |V|$ and *partial* otherwise. Each model M of a formula F is also an implicant of F , denoted by $M \models F$. An implicant M is a prime implicant of F iff $\nexists M' \subset M, M' \models F$.

Prime implicants for CNF formulas can efficiently be calculated through eager minimization of a model with an incremental SAT solver [18, 15]. If the CNF formula encodes a *monotonic* combinatorial circuit, it is sufficient to minimize the model for the input variables of that circuit, as a complete assignment can later be deduced from the minimized input assignment [15].

3 Approach

In the following we describe CNF encodings for decision tree classifiers (Section 3.1) and random forests (Section 3.2). The encodings resemble monotonic circuits which constrain feature values at their inputs. In Section 3.3, we outline an algorithm to efficiently compute prime implicants for these formulas. In Section 3.3.2, we show how to decode these prime implicants in order to map them to a set of case distinctions in the feature space.

3.1 Encoding Decision Tree Classifiers

Given a classification instance (T, K, G) and a corresponding decision tree $\mathcal{D} = (V, E, f, t)$, we create its propositional encoding as follows.

3.1.1 Variables

We introduce three sets of Boolean variables (α, β , and γ), which together form the set of variables V over which our encoding F is defined. For each class $k \in K$, we introduce a *class variable* $\alpha(k)$ for indicating the classification result. For each inner node $v \in V^I$, we introduce two *node variables* $\beta^+(v)$ and $\beta^-(v)$, with $\beta^+(v)$ denoting its successor $\text{hi}(v)$, and $\beta^-(v)$ denoting its successor $\text{lo}(v)$. For each feature f , we construct the auxiliary set of thresholds σ_f as specified by Equation 4. From σ_f we construct the auxiliary set of threshold intervals τ_f as shown in Equation 5.

$$\sigma_f := \{t(v) \mid f(v) = f, v \in V\} \cup \{-\infty, \infty\} \quad (4)$$

$$\tau_f := \{(t_0, t_1] \mid t_0, t_1 \in \sigma_f \wedge t_0 < t_1 \wedge \nexists t' \in \sigma_f, t_0 < t' < t_1\} \quad (5)$$

For each feature f and each threshold interval $z \in \tau_f$, we introduce the Boolean variable $\gamma(f, z)$. Variables of type $\gamma(f, z)$ indicate whether the respective interval is excluded by the decision tree. This type of value encoding has previously been proposed [10].

3.1.2 Clauses

For each class $k \in K$, we determine the set of leaf nodes $V_k^L \subseteq V^L$ in which the classifier outputs k . Then we encode the *class constraint* for each class k as depicted in Encoding 3.1.

► **Encoding 3.1** (Class Constraints).

$$\forall k \in K, \alpha(k) \rightarrow \bigvee_{v \in V_k^L} \beta^+(v)$$

For each inner node $v \in V^I$, we encode the *node constraints* as shown in Encoding 3.2.

► **Encoding 3.2** (Node Constraints).

$$\forall v \in V^I, \beta^+(\text{hi}(v)) \rightarrow \beta^+(v)$$

$$\forall v \in V^I, \beta^-(\text{hi}(v)) \rightarrow \beta^+(v)$$

$$\forall v \in V^I, \beta^+(\text{lo}(v)) \rightarrow \beta^-(v)$$

$$\forall v \in V^I, \beta^-(\text{lo}(v)) \rightarrow \beta^-(v)$$

For each inner node $v \in V^I$, we encode its *interval constraints* as follows. We split the set of threshold variables $\tau_{f(v)}$ and into two auxiliary sets $\tau^+(v)$ and $\tau^-(v)$ which are defined as follows.

$$\tau^+(v) := \{(t_0, t_1] \in \tau_{f(v)} \mid t_1 \leq t(v)\}$$

$$\tau^-(v) := \{(t_0, t_1] \in \tau_{f(v)} \mid t_0 > t(v)\}$$

Then we encode the *interval constraints* as in Encoding 3.3.

► **Encoding 3.3** (Interval Constraints).

$$\forall v \in V^I, \bigwedge_{\tau \in \tau^-(v)} \beta^+(v) \rightarrow \gamma(v, \tau)$$

$$\forall v \in V^I, \bigwedge_{\tau \in \tau^+(v)} \beta^-(v) \rightarrow \gamma(v, \tau)$$

3.2 Encoding Random Forest Classifiers

Let a random forest $\mathcal{R}^d = \{(T_1, \mathcal{D}_1), \dots, (T_d, \mathcal{D}_d)\}$ be given. Each decision tree \mathcal{D}_i was trained with the classification instance (T_i, K, G) and is given by $\mathcal{D}_i = (V_i, E_i, f_i, t_i)$.

The encoding for each decision tree is similar to the encoding described in Section 3.1. In particular, the node constraints given by Encoding 3.2 are just the same. But as in a random forest, classes are not simply determined by a singular leaf nodes, we need a different encoding for class constraints (cf. Section 3.2.2). Moreover, we need to adapt the interval encoding for reasons which we will explain in the following section.

3.2.1 Encoding Interval Constraints for Random Forests

As thresholds are chosen independently the size of the threshold set $\xi = |\{t(v_i) \mid f(v_i) = f, v_i \in V_i, 1 \leq i \leq d\}|$ can be very large for a given feature f . In preliminary experiments with 100 decision trees ξ could go into the thousands. The number of interval clauses generated when using Encoding 3.3 is in $\mathcal{O}(\xi^2)$. We therefore use a new encoding for interval constraints

202 which has a higher *constant* size overhead in terms of variables as well as clauses but scales
203 with $\mathcal{O}(\xi)$.

204 For each feature f with intervals τ_f (defined in Equation 5), we introduce two new sets
205 of variables $\epsilon^+(f, z)$ and $\epsilon^-(f, z)$ with $z \in \tau_f$. The interval order of the $z_i \in \tau_f$ also implies
206 an ordering of the variables $\epsilon^+(f, z_i)$ and $\epsilon^-(f, z_i)$. We first encode for each feature f the
207 *connect constraints*, as depicted in Encoding 3.4.

► **Encoding 3.4** (Interval Connect Constraints).

$$208 \quad \forall f \in [1, \dots, n], \forall z_i \in \tau_f, \epsilon^-(f, z_i) \rightarrow \gamma(f, z_i)$$

$$209 \quad \forall f \in [1, \dots, n], \forall z_i \in \tau_f, \epsilon^+(f, z_i) \rightarrow \gamma(f, z_i)$$

211 We further introduce the following *order constraints*.

► **Encoding 3.5** (Order Constraints).

$$212 \quad \forall f \in [1, \dots, n], z_i \in \tau_f, \epsilon^+(f, z_1) \rightarrow \dots \rightarrow \epsilon^+(f, z_{|\tau_f|})$$

$$213 \quad \forall f \in [1, \dots, n], z_i \in \tau_f, \epsilon^-(f, z_1) \leftarrow \dots \leftarrow \epsilon^-(f, z_{|\tau_f|})$$

215 Then the new *interval constraints* can be encoded for each leaf nodes in the forest as in
216 Encoding 3.6.

► **Encoding 3.6** (Interval Constraints).

$$217 \quad \forall v \in V^I, \beta^+(v) \rightarrow \epsilon^+(v, t(v))$$

$$218 \quad \forall v \in V^I, \beta^-(v) \rightarrow \epsilon^-(v, t(v))$$

220 3.2.2 Encoding Class Constraints for Random Forests

221 Recapitulate how classification works in the random forests described in Section 2.1.2. Given
222 a sample s , for each decision tree \mathcal{D}_i with leaf nodes V_i^L , there is exactly one leaf node
223 $v_i \in V_i^L$ such that $s \in P_{\{s\}}(v_i)$. Given the tuple of leaf nodes (v_1, \dots, v_d) , i.e., one for each
224 tree, the class of s is determined by the maximum average class probability over those leaf
225 nodes as depicted in Equation 3.

226 One way to encode this, would be to encode an arithmetic circuit.¹ For the implementation,
227 which we present in this paper, we took a simpler approach. Consider again the tuples
228 of leaf nodes $L := V_1^L \times \dots \times V_d^L$. To each tuple in $t \in L$, we can assign a class in K by
229 determining the maximum average class probability for the leaf nodes. But the size of L
230 grows exponentially with the number of decision trees in the forest. Fortunately, only a
231 fraction of tuples in L is actually a valid combination of leaf nodes, i.e., they also satisfy the
232 node and interval constraints.

¹ In this circuit, for each decision tree in the forest, we would encode a list of bit-vectors representing the probabilities of each class, which are encoded to be equivalent to the known class probabilities – dependent on the activated leaf node. For each class, we would further encode an adder circuit to calculate the sum of each classes probabilities over all trees in the forest. On top of that, we would then encode a circuit which ensures that the sum of probabilities of the class to explain is the maximum of all classes.

3.2.2.1 Determining Valid Leaf Tuples

In order to determine the set of valid tuples $L' \subseteq L$, we encode a small auxiliary SAT problem. We use the encoding described in the previous section and *validity constraints* as depicted in Figure 3.7. This ensures that the tuples allow for at least one interval per feature.

► **Encoding 3.7** (Validity Constraints).

$$\forall f \in [1, \dots, n], \bigvee_{z_i \in \tau_f} \neg \gamma(f, z_i)$$

Moreover, we encode an *exactly-one constraint* for the leaf nodes of each tree. The constraint consists of an at-least-one clause for the leaf nodes of each tree, and a simple pair-wise encoding of an at-most-one constraint. We then enumerate all solutions to variables in $\{\beta^+(v) \mid v \in V_i^L, 1 \leq i \leq d\}$ with a SAT solver to generate the set of valid tuples L' .

3.2.2.2 Connection Classes to Tuples

For each valid tuple in $t_i \in L'$, we introduce a new variable $\delta(t_i)$ and encode the *tuple constraint* as depicted in Encoding 3.8.

► **Encoding 3.8** (Tuple Constraints).

$$\forall t_i \in L', \bigwedge_{v \in t_i} \delta(t_i) \rightarrow \beta^+(v)$$

For each tuple t_i , we determine its class $k_i \in K$ as shown in Equation 3 and encode the class constraints as depicted in Encoding 3.9.

► **Encoding 3.9** (Class Constraints).

$$\forall k_i \in K, \alpha(k_i) \rightarrow \bigvee_{t_i \in L'} \delta(t_i)$$

3.3 Explaining Classes with Prime Implicants

Given a decision tree or random forest classifier, and a set of classes to explain $X \subseteq K$, we generate the clauses according to the encoding presented in this section and add the *explanation clause* $\{\alpha(k) \mid k \in X\}$. In most cases, the explanation clause might be a unit clause, since we often want shortest explanations for a single class, or we might want to solve the problem repeatedly once for each class in K .

3.3.1 Minimizing Positive Inputs

The presented encodings of decision trees and random forests resemble a monotonic combinatorial circuit. Thus, a minimal assignment to the input variables induces a minimal assignment to all the variables in the circuit. The circuit is rooted in the explanation clause, and the inputs of the circuit are the interval variables. In fact the clauses encode mostly binary implication chains. All implied variables are implied in their positive polarity. For this reason, we can compute prime implicants by minimizing the number of positively assigned interval variables.

Algorithm 1 outlines the procedure. The algorithm receives as input the formula F and the set of input variables I , i.e., the set of all interval variables. It then returns the set of all

XX:8 Explaining SAT Benchmark Classifiers with Prime Implicants

268 minimal assignments P under projection to the input variables. In the outer loop (line 2),
269 we subsequently generate models for F (lines 1 and 15). In the inner loop (line 3), we eagerly
270 minimize those models, by successive addition of clauses (lines 5, 8 and 11) which exclude the
271 current minimum of positively assigned input variables $v \in I$ (line 6). The minimization is
272 done in a strictly eager fashion, as we add assumption literals for already negatively assigned
273 input variables (lines 4, 10, and 12). If the assignment can not be further minimized, we
274 record the prime implicant (lines 13 and 14) and continue to find a new model to minimize
275 (line 15).

■ Algorithm 1 Incremental Computation of Prime Implicants

Input: CNF Formula: F
Input: Input Variables: I
Output: Prime Implicants: P

```
1 (sat, model)  $\leftarrow$  solve( $F, \emptyset$ )
2 while sat do
3   while sat do
4     assumptions  $\leftarrow \emptyset$ 
5     minim  $\leftarrow \emptyset$ 
6     for  $v \in I$  do
7       if  $v \in \text{model}$  then
8          $\perp$  minim  $\leftarrow \text{minim} \cup \{-v\}$ 
9       else
10         $\perp$  assumptions  $\leftarrow \text{assumptions} \cup \{-v\}$ 
11       $F \leftarrow F \cup \text{minim}$ 
12      (sat, model)  $\leftarrow$  solve( $F, \text{assumptions}$ )
13      if not sat then
14         $\perp$   $P \leftarrow P \cup \{-v \mid v \in \text{minim}\}$ 
15    (sat, model)  $\leftarrow$  solve( $F, \emptyset$ )
16 return  $P$ 
```

276 3.3.2 Decoding Prime Implicants

277 We can decode each minimal assignment $p \in P$ into a minimal set of case distinctions in the
278 feature space to explain the classes $k \in X$. For each feature f , we iterate the assignment
279 to interval variables $\gamma(f, z_i)$ according to the ordering induced by the intervals $z_i \in \tau_f$. For
280 each adjacent pair $a = \gamma(f, z_i)$ and $b = \gamma(f, z_{i+1})$, we decode case distinctions D as follows.
281 iff a is positive (negative) in p while at the same time b is negative (positive) in p . If a is
282 true in p and b is false in p , we decode $D := f > z_i$. If a is false in p and b is true in p , we
283 decode $D := f \leq z_i$. If both a and b are either true or false, we decode nothing.

4 Evaluation

4.1 Implementation and Dataset

An implementation of our approach can be found on GitHub.² It consists of a `Python` module written in `C++` which computes prime implicants (cf. Section 3.3) using the incremental SAT solver `CaDiCaL` by Armin Biere [8]. The encoding is implemented in `Python` for the decision tree classifier implementation in the `scikit-learn` package [21].

4.1.1 Dataset

Our implementation accesses data via our `gbd-tools` package³ which we originally presented in [17]. The three datasets which we used in our evaluations can be obtained at <https://gbd.iti.kit.edu/> and are described in the following. The code which we used to calculate the features described in Sections 4.1.1.2 and 4.1.1.3 can be found on GitHub.⁴

4.1.1.1 Meta Features

We collected a set of meta features in a huge database of more than 26k benchmark instances (mainly from SAT competitions since 2002). These meta features include the `sat/unsat` result (if known) and the instance family. The instance families have been manually deduced from the documents in SAT competition proceedings or SAT competition presentation slides.

4.1.1.2 Base Features

The features in the *base* dataset do not resemble but share some noteworthy coverage with the features used in [24] and [1].

Amounts This set of features includes the number of `clauses`, `variables`, the numbers of clauses of sizes 1 to 9, the number of `horn clauses`, `inverse horn clauses`, `positive clauses`, and `negative clauses`. The set consists of the following sub groups of features.

Distribution over Horn Clauses For this set of features, we count for each variable its number of occurrences in horn clauses. The counts are represented by five features, their `mean`, `variance`, `minimum`, `maximum` and `entropy`. We add another five features for the count of variable occurrences in inverse horn clauses.

Balance of Literal Polarities Here we report on two distributions in terms of their `mean`, `variance`, `minimum`, `maximum` and `entropy`. The first distribution captures for each variable the fraction of its number of positive occurrences in clauses divided by the number of negative occurrences in clauses. The second distribution captures for each clause the number of positive literals divided by the number of negative literals.

Distribution of Node Degrees Degree distributions by `mean`, `variance`, `minimum`, `maximum` and `entropy` for nodes in the variable interaction graph, nodes in the clause graph, variable nodes in the variable-clause graph, clause nodes in the variable-clause graph.

² <https://github.com/Udopia/pi-explanations>

³ <https://pypi.org/project/gbd-tools/>

⁴ <https://github.com/sat-clique/cnftools>

318 4.1.1.3 Gate Features

319 The features in the *gate* dataset were calculated for a recovered hierarchical gate structure
 320 which was generated by a gate extraction algorithm for cnf formulas [16, 15].

321 **Amounts** This sub group of gate features covers the total number of gates, the number of
 322 root variables, the number of input variables, the number of generically recognized gates, the
 323 number of monotonically nested gates, the number of non-monotonically nested and-gates,
 324 the number of non-monotonically nested or-gates, the number of non-monotonically nested
 325 trivial equivalence gates, the number of non-monotonically nested equiv- or xor-gates, as well
 326 as the number of non-monotonically nested full gate, i.e., max-term encodings, with more
 327 than two inputs.

328 **Distribution over Levels** For each gate type for which we recorded their amounts in the
 329 previous paragraph, we also determine their levels in the decoded hierarchical gate structure,
 330 for each type we report on the distribution of levels by their **mean**, **variance**, **minimum**,
 331 **maximum** and **entropy**.

332 4.2 Explaining Classification Results

333 In the following, we present some initial results for classifiers trained on the given dataset.
 334 We used the implementations `DecisionTreeClassifier` and `RandomForestClassifier` in
 335 the Python package `scikit-learn`. Both were used with their default settings and a seed
 336 of zero. For the `RandomForestClassifier` we additionally reduced the number of decision
 337 trees to 2 (default is 100).

338 4.2.1 Instance Family Classifier

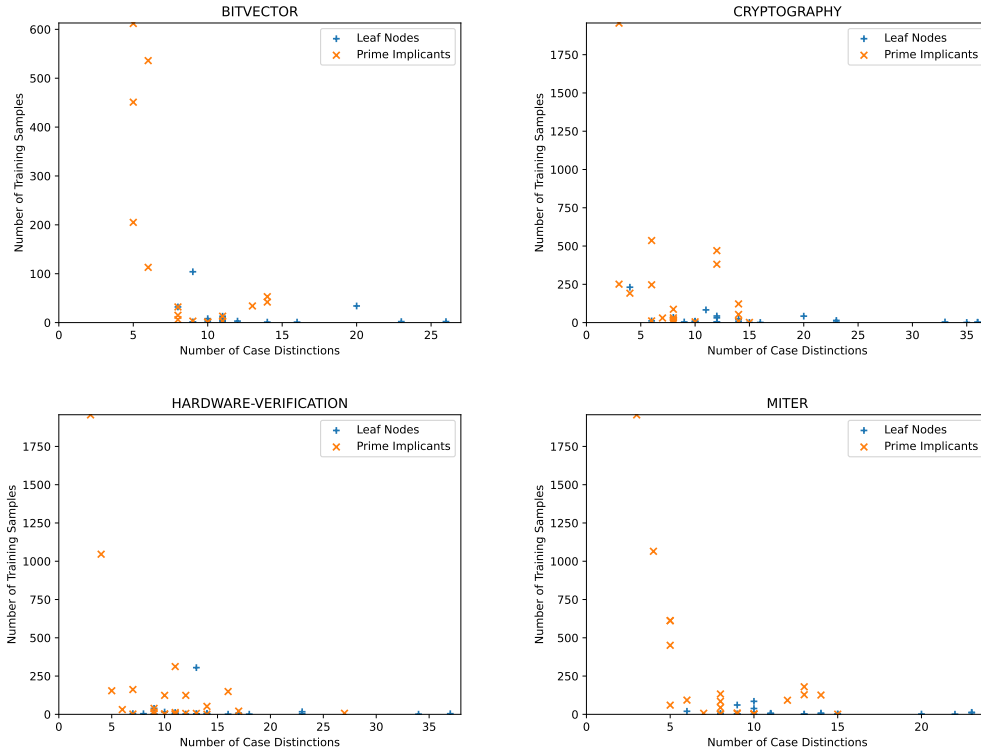
339 We trained a decision tree classifier to predict instance families for instances of SAT Com-
 340 petitions 2002 - 2021. Benchmarks of SAT Competitions 2004 and 2005 were missing. We
 341 excluded instances of random instance families and the agile tracks. That left us with a
 342 total 6647 instances of 124 distinct instance families. We train a decision tree classifier to
 343 predict the family of an instance given the instance features described in Sections 4.1.1.2
 344 and 4.1.1.3. The accuracy of the classifier in a cross-validation experiment leaving out 20%
 345 of the instances as a testing set is around 94%.

346 Figure 1 depicts the explanation for four exemplary families in terms of the leaf nodes
 347 (blue) and the prime implicants (orange). The x-axis shows the number of case distinctions,
 348 which corresponds to the node depths in terms of the leaf nodes and the number of decoded
 349 case distinctions in case of the prime implicants. The y-axis shows the number of training
 350 samples selected by the respective case distinctions. The figure shows that with prime
 351 implicants we generally need less case distinctions to explain a classification result. The figure
 352 also shows that prime implicants generally cover more samples than singular leaf nodes.

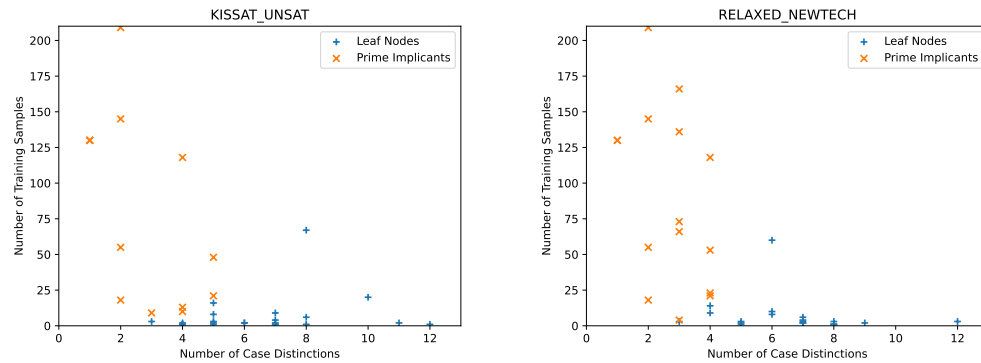
353 4.2.2 Small Portfolios Classifier

354 Frolejks et al. report on best small portfolios drawn from solvers of SAT Competition 2020 [13].
 355 The best performing 2-portfolio in terms of its VBS score for the 400 instances in the SAT
 356 Competition 2020 benchmarks is Kissat unsat and Relaxed newTech.

357 We train a decision tree classifier to predict the fastest solver for an instance given the
 358 instance features described in Sections 4.1.1.2 and 4.1.1.3. The accuracy of the classifier in a

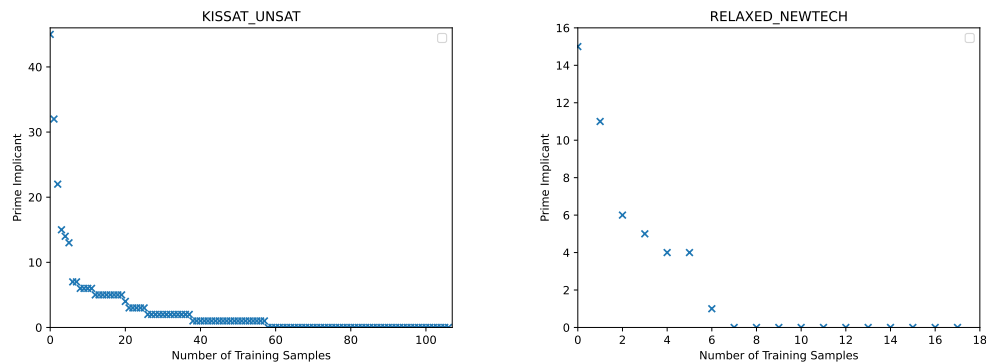


■ **Figure 1** Number of Case Distinctions in Feature Space and Number of Covered Training Samples for *Prime Implicants* (orange) vs. *Leaf Nodes* (blue). The classifier was trained to separate instances by their family. The plots show the explanations for the four exemplary families *bitvector*, *cryptography*, *hardware-verification* and *miter*.



■ **Figure 2** Number of Case Distinctions in Feature Space and Number of Covered Training Samples for *Prime Implicants* (orange) vs. *Leaf Nodes* (blue). The classifier was trained to separate instances for *Kissat Unsat* from instances for *Relaxed NewTech* (by solver speed).

XX:12 Explaining SAT Benchmark Classifiers with Prime Implicants



■ **Figure 3** Number of Covered Training Samples for all *Prime Implicants*. The classifier was trained to separate instances for *Kissat Unsat* from instances for *Relaxed NewTech* (by solver speed).

359 cross-validation experiment leaving out 20% of the instances as a testing set is around 68%.
360 Figure 2 shows that also in this case, with prime implicants we need less case distinctions
361 while at the same time covering more samples when using prime implicants as opposed to
362 leaf nodes.

363 We also train a random forest classifier consisting of only 2 decision trees for the same
364 data. For this classifier, the accuracy in a cross-validation experiment leaving out 20% of the
365 instances as a testing set raises to around 76%. Figure 3 shows the number of training samples
366 for each prime implicant of the two classes. We observe a large number of prime implicants
367 with zero training samples. This is due to “spurious” valid tuples of leaf nodes emerging in
368 the random forest. The total number of leaf node combinations in this forest is 1518. The
369 number of valid tuples determined by our encoding algorithm is 125 (cf. Section 3.2.2.1).

370 We also train a random forest classifier consisting of only 3 decision trees for the same
371 data. For this classifier, the accuracy in a cross-validation experiment leaving out 20% of
372 the instances as a testing set raises to around 78%. In this scenario, we have a total of
373 66792 leaf node combinations, of which only 11082 are valid, i.e., they are reachable by some
374 combination of values in the feature space (cf. Section 3.2.2.1).

375 5 Conclusion

376 Our evaluation shows that with prime implicants we indeed need less case distinctions while
377 at the same time covering more samples for explaining classification results. Calculating
378 prime implicants has the advantage that we can extract useful information such as the
379 smallest prime implicant covering most of the samples. This can for example be used to only
380 calculate a small subset of the features to use only a small set of case distinction for selecting
381 solver configurations. Moreover, specific prime implicants can become subject to research,
382 e.g., to reason about the imposed taxonomy or for evaluating biases in benchmarks.

383 One problem for our approach is the growing number of interval thresholds in random
384 forests. In initial experiments with the random forest encoding, we see a large number of
385 prime implicants which is not represented by a sample in the training set, i.e, we discover
386 a large number of valid leaf tuples in the random forest which is actually empty (in terms
387 of training samples) but would still produce a classification result for yet unseen samples
388 matching the imposed case distinctions. Our initial implementation of the random forest
389 encoding described in this paper seemed only feasible for random forest of up to 3 desicion

trees. One reason for this can also be seen in the large number of “spurious” valid leaf *tuples*.

We assume that the number of valid leaf tuples could be reduced by reducing the allowed feature thresholds, e.g., through rounding, in the implementation of the classifier. This would reduce the number of intervals for each feature and could thus positively impact the number of valid nodes due to less implicants with zero training samples. This could in turn also positively impact the performance of our approach and lead to more expressive implicants for random forests. We also intend to implement and analyze the encoding for random forests which we described in the footnote of Section 3.2.

In future work, we want to analyze benchmark taxonomies in greater detail under inclusion of new features, e.g., graph modularity or clustering based features [2, 20]. We will also study feature importance measures and generalization power based on prime implicants. Moreover, we will evaluate portfolio performance with solver selection based on only a few prime implicants.

References

- 1 Enrique Matos Alfonso and Norbert Manthey. New CNF features and formula classification. In Daniel Le Berre, editor, *POS-14. Fifth Pragmatics of SAT workshop, a workshop of the SAT 2014 conference, part of FLoC 2014 during the Vienna Summer of Logic, July 13, 2014, Vienna, Austria*, volume 27 of *EPiC Series in Computing*, pages 57–71. EasyChair, 2014. URL: <https://doi.org/10.29007/b8t1>.
- 2 Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, Jordi Levy, and Laurent Simon. Community structure in industrial SAT instances. *J. Artif. Intell. Res.*, 66:443–472, 2019. doi:10.1613/jair.1.11741.
- 3 Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *CoRR*, abs/1910.10045, 2019.
- 4 Gilles Audemard, Steve Bellart, Louenas Bounia, Frédéric Koriche, Jean-Marie Lagniez, and Pierre Marquis. On the computational intelligibility of boolean classifiers. In Meghyn Bienvenu, Gerhard Lakemeyer, and Esra Erdem, editors, *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, pages 74–86, 2021.
- 5 Gilles Audemard, Frédéric Koriche, and Pierre Marquis. On tractable XAI queries based on compiled representations. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pages 838–849, 2020. URL: <https://doi.org/10.24963/kr.2020/86>.
- 6 Gilles Audemard and Laurent Simon. Extreme cases in SAT problems. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 87–103. Springer, 2016.
- 7 Tomás Balyo, Armin Biere, Markus Iser, and Carsten Sinz. SAT race 2015. *Artif. Intell.*, 241:45–65, 2016. URL: <https://doi.org/10.1016/j.artint.2016.08.007>.
- 8 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.
- 9 Mohamed Sami Cherif, Djamel Habet, and Cyril Terrioux. Combining VSIDS and CHB using restarts in SAT. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual*

- Conference), October 25-29, 2021, volume 210 of *LIPICs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: <https://doi.org/10.4230/LIPICs.CP.2021.20>.
- 10 Arthur Choi, Andy Shih, Anchal Goyanka, and Adnan Darwiche. On symbolically encoding the behavior of random forests. *CoRR*, abs/2007.01493, 2020. URL: <https://arxiv.org/abs/2007.01493>.
- 11 Adnan Darwiche and Auguste Hirth. On the reasons behind decisions. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 712–720. IOS Press, 2020.
- 12 Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, Jakob Nordström, and Laurent Simon. Seeking practical CDCL insights from theoretical SAT benchmarks. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1300–1308. ijcai.org, 2018. URL: <https://doi.org/10.24963/ijcai.2018/181>.
- 13 Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. SAT competition 2020. *Artif. Intell.*, 301:103572, 2021. URL: <https://doi.org/10.1016/j.artint.2021.103572>.
- 14 Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer Series in Statistics. Springer, 2009. URL: <https://doi.org/10.1007/978-0-387-84858-7>.
- 15 Markus Iser. *Recognition and Exploitation of Gate Structure in SAT Solving*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2020. URL: <https://nbn-resolving.org/urn:nbn:de:101:1-2020042904595660732648>.
- 16 Markus Iser, Norbert Manthey, and Carsten Sinz. Recognition of nested gates in CNF formulas. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2015. URL: https://doi.org/10.1007/978-3-319-24318-4_19.
- 17 Markus Iser and Carsten Sinz. A problem meta-data library for research in SAT. In Daniel Le Berre and Matti Järvisalo, editors, *Proceedings of Pragmatics of SAT 2015, Austin, Texas, USA, September 23, 2015 / Pragmatics of SAT 2018, Oxford, UK, July 7, 2018*, volume 59 of *EPiC Series in Computing*, pages 144–152. EasyChair, 2018. URL: <https://doi.org/10.29007/gdbb>.
- 18 Markus Iser, Carsten Sinz, and Mana Taghdiri. Minimizing models for tseitin-encoded SAT instances. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 224–232. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-39071-5_17.
- 19 Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC - instance-specific algorithm configuration. In Helder Coelho, Rudi Studer, and Michael J. Wooldridge, editors, *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 751–756. IOS Press, 2010.
- 20 Chunxiao Li, Jonathan Chung, Soham Mukherjee, Marc Vinyals, Noah Fleming, Antonina Kolokolova, Alice Mu, and Vijay Ganesh. On the hierarchical community structure of practical boolean formulas. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 359–376. Springer, 2021. URL: https://doi.org/10.1007/978-3-030-80223-3_25.

- 492 21 Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion,
493 Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-
494 learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830,
495 2011.
- 496 22 Chris Percy, Simo Dragicevic, Sanjoy Sarkar, and Artur S. d’Avila Garcez. Accountability
497 in AI: from principles to industry-specific accreditation. *CoRR*, abs/2110.09232, 2021. URL:
498 <https://arxiv.org/abs/2110.09232>.
- 499 23 Mate Soos, Raghav Kulkarni, and Kuldeep S. Meel. Crystalball: Gazing in the black box
500 of SAT solving. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of*
501 *Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal,*
502 *July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages
503 371–387, 2019. URL: https://doi.org/10.1007/978-3-030-24258-9_26.
- 504 24 Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based
505 algorithm selection for SAT. *J. Artif. Intell. Res.*, 32:565–606, 2008. URL: [https://doi.org/](https://doi.org/10.1613/jair.2490)
506 [10.1613/jair.2490](https://doi.org/10.1613/jair.2490).