Globally Interpretable Classifiers via Boolean Formulas with Dynamic Propositions

Reijo Jaakkola[1 [0000−0003−4714−4637]], Tomi Janhunen[1 [0000−0002−2029−7708]], Antti Kuusisto[1,2 [0000−0003−1356−8749]], Masood Feyzbakhsh Rankooh[2 [0000−0001−5660−3052]], and Miikka Vilander[1 [0000−0002−7301−939X]]

[1] Tampere University, Tampere, Finland, [2] University of Helsinki, Helsinki, Finland

Abstract. Interpretability and explainability are among the most important challenges of modern Artificial Intelligence, even mentioned in various legislative sources. In this work, we develop a method for extracting immediately human interpretable classifiers from tabular data. Our classifiers take the form of short Boolean formulas built with propositions that can either be directly extracted from categorical attributes or dynamically computed from numeric ones during training. Our method is implemented using Answer Set Programming. We investigate eight well-known tabular datasets and compare our formula-based methods with state-of-the-art techniques for tabular data, namely, XGBoost and random forests. Over all datasets, the accuracies obtained by our approach are similar to the reference methods. The overall advantage of our classifiers is that they are very short and immediately human intelligible, as opposed to black-box classifiers produced by the reference methods.

1 Introduction

Human interpretability is one of the main challenges of modern Artificial Intelligence (AI). This has led to an increasing interest in explainable AI (or XAI for short). The right to obtain explanations from automated processes concerning individuals is mentioned in various legislative sources such as the EU General Data Protection Regulation [13] and the California Consumer Privacy Act [10].

Interpretability is typically divided into *local* and *global* variants. The former relates to providing reasons why a classifier made a particular decision for a particular input, while global interpretability means that the classifier's behaviour can be explained comprehensively—without limiting to specific inputs.

In this paper, we introduce and investigate a method for producing globally interpretable classifiers for binary classification tasks on tabular data. The global nature of the interpretability of our classifiers stems from the fact that the formulas acting as classifiers are very short Boolean assertions about the attributes of the original datasets. Thus, the formulas also read as *very simple statements in natural language*. As a concrete example, the formula

$$\begin{aligned}\textbf{not}\big(\quad & \texttt{bare\_nuclei\_measure} \geq 6 \\ \textbf{or}\ \ & \texttt{clump\_thickness} \geq 7 \\ \textbf{or}\ \ & \texttt{cell\_size\_uniformity\_measure} \geq 5\ \big)\end{aligned}$$

is a classifier obtained from our experiments on a dataset concerning breast cancer. The formula is clearly short and it is immediately clear what the general meaning of the formula is—at least to an expert from the medical field in question. The accuracy of this formula is 97.1 %. This is striking, as the state-of-the-art method XGBoost [11] obtains practically the same average accuracy of 97.2 % on the same data, but produces a large black-box classifier.

Our method is implemented declaratively by deploying Answer Set Programming (ASP; see [9] for an overview) and, more specifically, its theory-based extension ASP *modulo difference logic*, abbreviated ASP(DL) [19]. Using simple difference constraints, the discretization of numeric values takes place dynamically during the search for actual (optimal) classifiers. As shown in this work, Boolean statements resulting from such dynamic discretization offer more fine-grained primitives for classifying data points and give rise to even shorter classifying formulas compared to an earlier static median-based approach [18].

Results. We run our method on eight datasets originating from the UCI machine learning repository. As tabular data is still challenging to deep learning approaches [7,15], we use XGBoost [11] and Scikit-learn's implementation of random forests [8] as reference methods. These are esteemed state-of-the-art methods for tabular data [15,25] and thus natural targets of comparison.

We investigate three different variants of our method: the static median-based formula-size method of [18], a dynamic discretization based on single pivot points r, and a version with intervals $[r, r']$. The focus is on investigating how much improvement dynamic discretization approaches can provide. From Fig. 2 and Table 4, one can see that for several datasets, the dynamic approaches either improve accuracy or decrease formula size needed to obtain a similar accuracy.

The average accuracy and standard deviation of each method over ten-fold cross-validation are reported in Fig. 2. For all datasets, we obtain immediately interpretable formulas with accuracy similar to the reference methods. Formulas obtained as classifiers are reported in Table 5. Table 4 lists average formula lengths over all runs. The *average over the average formula lengths* of all experiments was 5.575, 4.45, and 3.625 for the median, pivot and interval methods, respectively, showing decreased length and thus potentially better interpretability when moving from median via pivot to the interval method (cf. Table 4).

The example formulas in Table 5 were taken from the first round of the ten-fold cross validation in each case. As an example, consider the classifier $p_{[25,60]} \wedge q_{[128,196]}$ obtained with the interval method from the Diabetes dataset. Here the attribute p is the age of the person and the attribute q is their measured glucose value. The formula offers high glucose and age between 25 and 60 as an explanation for diabetes. Another example is the formula $p \vee q_{\geq 767}$ obtained with the pivot method from four different splits of the BankMarketing dataset (with slight variations on the bound of q). The classification task concerns whether a marketing call resulted in subscription for a term deposit. Here the attribute p indicates whether the customer has previously subscribed for a deposit and the attribute q gives the duration of the call in seconds. So, a customer is likely to subscribe to a deposit if they have done so previously or the telemarketer can

keep them on the phone for long enough. The accuracies of this formula on the holdout data ranged from 88.1 % to 90.3 % for different splits.

We conclude that in all cases, our classifiers are immediately globally interpretable and the accuracy is similar to the ones obtained by XGBoost and random forests. The interpretability is a real gain, as classifiers produced by these reference methods are black-box by nature. Indeed, the classifiers returned by them are ensembles of decision trees with tens to thousands of members.

A shortcoming of our methods is its computational cost, making it unfeasible if classifiers must be obtained fast. The most demanding experiments took days in a high-power computation environment, while the comparison methods completed faster with an ordinary laptop. The runtimes are discussed in Sect. 5.3.

Related work. Explainable AI is an active field and logic-based explainability a growing subfield; see [20] for a thorough survey. Much of the recent work on logic-based XAI was initiated in [24] and [17], where natural minimality notions such as prime implicants are utilized as explanations. While much of logic-based XAI focuses on local aspects, the current work studies global explainability via small formulas. Small classifiers are generally regarded as global explanations [2,5], and relating to this "simplicity first" philosophy, already [16] reported that in many of the 16 datasets studied, even a single attribute was sometimes enough to get an accuracy not drastically smaller than that of a C4.5 decision tree.

The current paper continues the line of research initiated in [18] where each numeric attribute was made Boolean statically, with the intuitive reading "the value of the attribute is above median" for the respective Boolean variable. That paper presents three experiments with a single split cross validation. In the current paper, we redo the experiments of [18] with the eight datasets studied, including the ones used in [18]. As mentioned above, the related results are in Fig. 2 and Tables 4–6. While reasonable classifiers are obtained, the ones using dynamic predicates $p_{[r,r']}$ and $p_{\geq r}$ lead to formulas that are shorter or more accurate, while keeping the running times doable.

Other relevant approaches to global explainability using small classifiers include, e.g., the articles [2,5]. These studies investigate the use of small rule lists and decision trees which are optimal with respect to size and training error. The empirical results reported in these works also demonstrate the surprising effectiveness of interpretable models on real-world tabular data. For further related work on global explanations and interpretable AI, see the survey [23].

The quest for short explanations is compatible with the principle of Occam's razor [6] and the early search-based approaches, such as current best hypothesis search [21], already implement this idea. However, the performance of computers has improved drastically since then and the solver technology used to implement the search for hypotheses in the present work has emerged by the side.

2 Preliminaries

In this section, we introduce logical concepts related to discretization and use the toy data set given in Table 1 to illustrate the concepts in a series of exam-

Table 1. Toy data set randomly extracted from the Diabetes dataset and its Booleanized version based on the median values (Mdn) of the columns. The attributes are Glucose level (Glu), Insulin level (Ins), Body Mass Index (BMI), and Age (A).

W	Glu q	Ins t	BMI r	A p	C c
1	138	167	34.6	21	1
2	86	71	30.2	24	0
3	189	846	30.1	59	1
4	88	44	29.9	23	0
5	141	128	25.4	24	0
6	90	54	37.7	29	0
7	107	48	22.9	23	1
8	120	105	39.7	29	0
9	124	205	32.9	30	1
10	129	125	38.5	43	1
Mdn	124	125	32.9	29	

W	Glu q_m	Ins t_m	BMI r_m	A p_m	C c
1	1	1	1	0	1
2	0	0	0	0	0
3	1	1	0	1	1
4	0	0	0	0	0
5	1	1	0	0	0
6	0	0	1	1	0
7	0	0	0	0	1
8	0	0	1	1	0
9	1	1	1	1	1
10	1	1	1	1	1

ples. A finite set $\tau = \{p_1, \ldots, p_k\}$ of proposition symbols is called a **Boolean vocabulary**. The syntax of propositional logic over the vocabulary τ, denoted PL[τ], is given by the following BNF grammar: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi$, where $p \in \tau$. We define the equivalence connective as the abbreviation $\varphi \leftrightarrow \psi := (\varphi \wedge \psi) \vee (\neg\varphi \wedge \neg\psi)$. Formulas in PL[$\tau$] are also called τ-formulas in the sequel. The **size** of a τ-formula φ is defined as the number of occurrences of proposition symbols and connectives $\neg, \vee, \wedge$ in the formula.

Example 1. The vocabulary of the Booleanized data set in Table 1 is $\tau = \{p_m, q_m, r_m, t_m, c\}$. For instance, the size of the τ-formula $\neg(p_m \wedge r_m)$ is 4. ■

A **Boolean τ-model** is a structure $M = \langle W, V\rangle$, where the finite set $W \neq \emptyset$ is the **domain** of M and the function $V : \tau \to \mathcal{P}(W)$ is a **τ-valuation**. More informally, a Boolean τ-model is just a multiset of propositional assignments $\tau \to \{0, 1\}$. For $p \in \tau$, the set $V(p) \subseteq W$ is the set of points, where the proposition p is considered to be true. Given a τ-valuation V, we extend it in the standard way to a valuation $V : \text{PL}[\tau] \to \mathcal{P}(W)$ so that all τ-formulas are covered.

Example 2. The Booleanized data set in Table 1 gives rise to a τ-model $M = \langle W, V\rangle$ where $W = \{1, \ldots, 10\}$ is represented by the rows of the table. The valuation V is determined by the Boolean values listed in the columns of propositions so that $V(q_m) = \{1, 3, 5, 9, 10\} = V(t_m)$, $V(r_m) = \{1, 6, 8, 9, 10\}$, and $V(p_m) = \{3, 6, 8, 9, 10\}$. Thus, we have, e.g., $V(p_m \wedge (t_m \vee q_m)) = \{3, 9, 10\}$. ■

Consider a $\tau \cup \{c\}$-model $M = \langle W, V\rangle$, where c (for *classification*) is a target attribute. Given a τ-formula φ, the **accuracy** of φ over M is

$$\text{acc}(\varphi, M) := |V(\varphi \leftrightarrow c)|/|W|. \tag{1}$$

Intuitively, the accuracy of φ is the percentage of points where φ agrees with c.

Example 3. The target attribute in Table 1 is c. The accuracies of propositions are $\mathrm{acc}(q_m, M) = \mathrm{acc}(t_m, M) = 0.80$, $\mathrm{acc}(r_m, M) = 0.60$, and $\mathrm{acc}(p_m, M) = 0.60$. For the formula $p_m \wedge (t_m \vee q_m)$ from Ex. 2, the accuracy is 0.80. ■

Next, we move on from propositional structures to structures with both Boolean and numeric attributes. The datasets we consider in the experiments also have categorical attributes with more than two categories, but we use one-hot encodings to Booleanize the categorical attributes as a preprocessing step, see Sect. 5.1 for details. A **Boolean attribute** over a set of data points W is simply a subset of W. As seen above, the interpretations of proposition symbols p in propositional logic are Boolean attributes $V(p) \subseteq W$. A **numeric attribute** over W is a function $f : W \to R$, where R is a finite set of floating point numbers, linearly ordered by $\leq$. We denote the set of functions $f : W \to R$ by R^W.

Example 4. For the dataset of Table 1, the codomain R consists of the numeric values listed in the columns of the functions q, t, r, and p. E.g., for the valuation function q associated with the Glucose level attribute, we have $q(5) = 141$. ■

A **general vocabulary** $\nu = \tau \uplus \eta$ is a disjoint union of a Boolean vocabulary $\tau = \{p_1, \ldots, p_\ell\}$ and a **numeric vocabulary** $\eta = \{s_1, \ldots, s_m\}$. A **general valuation** V over ν and R is a function $V : \nu \to \mathcal{P}(W) \uplus R^W$ which splits naturally into $V_\tau : \tau \to \mathcal{P}(W)$ and $V_\eta : \eta \to R^W$. For convenience, the function $V_\eta(s)$ assigned to a numeric attribute $s \in \eta$ is denoted by s^{V_η}. A **general structure** is a pair $\langle W, V \rangle$ with a domain W and a general valuation V.

Example 5. For the dataset of Table 1, the vocabulary ν is partitioned into $\tau = \{c\}$ and $\eta = \{q, t, r, p\}$. The respective valuation V maps, for example, c to $\{1, 3, 7, 9, 10\}$ and p to a function p^V from W to the set of ages listed in the column of p. ■

In what follows, we define two different ways to discretize general vocabularies and structures. The first is based on a single *pivot value* $r \in R$ while the second exploits an *interval* $[r, r'] \subseteq R$ with $r, r' \in R$. We begin by the pivotal case.

Let $\nu = \tau \uplus \eta$ be a general vocabulary. The **complete pivot vocabulary** over η and R is the (finite) set of proposition symbols $s_{\geq r}$ where $s \in \eta$ and $r \in R$. Intuitively, the complete pivot vocabulary contains all possible propositions $s_{\geq r}$ one might need when Booleanizing the numeric vocabulary η with pivots. A subset τ_η of the complete pivot vocabulary is an **R-pivot discretization** of a numeric vocabulary η if there is a bijection $f : \eta \to \tau_\eta$ such that for each $s \in \eta$, $f(s) = s_{\geq r}$ for some pivot $r \in R$. In this way, each numeric attribute $s \in \eta$ is discretized as a single Boolean attribute $s_{\geq r}$.

Example 6. The median-based discretization shown in Table 1 is a special case of R-pivot discretizations. The Boolean variables q_m, t_m, r_m, and p_m are equivalent to the pivoted attributes $q_{\geq 124}$, $t_{\geq 125}$, $r_{\geq 32.9}$, and $p_{\geq 29}$, respectively. ■

Let $M = \langle W, V \rangle$ be a general structure over $\nu = \tau \uplus \eta$. A Boolean structure $M' = \langle W, V' \rangle$ is a **pivot discretization** of M, if the following conditions hold:

1. V' is a function $V' : \tau \uplus \tau_\eta \to \mathcal{P}(W)$, where τ_η is an R-pivot discretization of η;
2. $V'(p) = V(p)$ for every Boolean attribute $p \in \tau$; and
3. $V'(s_{\geq r}) = \{w \in W \mid s^V(w) \geq r\}$ for every pivoted attribute $s_{\geq r} \in \tau_\eta$, where s^V is the function $V(s) : W \to R$ assigned to the numeric attribute $s \in \eta$.

The **pivot class** over M is the set $\mathrm{PD}(M)$ of all pivot discretizations of M.

Example 7. Besides the median-based discretization pointed out in Example 6, the pivot class $\mathrm{PD}(M)$ contains a pivot discretization based on every possible choice of pivots for the numeric attributes in $\eta = \{q, t, r, p\}$. ■

Our second approach to discretizing a numeric attribute $s \in \eta$ exploits an interval $[r, r']$ with $r, r' \in R$ rather than a singe pivot value. This gives rise to a Boolean attribute $s_{[r,r']}$ denoting the fact that $r \leq s \leq r'$. Obviously, if m is the maximum value for s, then $s_{\geq r}$ is equivalent to $s_{[r,m]}$. On the other hand, $s_{[r,r']}$ is expressible with a Boolean formula $s_{\geq r} \wedge \neg s_{\geq r''}$ where the bound r'' is the least value of s such that $r'' > r'$. If r' is maximal and there is no such r'', then $s_{\geq r'}$ is sufficient alone. Based on these observations, interval-based formulas are potentially more succinct than respective pivot-based formulas.

Example 8. The flexibility of Boolean attributes $s_{[r,r']}$ based on intervals can be observed from Table 2. While $p_{[30,59]}$ captures a range of values, the formula $\neg p_{[24,29]}$ covers two ranges and $r_{[29.9,29.9]}$ singles out a unique value. ■

A **complete interval vocabulary** over η and R consists in propositions of the form $s_{[r,r']}$ with $s \in \eta$ and $r \leq r'$ for $r, r' \in R$. Moreover, the notions of an R-**interval discretization** and the **interval discretization** of a general structure M are analogues of their pivotal counterparts, obtained by replacing $s_{\geq r}$ by $s_{[r,r']}$, and the condition $s^V(w) \geq r$ by $r \leq s^V(w) \leq r'$. The **interval class** $\mathrm{ID}(M)$ collects all possible interval discretizations of M.

Given a set of Boolean formulas $\mathcal{F}$ and $\mathcal{M}$ a set of Boolean models (possibly over different vocabularies), the **maximum accuracy** over $\mathcal{F}$ and $\mathcal{M}$ is the maximum value $\mathrm{acc}(\varphi, M)$ such that $\varphi \in \mathcal{F}$, $M \in \mathcal{M}$, and only propositions from the vocabulary of M occur in φ. In this case, the Boolean formula φ is deemed to **realize** the maximum accuracy (over the model M). As regards a general structure M over $\nu = \tau \uplus \eta$ and R, a Boolean formula φ is an M-**pivot formula** (resp. M-**interval formula**), if φ is a Boolean $\tau \uplus \tau_\eta$-formula for an R-pivot (resp. R-interval) discretization τ_η of η. We use $\mathrm{PF}(M)$ (resp. $\mathrm{IF}(M)$) to denote the set of all M-pivot formulas (resp. M-interval formulas). Moreover, given a bound ℓ, the set of M-pivot formulas (resp. M-interval formulas) of size at most ℓ is denoted by $\mathrm{PF}_\ell(M)$ (resp. $\mathrm{IF}_\ell(M)$). In the sequel, it is particularly interesting to find formulas φ from either $\mathrm{PF}_\ell(M)$ and $\mathrm{IF}_\ell(M)$ that realize the maximum accuracy over M—the goal of the method presented in the next section.

Example 9. The dataset of Table 1 is characterized by the Boolean formulas collected in Table 2. They have been computed using the encodings of Sect. 4. Two observations are immediate. First, longer formulas are more expressive and

Table 2. Maximally accurate formulas of increasing length. The missing entries (−) omit the respective formulas when accuracy cannot be improved at length ℓ.

Discretization	$\ell = 1$		$\ell = 2 \ldots 3$		$\ell = 4 \ldots 5$	
Median	t_m	80%	−		$q_m \wedge (r_m \vee p_m)$	90%
Pivot	$p_{\geq 30}$	80%	$q_{\geq 124} \wedge r_{\geq 30.1}$	90%	−	
Interval	$p_{[30,59]}$	80%	$\neg p_{[24,29]}$	90%	$\neg(r_{[29.9,29.9]} \vee p_{[24,29]})$	100%

more accurate—regardless of the discretization principle. Second, pivot-based (resp. interval-based) discretizations tend to be more accurate than their median-based (resp. pivot-based) counterparts. Interestingly, negated formulas appear when interval-based propositions are used. E.g., $\neg p_{[24,29]}$ is true when the age (p) of the person is less than 24 or greater than 29. While this is a bit vague justification for diabetes as such, the size $\ell = 4$ explanation incorporates an analogous condition on the BMI attribute (r). Since a single data point is removed from consideration, the resulting 100 %-accurate formula suggests overfitting. The full diabetes dataset is analyzed later (cf. row D in Table 5) using the dynamic method of Sect.3 together with cross-validation to avoid overfitting. ■

When discussing real-life datasets in the other sections below, we will suppress the model notation $M = \langle W, V \rangle$ and only discuss the set W of data points as well as some subsets of W, to be interpreted as submodels of M.

3 The dynamic formula size method

We next give a step-by-step description of our method. We detail the method for pivot discretization; the case of interval discretization is identical with pivot formulas replaced by interval formulas. The median method uses a static discretization based on medians, otherwise being the same as the other variants.

Note that we here assume that we have already separated ten percent of the original dataset to be used as a final testing data (also known as *holdout data*). Thus the input to the process is a dataset W consisting of the remaining 90 %.

In the procedure, we scan through all formulas up to increasing formula length bounds. We shall describe in Sect. 4 how this can be done quite efficiently, avoiding a naive brute-force search. We use early stopping to avoid overfitting and formulas too long to be interpretable. The stages of our method are:

1. An input to the method is a tabular dataset W with binary and numeric attributes $X_1, \ldots, X_m, q$, where q is a binary target attribute. From this data W, we randomly separate a 30 % set $W_v \subseteq W$ to be used as *validation data*. The remaining 70 % set $W_{tr} \subseteq W$ is called the *training data*.
2. For a length bound ℓ starting with 1, we scan through the set $\mathrm{PF}_\ell(W_{tr})$ of all W_{tr}-pivot formulas φ with maximum length ℓ, choosing the formula φ_ℓ that realizes the maximum accuracy over the training set W_{tr}.

3. For each chosen formula φ_ℓ, we record its accuracy over the validation set W_v. Then, we update the best validation accuracy so far, denoted Δ. If the accuracies of both $\varphi_{\ell-1}$ and φ_ℓ are smaller than Δ, we trigger early stopping and move to step 4. Otherwise, we increase ℓ by one and return to step 2.
4. We scan through the formulas $\varphi_1, \ldots, \varphi_{\ell-1}$ to find out the smallest number $L \in \{1, \ldots, \ell-1\}$ such that the accuracy of φ_L over W_v is the same as that of $\varphi_{\ell-1}$. Therefore, if many formulas of different lengths obtain the same accuracy, we choose the smallest length, leading to shorter final formulas.
5. Finally, we use the data $W = W_{tr} \cup W_v$ and the length bound L to compute the final output formula by scanning through the set $\mathrm{PF}_L(W)$ of all W-pivot formulas ψ with maximum length L, and then choosing the formula with the maximum accuracy over W.

We emphasize that the search conducted in steps 2 and 5 for the optimal pivot-formula is not carried out as a brute-force search. Even though the search remains a challenging problem, we use symmetry reductions and other techniques to reduce the size of the hypothesis space to make it practically feasible to conduct the search on real-life datasets as detailed in Sect. 4.1. Our experimental results presented in Sect. 5.2, along with the training times we report in Sect. 5.3, indicate that the method described above can be useful in practice.

In step 3, we stop once the accuracy of the formulas on the 30-percent validation data is worse than the best accuracy so far, twice in a row. This early stopping is done to avoid overfitting. In general, when the length bound is increased at first, the accuracy on both the training data W_{tr} and the validation data W_v improves. At some point, however, the validation accuracy will start to decline while the training accuracy keeps on improving. This is when the formulas start to overfit to the training data. Our simple early stopping strategy aims to avoid overfitting and, moreover, to obtain short, interpretable formulas.

Finally in step 5 we use the formula length found in step 4 to compute the final classifier using all of the available data. This is to ensure that our early stopping procedure does not lead to throwing away 30 percent of the data.

4 Implementation

In this section, we present an implementation of dynamic propositions in ASP *modulo difference logic* (ASP(DL)) [19] that is natively implemented by the *Clingo-dl* system[1]. This extension of ASP exploits *difference constraints* in a very simple normal form $x - y \leq k$ where x and y are variables and k is a constant. For the purposes of this work, the domain of integers is sufficient, since the order of attribute values is more significant than their actual magnitudes. It is also worth noting that the satisfiability of sets of difference constraints can be checked very efficiently, e.g., using the Bellman-Ford algorithm [4].

Without going into syntactic details of rules used in logic programs, we recall that the *answer sets* of a logic program P with a vocabulary τ_P are sets of atoms

[1] https://github.com/potassco/clingo-dl

$S \subseteq \tau_P$ satisfying a fix-point equation $S = \mathrm{LM}(P^S)$ where (i) $\mathrm{LM}(\cdot)$ denotes the *least model* of the positive program given as argument and (ii) the reduct P^S is obtained by partially evaluating the negative conditions of P with respect to S [14]. In view of definitions in Sect. 2, the answer sets $S_1, \ldots, S_n$ of P induce a τ_P-model $\langle \{1, \ldots, n\}, V_P \rangle$ such that $V_P(a) = \{i \mid a \in S_i\}$ for an atom $a \in \tau_P$. If difference constraints $(x - y \leq k)$ are incorporated into a logic program P, they are treated as special *difference atoms* $p_{\langle x,y,k \rangle}$ in τ_P forming the respective subsignature $\tau_d \subseteq \tau_P$ of the program P. If $\tau_d \neq \emptyset$, the set of linear inequalities

$$\{x - y \leq k \mid p_{\langle x,y,k \rangle} \in \tau_d \cap S\} \ \cup \ \{x - y > k \mid p_{\langle x,y,k \rangle} \in \tau_d \setminus S\} \tag{2}$$

must be additionally satisfied by $S = \mathrm{LM}(P^S)$. Note that the negations $x - y > k$ in (2) can be rewritten in the normal form as $y - x \leq -k - 1$, i.e., they can be efficiently treated as difference constraints. In contrast with traditional, *satisfiability modulo theories* (SMT) approaches [3], Eq. (2) assigns a *strict* interpretation to difference atoms [19], since auxiliary atoms $p_{\langle x,y,k \rangle}$ falsified by S contribute the negation $(x - y > k)$ to the theory being satisfied. This is in line with the $\subseteq$-minimality of answer sets in general and *program completion* [12]: the satisfaction of (2) amounts to satisfying the completing formula $p_{\langle x,y,k \rangle} \leftrightarrow (x - y \leq k)$ for $p_{\langle x,y,k \rangle} \in \tau_d$. When an answer set S is reported to the user, some concrete values can be generated for variables x and y involved in atoms $p_{\langle x,y,k \rangle} \in S \cap \tau_d$.

Example 10. One dataset in this paper has age as an attribute, actual ages ranging from 21 to 81 years. Using an integer variable a for an age limit, we can define its range in terms of two ASP(DL) facts: $z - a \leq -21$ and $a - z \leq 81$ where z is an auxiliary variable thought to hold 0 as its value (cf. footnote 2). Note that if the value of z were $v \neq 0$, then z can be reset to 0 by subtracting v from the values of all variables. Assuming a particular data point where the age happens to be 27, its Booleanization b can be captured with a rule $b \leftarrow a - z \leq 27$. Using auxiliary atoms introduced above, the facts and the rule above give rise to answer sets $S_1 = \{p_{\langle z,a,-21 \rangle},\ p_{\langle a,z,81 \rangle},\ p_{\langle a,z,27 \rangle},\ b\}$ and $S_2 = \{p_{\langle z,a,-21 \rangle},\ p_{\langle a,z,81 \rangle}\}$. The former is supported by assignments $z = 0$ and $a = 21$, and the latter by $z = 0$ and $a = 28$, i.e., minimal non-negative values also listed by *Clingo-dl*. ■

Our implementation is based on a modular architecture divided into three layers as illustrated in Fig. 1. The top layer adopts the encoding of [18] to generate hypotheses, evaluate them with respect to data points, and calculate the overall accuracy as part of the objective function. The middle layer encodes the principles for Booleanizing the data and decides which attribute is the target of explanation and which attributes may contribute to the actual explanations. The bottom layer extracts the values of the attributes from the real data as a preprocessing step. Although the architecture in Fig. 1 looks *stratified* at first glance, it does not have a unique answer set as typical for stratified logic programs. In particular, the two top layers involve choices that create a search space for optimization. The layers are further detailed in Sect. 4.1–4.3 below. By changing logic programs that encode the topmost layers, it is also easy to take different hypothesis spaces and strategies or Booleanizing attributes into consideration.

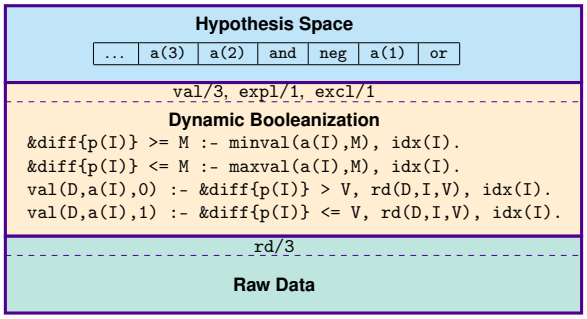

Fig. 1. Three Layers of the Implementation

4.1 Generating hypotheses

The top layer has access to the underlying data set in Boolean form only. The data is abstracted using a 3-argument predicate `val(D,A,V)` where `D` is the identifier of the data point, `A` the attribute and `V` the value, i.e., either `0` or `1`. In addition to this view over the data, the hypothesis generator receives a target attribute and any denied attributes expressed in terms of predicates `expl/1` and `excl/1`, respectively. Based on this interface, the generator is completely unaware whether the *original* data is Boolean or not, and thus its implementation can be kept completely independent of this aspect.

We have revised the hypothesis generator from [18] to respect the interface described above. The generator represents formulas as sequences of attributes and logical operators following the idea of *reverse Polish notation*. For instance, the sequence $a_3, a_2, \wedge, \neg, a_1, \vee$ represents the formula $a_1 \vee \neg(a_2 \wedge a_3)$ that can be reconstructed as well as evaluated using a stack, e.g., for any data point. The encoding deploys a parameter `l` for the maximum length of the sequence sought. The dots "..." in the first cell (see Fig. 1) illustrate the fact that the sequence may start by unused symbol positions whenever a shorter (optimal) explanation is feasible. We refer the reader to [18] for further details of the encoding, but recall that the objective function maximizes the accuracy defined in (1) as the primary criterion, and minimizes the length of the formula as the secondary criterion. In this way, the length parameter `l` acts just as an upper bound and the user need not provide the exact length of explanations in advance.

It should also be emphasized that the search implemented by a conflict-driven learning solver (such as *Clingo-dl)* is substantially different from exhaustive brute-force search. The search space is limited by several factors, such as attribute values admitted by the underlying data set, the value of the objective function, as well as conflicts learned during the search. In addition, the encoding of [18] imposes syntactic symmetry constraints to reduce the hypothesis space.

4.2 Dynamic Booleanization of attributes

The values of Boolean attributes and one-hot encoded categorical attributes can be forwarded unchanged through the `val/3` predicate. The dynamic approach is applied to other numeric attributes that are made available in terms of the `rd(D,I,V)` predicate with arguments `D` for a data point, `I` for the index of an attribute, and `V` for an integer value, thus assuming some mapping from actual values to the integer domain (see Sect. 4.3). Then, given an attribute `a(I)` indexed by `I`, its minimum and maximum values can be determined and recorded as respective facts using predicates `minval/2` and `maxval/2`.

In the dynamic approach based on a single pivot, the first two rules in Fig. 1 choose a value `p(I)` in this range.[2] The latter two rules compare the actual value `V` at a data point `D` against `p(I)` and Booleanize `V` as `0`, if `V` is smaller than `p(I)`, and as `1`, otherwise. Although difference atoms are exploited in a corner case (cf. footnote 2), the benefit is that the values of the attribute `a(I)` need not be sorted beforehand: the Boolean value can be decided locally based on the pivot `p(I)`. The choice of pivots is completely dynamic: any values are fine as long as the required Booleanization is realized and the current hypothesis is justified in terms of accuracy and length—in harmony with the idea of SMT. Moreover, the ASP(DL) ecoding given in Fig. 1 is essentially linear.

The generalization for the interval-based Booleanization is obtained by introducing two pivots, namely `l(I)` and `u(I)`, instead of `p(I)` for an attribute `a(I)`. The rules in Fig. 1 are refined accordingly. While the minimum value of an attribute `a(I)` is the least value for `l(I)`, its maximum value gives the greatest value for `u(I)`. Moreover, the difference constraint `&diff{u(I) - l(I)} >= 0` makes `l(I)` smaller than or equal to `u(I)`. Based on their values, the value `V` of the attribute `a(I)` is Booleanized as `1` if `V` falls in the range from `l(I)` to `u(I)` including endpoints and `0`, otherwise. These updates to the encoding do not jeopardize linearity as discussed above in the case of a single pivot.

4.3 Treating non-integer numeric values

Given a *raw* data file, e.g., as a comma-separated values (CSV) file, the predicate `rd/3` can be computed procedurally. We leave the exact mapping to integers at the user's discretion. For instance, non-integer values can be multiplied by suitable constants and rounded off or truncated at the decimal point. For the dynamic approach based on pivots, it is more important to preserve the order than the magnitudes when considering the values of a particular attribute.

5 Experiments

We compare the interval, pivot and median variants of the formula-size method empirically to random forests and XGBoost. The two latter methods are very

[2] Note that the constant bounds $l \leq x \leq u$ for a variable x can be expressed by $z - x \leq -l$ and $x - z \leq u$ in the normal form, using an auxiliary variable z for "zero".

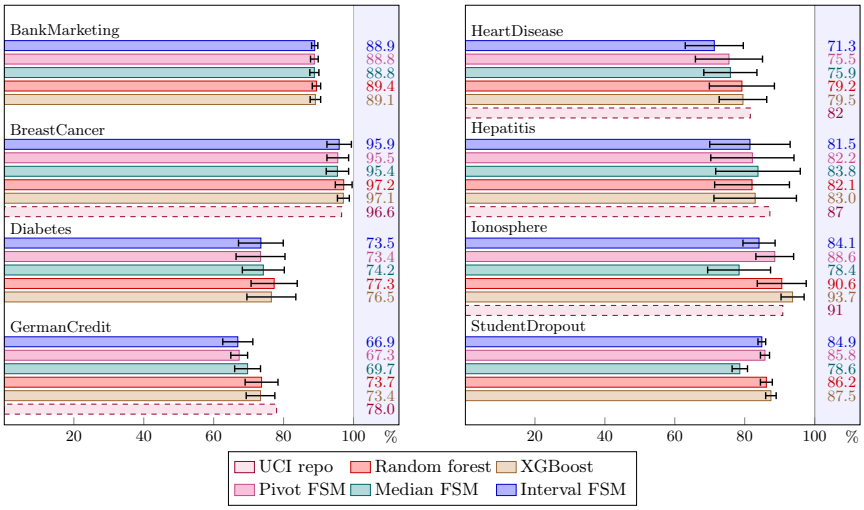

Fig. 2. The average test accuracies and standard deviations obtained for each dataset. Where available, we include also accuracies reported as Baseline Model Performance in the UCI repository, although these are not directly comparable due to the unknown technical particularities behind the UCI numbers.

commonly used state-of-the-art methods for tabular data [15,25]. We first present our experimental setup in detail, then discuss the results both in terms of accuracy and the formulas obtained, and finally report the runtimes of the methods.

5.1 Experimental setup

We test all five methods on eight binary classification tasks from the UCI machine learning repository: BankMarketing, BreastCancer, Diabetes, GermanCredit, HeartDisease, Hepatitis, Ionosphere, and StudentDropout. See Table 3 for a summary of the datasets. The dataset StudentDropout was originally a ternary classification task, but we converted it into a binary one. The Diabetes dataset is no longer available from the UCI repository, but it can be found in OpenML repository with ID:43582.

The datasets we use contain both categorical and numeric attributes. For categorical attributes we use one-hot encodings. That is, for each categorical attribute X and category a of X, we create a Boolean attribute p_a^X, which is true in the points of category a w.r.t. the attribute X. We use this preprocessing for all tested methods, including the reference methods random forest and XGBoost. This type of preprocessing is necessary for the reference methods as the implementations we use only support numeric attributes.

We move on to numeric attributes. As dynamic discretization is an important feature of our method and the reference methods work directly with numeric attributes, we do not Booleanize numeric attributes beforehand. However, for

Table 3. Summary of datasets used in the experiments.

Dataset	# Features	# Datapoints
BankMarketing	51	4521
BreastCancer	9	683
Diabetes	8	768
GermanCredit	20	1000
HeartDisease	13	303
Hepatitis	17	155
Ionosphere	34	351
StudentDropout	36	4424

the formula size methods, we perform a small preprocessing step to transform floating point attributes into integers for easier compatibility with ASP by simply multiplying the numbers by a suitable power of ten to make them integers. This preprocessing is not done in the case of random forests and XGBoost, which receive the numeric attributes as such.

All missing values have been removed from the datasets. In the case of the Hepatitis dataset, we removed two columns with many missing values. The Diabetes dataset does not have missing values, but it has features which take values that are inconsistent with background knowledge (e.g., the dataset contains rows where the value of blood pressure is zero). We removed all rows with such values.

We use ten-fold cross-validation for all methods and all datasets. That is, we randomly split the original data W into ten equal (when possible) parts. For each such 10 % part, we feed the 90 % complement into the method as the full training data W_0 and set aside the 10 % part itself as holdout data W_{hold}. After the method has obtained a final classifier using W_0, we record the accuracy of that classifier on W_{hold}. We report the average and standard deviation of the accuracy over the ten 90/10 splits given in the beginning.

As the methods based on formula size are computationally quite intensive, we employ a timeout for runs. This raises the possibility that our early stopping trigger is never met before a run times out. In this circumstance, we record the best validation accuracy obtained before timeout and look back at the smallest formula length that achieves the best accuracy. In a vast majority of our experiments, the early stopping was triggered before the computation timed out. The exceptions are the dataset StudentDropout and some individual splits scattered among the other datasets.

For all three formula-size methods, the runs were conducted on a Linux cluster featuring Intel Xeon 2.40 GHz CPUs with 8 CPU cores per run, employing a timeout of 72 hours per instance and a memory limit of 64 GB.

For random forests we use the implementation of Scikit-learn [22]. We use nested cross-validation for tuning the hyperparameters for both random forests and XGBoost. As for the formula-size methods, we use a 70/30-split. For hyperparameter optimization, we use Optuna [1] with 100 trials. The hyperparameter spaces used for random forests and XGBoost are the same as in [15]. For the

Table 4. The average length of the ten formulas obtained for all ten 90/10 splits for all three formula-size methods. The averages of the average lengths are 3.625, 4.45 and 5.575 for the interval, pivot and median methods, respectively.

Dataset	Interval FSM	Pivot FSM	Median FSM
BM	2.8	4.0	6.6
BC	5.5	6.5	8.8
D	4.3	2.9	4.2
GC	3.1	3.0	5.5
HD	2.9	4.4	4.2
H	2.3	2.8	2.5
I	4.0	5.1	5.7
SD	4.1	6.9	7.1

Table 5. Example formulas for all three formula-size methods. These formulas were obtained from the first 90/10 split of each dataset.

Dataset	Interval FSM	Pivot FSM	Median FSM
BM	$p \vee q_{[794,2769]}$	$(p \vee q_{\geq 767}) \wedge \neg r$	$q_m \wedge (s \vee p)$
BC	$p_{[1,5]} \wedge q_{[1,6]} \wedge r_{[1,4]}$	$\neg(p_{\geq 6} \vee q_{\geq 7} \vee r_{\geq 5})$	$\neg((q_m \wedge s_m) \vee (p_m \wedge t_m \wedge u_m))$
D	$p_{[25,60]} \wedge q_{[128,196]}$	$q_{\geq 158} \vee (p_{\geq 29} \wedge r_{\geq 27.6} \wedge s_{\geq 109})$	$s_m \wedge p_m \wedge (t_m \vee q_m)$
GC	$p \vee q_{[4,30]}$	$\neg r_{\geq 10876}$	$s \vee \neg(t \wedge u)$
HD	$p_{[0.0,2.6]} \wedge (q \vee (r \wedge s_{[176,240]}))$	$\neg t \vee (q \wedge r)$	q
H	$p_{[2.9,6.4]}$	q	q
I	$s_{[0.24,0.39]} \vee (r_{[-0.73,0.75]} \wedge q_{[0.41,1.00]})$	$r_{\geq -0.79} \wedge q_{\geq 0.15} \wedge p$	$((v \wedge u) \vee t) \wedge p$
SD	$p_{[0,2]} \vee \neg q$	$\neg q \wedge (p_{\geq 2} \vee \neg r_{\geq 1})$	$\neg(p_m \vee (q \wedge (s \vee t_m)))$

reader's convenience, we also report the used hyperparameter spaces in Appendix A. We ran these experiments simply on a laptop.

5.2 Results

The results of our experiments are summarized in Fig. 2 and Tables 4 and 5. In Fig. 2, we report the average accuracy and standard deviation of each method over ten-fold cross-validation. We can see that the formula-size methods give accuracies comparable to those of random forests and XGBoost for every dataset. The best performance was obtained in the BankMarketing dataset, where the difference is less than 1 percentage point, while the largest gap of 5 percentage points is found in the Ionosphere dataset.

To measure interpretability, we report the average length of the final formula for each dataset (Table 4) and an example formula for each dataset (Table

5). The example formulas are from the first 90/10 split of each ten-fold cross-validation. The formulas are short for each method, the largest average length being 8.8 (median method, BreastCancer data). Keeping in mind that formula length includes all connectives as well as proposition symbols, this means that the formulas obtained are easily human interpretable as well as compact abstractions of original datasets. The *averages of the average lengths* are 3.625, 4.45 and 5.575 for the interval, pivot and median methods, respectively. Thus the transition from median via pivot to the interval method tends to shorten the formulas, so our dynamic approaches can improve interpretability.

Comparing the three formula-size methods, we can observe from Table 4 an antitone relationship between the expressive power of a method and the average lengths of formulas produced by it: the more expressive the method is, the shorter the formulas it produces. This makes sense, as the expressive power of a method is related to how quickly the early stopping criterion is met.

From Fig. 2 we can also observe that on these datasets, the greater expressive power of a method does not always translate to better accuracy on the holdout data. Indeed, the median method, which has the lowest expressive power, obtains on several datasets accuracies that are similar to or even slightly better than the ones obtained via the alternative methods based on formula size. One possible explanation for this phenomenon is that when utilizing the same formula size, the interval and pivot methods have a higher susceptibility to overfitting to the training data in contrast with the more robust median-based method.

For Ionosphere and the StudentDropout, the interval and pivot methods have obtained significantly better accuracies than the median method. In both cases this can be explained by the fact that these datasets contain numerical features for which the median is not a good splitting point. In the case of Ionosphere both the interval and pivot methods use the features r and q for which the medians are 0.02 and 0.81 respectively, values which differ significantly from those that occur in the formulas found by these methods. For StudentDropout, all the formulas use the attribute p for which the median is 5 while the pivot and interval methods choose a pivot point of 2.

5.3 Runtimes

The average training time of each formula-size method is reported in Table 6. The runtimes, while large, are still very feasible to run on powerful computation setups that are available today. We do not report the runtimes of random forests and XGBoost as they were all run on a laptop and took less than five minutes.

In most cases, the more computationally intensive method has a higher runtime. However, this is occasionally reversed. For example, in the BankMarketing dataset this happens for the pivot and the median methods. There are at least two possible reasons for this. Firstly, the more expressive pivot method could trigger early stopping at shorter formulas, thus saving runtime compared to the median method. Secondly, due to its computational intensity, the pivot method can reach timeout sooner, while the median-based method can keep performing runs for longer formulas and accumulate more runtime overall.

Table 6. The average training time in seconds per split of each formula-size method for each dataset.

Dataset	Interval FSM	Pivot FSM	Median FSM
BM	281 120	98 403	249 263
BC	95 868	4 089	2 193
D	190 758	61 903	19
GC	103 844	198 504	146 400
HD	120 394	3 817	139
H	26 722	119	47
I	5 811	4 310	4 545
SD	375 093	480 024	260 895

6 Conclusion

In this work, we introduce a method that extracts immediately human interpretable classifiers from tabular data using dynamic discretizations that are determined on the fly while optimizing the accuracy and the length of the classifier. Based on comparing three variants of the method against state-of-the art methods for classifying tabular data, viz. random forests and XGBoost, we conclude that explanations obtained by our dynamic methods considered are competitively accurate while much easier and informative to interpret.

As regards future work, the layered architecture described in Sect. 4 opens up several avenues for further research. When it comes to the syntax of explanations, it is interesting to consider further Boolean connectives and specific normal forms for classifiers. Yet further strategies for the discretization of attributes can be obtained as refinements of the pivot/interval-based approaches of this work.

References

1. T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *ACM SIGKDD 2019*, pages 2623–2631, 2019.
2. E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research*, 18(234):1–78, 2018.
3. C. Barrett and C. Tinelli. *Satisfiability Modulo Theories*, pages 305–343. Springer, 2018.
4. R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:78–90, 1958.
5. D. Bertsimas and J. Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.
6. A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Inf. Process. Lett.*, 24(6):377–380, 1987.
7. V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(6):7499–7519, 2024.

8. L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
9. G. Brewka, T. Eiter, and M. Truszczynski. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.
10. California-OAG. CCPA regulations: Final regulation text, 2021.
11. T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *ACM SIGKDD 2016*, pages 785–794. ACM, 2016.
12. K. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.
13. EU-GDPR. Regulation (EU) 2016/679 of the European Parliament and of the Council, 2016.
14. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP 1988*, pages 1070–1080, 1988.
15. L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *NeurIPS Datasets and Benchmarks Track*, 2022.
16. R. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63—90, April 1993.
17. A. Ignatiev, N. Narodytska, and J. Marques-Silva. Abduction-based explanations for machine learning models. In *AAAI 2019*, pages 1511–1519. AAAI Press, 2019.
18. R. Jaakkola, T. Janhunen, A. Kuusisto, M. Rankooh, and M. Vilander. Short Boolean formulas as explanations in practice. In *JELIA 2023*, pages 90–105. Springer, 2023.
19. T. Janhunen, R. Kaminski, M. Ostrowski, S. Schellhorn, P. Wanko, and T. Schaub. Clingo goes linear constraints over reals and integers. *Theory and Practice of Logic Programming*, 17(5–6):872–888, 2017.
20. J. Marques-Silva. Logic-based explainability in machine learning. In L. Bertossi and G. Xiao, editors, *Reasoning Web, 2022, Tutorial Lectures*, pages 24–104. Springer, 2022.
21. T. Mitchell. Generalization as search. *Artif. Intell.*, 18(2):203–226, 1982.
22. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
23. C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.
24. A. Shih, A. Choi, and A. Darwiche. A symbolic approach to explaining Bayesian network classifiers. In *IJCAI 2018*, pages 5103–5111, 2018.
25. R. Shwartz-Ziv and A. Armon. Tabular data: Deep learning is not all you need. *Inf. Fusion*, 81:84–90, 2022.

A Appendix

Here we report the hyperparameter spaces for random forest and XGBoost. Hyperparameters not listed below were kept at their default values.

Random forests

- `max_depth`: None, 2, 3, 4
- `n_estimators`: Integer sampled from $[9.5, 3000.5]$ using log domain
- `criterion`: gini, entropy
- `max_features`: sqrt, log2, None, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
- `min_samples_split`: 2, 3
- `min_samples_leaf`: Integer sampled from $[1.5, 50.5]$
- `bootstrap`: True, False
- `min_impurity_decrease`: 0.0, 0.01, 0.02, 0.05

XGBoost

- `max_depth`: Integer sampled from [1,11]
- `n_estimators`: Integer sampled from [100, 5900] with step-size being 200
- `min_child_weight`: Float sampled from [1.0, 100.0] using log domain
- `subsample`: Float sampled from [0.5, 1.0]
- `learning_rate`: Float sampled from [1e-5, 0.7] using log domain
- `colsample_bylevel`: Float sampled from [0.5, 1.0]
- `gamma`: Float sampled from [1e-8, 7.0] using log domain
- `reg_lambda`: Float sampled from [1.0, 4.0] using log domain
- `reg_alpha`: Float sampled from [1e-8, 100.0] using log domain