Generating Explainable Rule Sets from Tree-Ensemble Learning Methods by Answer Set Programming

Akihiro Takemura   Katsumi Inoue

The Graduate University for Advanced Studies, SOKENDAI, Japan
National Institute of Informatics, Japan

atakemura@nii.ac.jp   inoue@nii.ac.jp

We propose a method for generating explainable rule sets from tree-ensemble learners using Answer Set Programming (ASP). To this end, we adopt a decompositional approach where the split structures of the base decision trees are exploited in the construction of rules, which in turn are assessed using pattern mining methods encoded in ASP to extract interesting rules. We show how user-defined constraints and preferences can be represented declaratively in ASP to allow for transparent and flexible rule set generation, and how rules can be used as explanations to help the user better understand the models. Experimental evaluation with real-world datasets and popular tree-ensemble algorithms demonstrates that our approach is applicable to a wide range of classification tasks.

1 Introduction

Interpretability in machine learning is the ability to explain or to present in understandable terms to a human [8]. Interpretability is particularly important when, for example the goal of the user is to gain knowledge from some form of explanations about the data or process through machine learning models, or when making high-stakes decisions based on the outputs from the machine learning models where the user has to be able to trust the models.

In this work we address the problem of explaining and understanding tree-ensemble learners by extracting meaningful rules from them. This problem is of practical relevance in business domains where the understanding of the behavior of high-performing machine learning models and extraction of knowledge in human readable form can aid users in the decision making process. We use *Answer Set Programming (ASP)* [14, 22] to generate rule sets from tree-ensembles. ASP is a declarative programming paradigm for solving difficult search problems. An advantage of using ASP is its expressiveness and extensibility, especially when representing constraints. To our knowledge, ASP has never been used in the context of rule sets generation from tree-ensembles, although it has been used in pattern mining, e.g., [19, 16, 12, 27].

Generating interpretations for machine learning models is a challenging task since it is often necessary to account for multiple competing objectives. For instance, if accuracy is the most important metric, then it is in direct conflict with interpretability, because accuracy favors specialization while interpretability favors generalization. Any interpretation method should also strive to imitate the behavior of learned models as to minimize misrepresentation of models, which in turn may result in misinterpretation by the user. While there are many interpretation methods available (some are covered in Section 2), we propose to use ASP as a medium to represent the user requirements declaratively and to quickly search feasible solutions for faster prototyping. By implementing a rule selection method as a post-processing step to model training, we aim to offer an off-the-shelf objective interpretation tool as an alternative to subjective manual rule selection, which could be applied to existing processes with minimum modification.

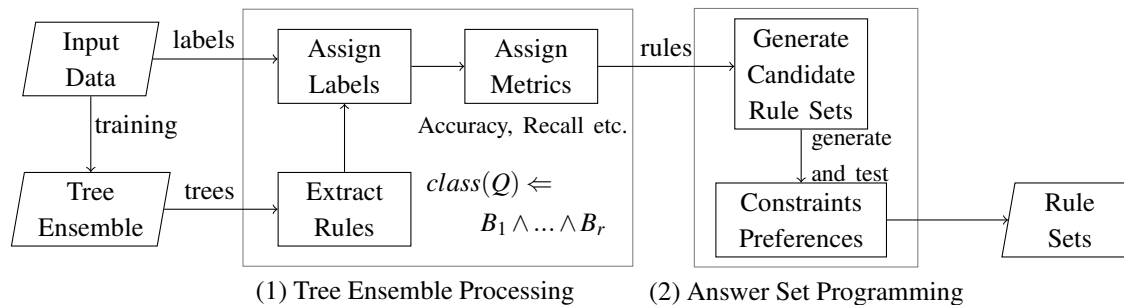(1) Tree Ensemble Processing   (2) Answer Set Programming

Figure 1: Overview of our framework

We consider the two-step procedure for rule set generation from tree-ensembles (Figure 1): (1) extracting rules from trained decision tree-ensembles, and (2) computing sets of rules according to selection criteria and preferences encoded declaratively in ASP. For the first step, we employ the efficiency and prediction capability of modern tree-ensemble algorithms in finding useful feature partitions for prediction from data. For the second step, we exploit the expressiveness of ASP in encoding constraints and preference to select useful rules from tree-ensembles, and rule selection is automated through a declarative encoding. The generated rule sets therefore not only act as interpretations for tree-ensemble models but are also explainable.

We then evaluate our approach from two perspectives: the number and relevance of rules in the rule sets. The number of rules is often associated with interpretability, with a large number of rules being less desirable. Performance metrics such as classification accuracy, precision and recall can be used as a measure of relevance of the rules to the prediction task.

This paper makes the following contributions:

- We present a novel application of Answer Set Programming (ASP) for interpreting machine learning models. We propose a method to generate explainable rule sets from tree-ensemble models with ASP. More generally, this work contributes to the growing body of knowledge on integrating symbolic reasoning with machine learning.

- We present how the rule set generation problem can be reformulated as an optimization problem, where we leverage existing knowledge on declarative pattern mining with ASP.

- To demonstrate the practical applicability of our approach, we provide both qualitative and quantitative results from evaluations with public datasets, where machine learning models are used in a realistic setting.

The rest of this paper is organized as follows. In Section 2 we review and discuss related works. In Section 3, we review tree-ensembles, ASP and pattern mining. Section 4 presents our method to generate rule sets from tree-ensembles using pattern mining and optimization encoded in ASP. Section 5 presents experimental results on public datasets. Finally in Section 6 we present the conclusions.

2   Related Works

Summarizing tree-ensembles has been studied in literature, see for example, Born Again Trees [3], defragTrees [17] and inTrees [7]. While exact methods and implementations differ among these examples, a popular approach to tree-ensemble simplification is to create a simplified decision tree model that approximates the behavior of the original tree-ensemble model. Depending on how the approximate tree

model is constructed, this could lead to a deeper tree with an increased number of conditions which makes them difficult to interpret.

Integrating association rule mining and classification is also known, e.g., Class Association Rules (CARs)[23], where association rules discovered by pattern mining algorithms are combined to form a classifier. Repeated Incremental Pruning to Produce Error Reduction (RIPPER)[6] was proposed as an efficient approach for classification based on association rule mining, and it is a well-known rule-based classifier. In CARs and RIPPER, rules are mined from data with dedicated association rule mining algorithms, then processed to produce a final classifier.

Interpretable classification models is another area of active research. Interpretable Decision Sets (IDS)[21] are learned through an objective function which simultaneously optimizes accuracy and interpretability of the rules. In Scalable Bayesian Rule Lists (SBRL)[32], probabilistic IF-THEN rule lists are constructed by maximizing the posterior distribution of rule lists. In RuleFit[11], a sparse linear model is trained over rules extracted from tree-ensembles. RuleFit is the closest to our work in this regard, in the sense that both RuleFit and our method extract conditions and rules from tree-ensembles, but differ in the treatment of rules and representation of final rule sets. In RuleFit, rules are accompanied by regression coefficients, and it is left up to the user to further interpret the result.

Lundberg et al.[24] showed how a variant of SHAP[25], which is a post-hoc interpretation method, can be applied to tree-ensembles. While our method does not produce importance measures for each feature, the information about which rule fired to reach the prediction can be offered as an explanation in a human readable format. Shakerin and Gupta [30] proposed a method to use LIME weights[29] as a part of learning heuristics in inductive learning of default theories. Instead of learning rules with heuristics from data, our method directly handles rules which exist in decision tree models with answer set solver.

Guns et al.[15] applied constraint programming (CP), a declarative approach, to itemset mining. This constraint satisfaction perspective led to the development of ASP encoding of pattern mining e.g., [19, 16]. Gebser et al.[12] applied preference handling to sequential pattern mining, and Paramonov et al.[27] extended the declarative pattern mining by incorporating dominance programming (DP) from Negrevergne et al.[26] to the specification of global constraints. Paramonov et al.[27] proposed a hybrid approach where the solutions are effectively screened first with dedicated algorithms for pattern mining tasks, then declarative ASP encoding is used to extract condensed patterns. While aforementioned works focused on extracting interesting patterns from transaction or sequence data, our focus in this paper is to generate rule sets from tree-ensemble models to help users interpret the behavior of machine learning models. In terms of ASP encoding, we use dominance relations similar to the ones presented in Paramonov et al.[27] to further constrain the search space.

3 Background

3.1 Tree-Ensembles

Tree-Ensemble (TE) models are machine learning models widely used in practice, typically, but not limited to, when learning from tabular datasets. A TE consist of multiple base decision trees each trained on an independent subset of the input data. For example, Random Forests [2] and Gradient Boosted Decision Tree (GBDT) [10] are tree-ensemble models. Recent surge of efficient and effective GBDT algorithms, e.g., LightGBM [20], has led to wide adoption of TE models in practice. Although individual decision trees are considered to be interpretable [18], ensembles of decision trees are seen as less interpretable.

The purpose of using TE models is to predict the unknown value of an attribute y in the dataset, referred to as *labels*, using the known values of other attributes $\mathbf{x} = (x_1, x_2, ..., x_m)$, referred to as *features*. For brevity we restrict our discussion to classification problems. During the training or learning phase, each input instance to the TE models is a pair of features and labels, i.e. $(\mathbf{x}_i, y_i)$, where i denotes the instance index, and during the prediction phase, each input instance only include features, $(\mathbf{x}_i)$, and the model is tasked to produce predictions $\hat{y}_i$. A collection of input instances, complete with features and labels, is referred to as a *dataset*. Given a dataset $\mathscr{D} = \{(\mathbf{x}_i, y_i)\}$ with $n \in \mathbb{N}$ examples and $m \in \mathbb{N}$ features, a decision tree classifier t will predict the class label $\hat{y}_i$ based on the feature vector $\mathbf{x}_i$ of the i-th sample: $\hat{y}_i = t(\mathbf{x}_i)$. A tree-ensemble $\mathscr{T}$ uses $K \in \mathbb{N}$ trees and additionally an aggregation function f over the K trees which combines the output from the trees: $\hat{y}_i = f(t_{k \in K}(\mathbf{x}_i))$. In the case of Random Forest, for example, f is a majority voting scheme (i.e. `argmax` of `sum`), and in GBDT f may be a summation followed by softmax to obtain $\hat{y}_i$ in terms of probabilities.

In this paper a decision tree is assumed to be a binary tree where the internal nodes hold split conditions (e.g., $x_1 \leq 0.5$) and leaf nodes hold information related to class labels such as the number of supporting data points per class label that have been assigned to the leaf nodes. Richer collections of decision trees provide higher performance and less uncertainty in prediction compared to a single decision tree. Typically, each TE model has specific algorithms for learning base decision trees, adding more trees and combining outputs from the base trees to produce the final prediction. In GBDT, the base trees are trained sequentially by fitting the residual errors from the previous step. Interested readers are referred to [10], and its more recent implementations LightGBM [20] and XGBoost [5].

3.2   Answer Set Programming

Answer Set Programming [22] has its roots in logic programming and non-monotonic reasoning. A *normal logic program* is a set of rules of the form

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

where each a_i is a first-order atom with $1 \leq i \leq n$ and `not` is *default negation*. If only a_1 is included ($n = 1$), the above rule is called a *fact*, whereas if a_1 is omitted, it represents an *integrity constraint*. A normal logic program induces a collection of models, which are called *answer sets* defined by the stable model semantics [14]. Additionally, in modern ASP systems, constructs such as *conditional literals* and *cardinality constraints* are supported. The former in *clingo* [13] are written in the form $\{a(X) : b(X)\}$[1], and expanded into the conjunction of all instances of $a(X)$ where corresponding $b(X)$ holds. The latter are written in the form $s_1 \{a(X) : b(X)\} s_2$, which is interpreted as $s_1 \leq \#\texttt{count}\{a(X) : b(X)\} \leq s_2$ where s_1 and s_2 are treated as lower and upper bounds, respectively, thus the statement holds when the count of instances $a(X)$ where $b(X)$ holds, is between s_1 and s_2. The minimization (or maximization) of an objective function can be expressed with `#minimize` (or `#maximize`). *clingo* supports multiple optimization statements in a single program, and one can implement multi-objective optimization with priorities by defining two or more optimization statements.

3.3   Pattern Mining

In a general setting, the goal of pattern mining is to find interesting patterns from data, where patterns can be, for example, itemsets, sequences and graphs. For example, in *frequent itemset mining* [1], the task is

---

[1]Unless otherwise noted, we follow the Prolog-style notation in logic programs where strings beginning with a capital letter are variables, and others are predicate symbols or constants.

to find all subsets of items that occur together more than the threshold count in databases. In this work, the patterns of interest are sets of predictive rules. A *predictive rule* has the form $c \Leftarrow s_1 \wedge s_2 \wedge, ..., s_n$, where c is a class label, and $\{s_i\}$ ($1 \leq i \leq n$) represents conditions.

For pattern mining with constraints, the notion of *dominance* is important, which intuitively reflects pairwise preference relation ($<^*$) between patterns [26]. Let C be a constraint function that maps a pattern to $\{\top, \bot\}$, and let p be a pattern, then the pattern p is *valid* iff $C(p) = \top$, otherwise it is *invalid*. An example of C is a function that checks the support of a pattern is above the threshold. The pattern p is said to be *dominated* iff there exists a pattern q such that $p <^* q$ and q is valid under C. Dominance relations have been used in ASP encoding for pattern mining [27].

There are existing ASP encodings of pattern mining algorithms, e.g., [19, 12, 27], that can be used to mine itemsets and sequences. Here we develop and apply our own encoding on rules to extract interesting rules from tree-ensembles. On the surface, our problem setting may appear similar to frequent itemset and sequence mining, however, rule set generation is different from these pattern mining problems. We can indeed borrow some ideas from frequent itemset mining for encoding, however, our goal is not to decompose rules (cf. transactions) into individual conditions (cf. items) then constructing rule sets (cf. itemsets) from conditions, but rather to treat each rule in its entirety then combining rules to form rule sets. The body (antecedent) of a rule can also be seen as a sequence, where the conditions are connected by conjunction connective $\wedge$, however, in our case, the ordering of conditions does not matter, thus sequential mining encodings that use slots to represent positional constraints [12] cannot be applied directly to our problem.

4 Rule Set Generation

4.1 Problem Statement

The rule set generation problem is represented as a tuple $P = \{R, M, C, O\}$, where R is the set of all rules in the tree-ensemble, M is set of meta-data and properties associated with each rule in R, C is the set of user-defined constraints including preferences, and O is the set of optimization objectives. The goal is to generate a set of rules from R by selection under constraints C and optimization objectives O, where constraints and optimization may refer to the meta-data M. In the following sections, we describe how we construct each R, M, C and O, and finally how we solve this problem with ASP.

4.2 Rule Extraction from Decision Trees

Recall that a tree-ensemble $\mathscr{T}$ is a collection of K decision trees, and we refer to individual trees t_k with subscript k. An example of a decision tree-ensemble is shown in Figure 2. A decision tree t_k has N_{t_k} nodes and L_{t_k} leaves. Each node represents a split condition and there are L_{t_k} paths from the root node to the leaves. For simplicity, we assume only features that have orderable values (continuous features) are present in the dataset in the examples below.[2] The tree on the left in Figure 2 has 4 internal nodes including the root node with condition $[x_1 \leq 0.2]$ and 5 leaf nodes, therefore there are 5 paths from the root note to the leaf nodes 1 to 5.

---

[2]Real datasets may have unorderable categorical values. For example, in the *census* dataset, occupation (Sale, etc.) and education (Bachelors, etc.) are categorical features. Support for categorical feature split is implementation-dependent, however in general one can replace the continuous split with a subset selection e.g., $x_c \in \{x_{c1}, x_{c2}, ...\}$
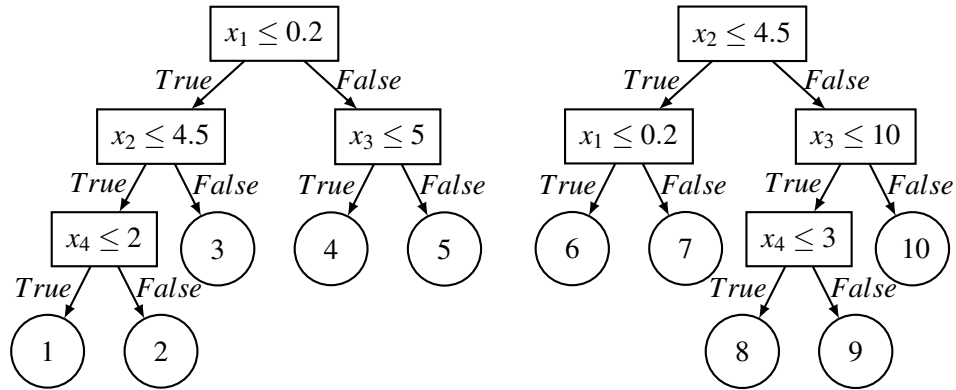
Figure 2: A simple decision tree-ensemble consisting of two decision trees. The rule associated with each node is given by the conjunction of all conditions associated with nodes on the paths from the root node to that node.

From the left-most path of the decision tree on the left in Figure 2, the following prediction rule is created. We assume that node 1 predicts class label 1 in this instance.[3]

$$class(1) \Leftarrow (x_1 \leq 0.2) \wedge (x_2 \leq 4.5) \wedge (x_4 \leq 2)$$

Assuming that node 2 predicts class label 0, we also construct the following rule (note the reversal of the condition on x_4):

$$class(0) \Leftarrow (x_1 \leq 0.2) \wedge (x_2 \leq 4.5) \wedge (x_4 > 2)$$

We can also construct subsets of rules by applying each of the conditions sequentially and computing the predicted label at each step. For example, from the last rule we may construct the following rule:

$$class(1) \Leftarrow (x_1 \leq 0.2) \wedge (x_2 \leq 4.5)$$

The set of all rules, R, is constructed as follows:

1. Enumerate all possible paths from the root node to the leaves. For a binary decision tree with depth d_k, the maximum number of leaf nodes is 2^{d_k}, which is also the maximum number of paths from the root node to the leaf nodes.

2. For each path, at each subsequent node on the path to the leaf node, the split condition of the node is appended to the body (antecedent, set of conditions) of the rule. For a decision tree the maximum number of such rules is the same as the maximum number of nodes in the tree, i.e. $2^{d_k+1} - 1$.

3. Compute the predicted class label for each rule. For simplicity, we apply all conditions in the rule and calculate the most likely class label from the count data (`argmax` of counts).

4. Add the generated rules to the candidate rule set R.

5. Repeat steps 1 to 4 for each tree t_k where $1 \leq k \leq K$, in the ensemble of K trees.

---

[3]Label=1 and 0 refer to the attributes in the dataset and have different meaning depending on the dataset. For example, in the *census* dataset, label=1 and 0 mean that the personal income is more than $50,000 and that it is no more than $50,000, respectively.

Table 1: List of predicates representing a rule in ASP.

Predicate	Meaning
`rule(X)`	`X` holds the rule index.
`condition(X,I)`	Rule `X` has condition `I`.
`support(X,S)`	Support `S` of rule `X`, the number of instances that is covered by rule `X`.
`size(X,L)`	Number of conditions in rule `X` (length, `L`).
`error_rate(X,E)`	Error rate ($1 - accuracy$), `E`, of the rule `X` evaluated in the training data.
`accuracy(X,A)`	Accuracy score of rule `X`.
`precision(X,P)`	Precision score of rule `X`.
`recall(X,R)`	Recall score of rule `X`.
`f1_score(X,F)`	F1-score of rule `X`.
`predict_class(X,C)`	Predicted class label `C` of rule `X`.

By constructing the candidate rule set *R* in this way, the bodies (antecedents) of rules included in rule sets are guaranteed to exist in at least one of the trees in the tree ensemble. Rule sets generated in this manner are therefore faithful to the representation of the original model in this sense. If we were to construct rules from the unique set of split conditions, the resulting rule may have combinations of conditions that may not exist in any of the trees.

4.3   Computing Metrics and Meta-data for Selection

After the candidate rule set *R* is constructed, we gather information about the performance and properties of each rule and collect them into a set *M*. Performance metrics, in general, measure how well a rule can predict class labels. Examples of widely adopted performance metrics in machine learning are: accuracy, precision, recall and F1-score. We compute multiple metrics for a single rule, to meet a range of user requirements for interpretation. For example, one user may only be interested in simply most accurate rules (maximize accuracy), whereas another user could be interested in more precise rules (maximize precision), or rules with more balanced performance (maximize F1-score). The meta-data, or properties, of a rule are information such as the size of the rule, as defined by the number of conditions in the rule, or the number of instances which are covered by the rule. These properties can be used in the selection step to define competing objectives. For example, one can expect a very long rule with relatively large number of rules to be precise, but the rule may be too specific and may not cover many instances. Moreover, a long rule is more difficult to comprehend than a short, concise rule. In this case, the size property needs to be minimized, while the precision metric is maximized.

The candidate rule set *R* and meta-data set *M* are represented as facts in ASP, as shown in Table 1. For example, the first rule in Section 4.2 may be represented as follows[4]:

```
% rule 1
rule(1). condition(1,1). condition(1,2). condition(1,3). support(1,10).
size(1,3). accuracy(1,50). error_rate(1,50). precision(1,30).
recall(1,40). f1_score(1,34). predict_class(1,1).
```

---

[4]The performance metrics are for illustration purposes only and are chosen arbitrarily.

4.4   Encoding Constraints

For the rule set generation task, we consider three types of constraints: (1) local constraints that are applied on a per-rule basis, for example, to select rules that meet the minimum support threshold, (2) pairwise constraints that are applied to pairs of rules, which include dominance relations, and (3) global constraints that are applied to a set of rules, for example to control the total number of conditions in the rule set.

To encode local constraints, a predicate `valid(X)` is introduced, to specify that a `rule(X)` is valid whenever `invalid(X)` cannot be inferred:

```
valid(X) :- rule(X), not invalid(X).
```

This example of a local constraint eliminates rules with low support:

```
% this will eliminate rules that apply to less than 10 instances
invalid(X) :- rule(X), support(X,S), S < 10.
```

Pairwise constraints can be used to encode dominance relations between rules. For a rule X to be dominated by Y, Y must be strictly better in one criterion than X and at least as good as X or better in other criteria. For example, in the following case we encode the dominance relation between rules using the F1 score, support and size of the rule, where we prefer rules that are small (more interpretable), have higher support (covers more instances) and perform well (higher F1 score).

```
% cannot be dominated
:- dominated.
% X is dominated by Y if ...
ge_f1_leq_size_geq_sup(Y) :- selected(X), valid(Y), size(X,Sx), size(Y,Sy),
    f1_score(X,Fx), f1_score(Y,Fy), support(X,Spx), support(Y,Spy),
    Fx < Fy, Sx >= Sy, Spx <= Spy.
geq_f1_le_size_geq_sup(Y) :- selected(X), valid(Y), size(X,Sx), size(Y,Sy),
    f1_score(X,Fx), f1_score(Y,Fy), support(X,Spx), support(Y,Spy),
    Fx <= Fy, Sx > Sy, Spx <= Spy.
geq_f1_leq_size_ge_sup(Y) :- selected(X), valid(Y), size(X,Sx), size(Y,Sy),
    f1_score(X,Fy), f1_score(Y,Fy), support(X,Spi), support(Y,Spy),
    Fx <= Fy, Sx >= Sy, Spx < Spy.
dominated :- valid(Y), ge_f1_leq_size_geq_sup(Y).
dominated :- valid(Y), geq_f1_le_size_geq_sup(Y).
dominated :- valid(Y), geq_f1_leq_size_ge_sup(Y).
```

Global constraints are applied to rule sets in addition to the local and pairwise constraints and preferences. For example, the following "generator" encoding puts a limit on the maximum size of rule sets that are considered:

```
% pick at least 1 rule and at maximum 10 rules for each predict_class
1 { selected(X) :  predict_class(X, K), valid(X) } 10 :- class(K).
```

This encoding will select at least 1 and up to 10 valid rules for each class label `K`. The properties of rule sets can also be used to construct constraints. For instance, one can put restrictions the maximum number of conditions in rule sets, using the aggregate atom `#sum`:

```
% total number of conditions should not exceed 30
:- #sum { S,X : size(X,S), selected(X) } > 30.
```

Exact set of constraints and preferences depend on the problem domain, use-case and/or intention of the user. The expressiveness of the ASP language allows one to represent constraints in a declarative manner under the semantics of logic programming.

4.5   Optimizing Rule Sets

Finally, we pose the rule set generation problem as a multi-objective optimization problem, given aforementioned facts and constraints encoded in ASP. The desiderata for generated rule sets may contain multiple competing objectives. For instance, we consider a case where the user wishes to collect accurate rules that cover a large number of instances, while minimizing the number of conditions in the set. This is encoded as a group of optimization statements:

```
% maximize accuracy and support, minimize the number of conditions
#maximize { A,X : selected(X), accuracy(X,A)}.
#maximize { S,X : selected(X), support(X,S)}.
#minimize { L,X : selected(X), size(X,L)}.
```

For optimization, we introduce a measure of overlap between the rules to be minimized. Intuitively, minimizing this objective should result in rule sets where rules share only a small number of conditions, which should further improve the interpretability of the resulting rule sets. Specifically, we introduce a predicate `rule_overlap(X,Y,Cn)` to measure the degree of overlap between rules X and Y.

```
% number of shared conditions between rules
rule_overlap(X,Y,Cn) :- selected(X), selected(Y), X!=Y,
    Cn = #count { Cx : Cx=Cy, condition(X,Cx), condition(Y,Cy) }.
#minimize { Cn,X : selected(X), selected(Y), rule_overlap(X,Y,Cn) }.
```

5   Experiments

We evaluate our rule set generation framework on several public datasets and compare the performance to existing methods including rule-based classifiers.

5.1   Experimental Setup

We used 10 publicly available datasets from the UCI Machine Learning Repository[5] [9]. The summary of these datasets is shown in Table 2. We used *Clingo* 5.4.0[6] [13] for answer set programming, and set the time out to 600 seconds.[7] We used RIPPER implemented in Weka [31] and an open source implementation of RuleFit[8] where Random Forest was selected as the rule generator, and scikit-learn[9] [28] for general machine learning functionalities. Our experimental environment is a desktop machine with Ubuntu 18.04, Intel Core i9-9900K 3.6GHz (8 cores/16 threads) and 64GB RAM.

In order to evaluate the performance of the extracted rule sets, we implemented a naive rule-based classifier which is constructed from the rule sets extracted with our method. In this classifier, we apply the rules sequentially to the validation dataset and if all conditions within a rule are true for an instance

---

[5]https://archive.ics.uci.edu/ml/index.php
[6]https://potassco.org/clingo/
[7]Full ASP encoding of our method is available in the supplementary materials.
[8]https://github.com/christophM/rulefit
[9]https://scikit-learn.org/

Table 2: Datasets used in the experiments. *#data* and *#feature* refer to the number of data points (rows) and features (columns), respectively. The number of categorical features is shown in parenthesis.

Dataset	# data	# feature	y = 1
autism	704	20 (18)	screening result
breast	699	9 (9)	malignant
census	299,286	42 (29)	income > 50k
credit_a	690	14 (8)	application accepted
credit_t	30,000	23 (10)	payment next month
heart	270	13 (8)	disease present
ionosphere	351	34 (0)	good radar return
kidney	400	24 (13)	chronic disease
krvskp	3,196	36 (36)	white can win
voting	435	16 (16)	democrat

in the dataset, the consequent of the rule is returned as the predicted class. More formally, given a set of rules $R_s \subset R$ with cardinality $|R_s|$ that shares the same consequent $class(Q)$, we represent this rule-based classifier as the disjunction of antecedents of the rules:

$$class(Q) \Leftarrow body(R_1) \lor body(R_2) \lor ... \lor body(R_r) \text{ where } 1 \leq r \leq |R_s|$$

For a given data point, it is possible that there are no rules applicable, and in such cases the most common class label in the training dataset is returned.

We conduct the evaluation experiment in the following order. First, we train Random Forest and LightGBM on the datasets in Table 2. We then apply our rule set generation method to the trained tree-ensemble models. Finally, we construct a naive rule-based classifier using the set of rules extracted in the previous step, and calculate performance metrics on the validation set. This process is repeated in a 5-fold stratified cross validation setting to estimate the performance. We compare the characteristics of our approach against the known methods RIPPER and RuleFit.

Table 3: Average number of candidate rules ($|R|$), size of the generated rule sets (# rule), averaged over 5 folds. Hyphen indicates a failure case where no rules could be found.

Dataset	LightGBM+ASP $|R|$	LightGBM+ASP # rule	RandomForest+ASP $|R|$	RandomForest+ASP # rule	RuleFit # rule	RIPPER # rule
autism	2.0	**1.0**	59.8	7.6	3.0	2.0
breast	131.2	**2.8**	27.8	8.8	55.8	13.0
census	8806.8	**9.0**	-	-	304.0	54.7
credit_a	275.2	**3.8**	123.4	7.4	55.2	7.0
credit_t	2098.4	**6.6**	-	-	187.8	7.4
heart	159.6	**2.8**	47.6	8.8	40.8	6.2
ionosphere	314.4	**5.2**	1127.0	9.8	272.0	7.0
kidney	179.6	**3.2**	101.0	5.8	160.6	4.4
krvskp	140.8	**7.6**	69.6	10.0	240.4	16.4
voting	59.6	**1.4**	45.2	3.4	44.0	6.2

Table 4: Average ratio of rule-based classifier's performance vs. original tree-ensembles. Acc.=accuracy, Prec.=Precision, Rec.=Recall and F1=F1 score. Performance ratio of 1 means the rule set's performance is identical to the original classifier. Hyphen indicates a failure case where no rules could be found.

	LightGBM+ASP				RandomForest+ASP				RuleFit			
Dataset	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
autism	**1.00**	**1.00**	**1.00**	**1.00**	0.70	0.47	**1.20**	0.69	1.05	1.00	**1.21**	1.11
breast	0.75	0.62	**1.05**	0.77	0.76	0.61	**1.08**	0.78	1.01	1.00	**1.03**	1.01
census	0.37	0.12	**2.01**	0.27	-	-	-	-	-	-	-	-
credit_a	0.81	0.78	**0.99**	0.85	0.94	0.89	**1.05**	0.96	1.02	0.97	**1.10**	1.03
credit_t	0.39	0.35	**2.49**	0.79	-	-	-	-	-	-	-	-
heart	0.83	0.79	**0.99**	0.85	0.69	0.59	**1.40**	0.86	1.04	0.98	**1.17**	1.08
ionosphere	0.80	0.85	**0.93**	0.87	0.69	0.71	**1.01**	0.83	1.01	**1.03**	0.98	1.00
kidney	0.74	0.73	**0.99**	0.83	0.63	0.64	**1.00**	0.78	**1.01**	**1.01**	1.00	1.00
krvskp	0.78	0.73	**0.93**	0.82	0.58	0.60	**1.03**	0.75	1.09	**1.14**	1.02	1.08
voting	0.94	**0.95**	**0.95**	**0.95**	0.66	0.64	**1.08**	0.81	1.03	1.01	**1.04**	1.02

5.2 Number of Rules

The average number of rules extracted from the data is shown in Table 3. RuleFit includes original features (called linear terms) as well as conditions extracted from the tree-ensembles in the construction of a sparse linear model, that is to say, the counts in Table 3 may be inflated by the linear terms. On the other hand, the output from RIPPER only contains rules, and RIPPER has rule pruning and rule set optimization to further reduce the rule set size. Moreover, RIPPER has direct control over which conditions to include into rules, whereas our method and RuleFit relies on the structure of the decision trees to construct rules.

Our approach consistently produces smaller rule sets compared to RuleFit, and the rule sets are comparable in size to, or smaller than, those produced by RIPPER. Comparing the size of the candidate rule set $|R|$ with the size of rule sets, our method can produce rule sets which are significantly smaller than the original model. Overall, in terms of the number of rules in the final rule set, where smaller count is desirable for better interpretability, LightGBM+ASP performed the best, followed by RIPPER. The failure cases with Random Forest (*census* and *credit_t* datasets) occurred due to leaf-only trees. Because leaf-only trees have no split conditions, rules could not be extracted and our method produced no rule sets as a result.

5.3 Relevance of Rules

To quantify the relevance of the extracted rules, we measured the ratio of performance metrics using the naive rule-based classifier by 5-fold cross validation (Table 4). Performance ratio of less than 1.0 means that the rule-based classifier performed worse than the original classifier (LightGBM and Random Forest), whereas performance ratio greater than 1 means the rule set's performance is better than the original classifier.

From Table 4 we observe that this particular encoding yields rules that have good recall, but other metrics could suffer especially in larger datasets such as *census* and *credit_t*. In this instance, F1-score was used to define dominance relations in the ASP encoding, and the performance is mostly comparable with the original model, with the exception of the *census* dataset where the F1-score was noticeably

worse. For this evaluation, we did not set any restrictions on the number of rules RuleFit could have, and it performs as well as the original Random Forest classifier in most cases.

5.4   Changing Optimization Criteria

The definition of optimization objectives has a direct influence over the performance of the resulting rule sets, and the objectives need to be set in accordance with user requirements. Because the solution space is bound by the constraints, changing the optimization statements by themselves may not give desired solutions. In an extreme case, e.g., LightGBM+ASP on the *autism* dataset, there is only 1 candidate rule to begin with and changing the optimization statements (e.g., more weight on precision) will have no effect on the final solution.

The answer sets found by *clingo* with multiple optimization statements are optimal with respect to the set of goals defined by the user. Instead of using accuracy one may use other rule metrics as defined in Table 1 such as precision and/or recall. If there are priorities between optimization criteria, then one could use the priority notation (`weight@priority`) in *clingo* to define them. Optimal answer sets can be computed in this way, however, if enumeration of such optimal sets is important, then one could use the `pareto` or `lexico` preference definitions provided by *asprin* [4] to enumerate Pareto optimal answer sets. Instead of presenting a single optimal rule set to the user, this will allow the user to explore other optimal rule sets.

6   Conclusion

In this work, we presented a method for generating explainable rule sets from tree-ensembles using pattern mining techniques encoded in ASP for the interpretation of tree-ensembles. Adopting the declarative programming paradigm with ASP allows the user to take advantage of the expressiveness of ASP in representing constraints and preferences. This makes our approach particularly suitable for situations where fast prototyping is required, since changing the constraint and preference settings require relatively low effort compared specialized mining algorithms. Useful interpretations can be generated using our approach, and combined with the expressive ASP encoding, we hope that our method will help the users of tree-ensemble models to better understand the behavior of such models.

A limitation of our method in terms of scalability is the size of search space, which is exponential in the number of valid rules. When the number of candidate rules is large, we suggest using stricter local constraints on the rules, or reducing the maximum number of rules to be included into rule sets (Section 4.4), in order to achieve reasonable solving time.

There is a number of directions for further research. First, while the current work did not modify the conditions in the rules in any way, rule simplification approaches could be incorporated to remove redundant conditions. Second, we could extend the current work to support regression problems. More generally, in future, we plan to explore how ASP and modern statistical machine learning could be integrated effectively to produce more interpretable machine learning systems.

Acknowledgments

References

[1] Rakesh Agrawal & Ramakrishnan Srikant (1994): *Fast Algorithms for Mining Association Rules*. In: *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94 1215, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 487–499.

[2] Leo Breiman (2001): *Random Forests*. Machine Learning 45(1), pp. 5–32, doi:10.1023/A:1010933404324.

[3] Leo Breiman & Nong Shang (1996): *Born Again Trees*. University of California, Berkeley, Berkeley, CA, Technical Report 1, p. 2.

[4] Gerhard Brewka, James Delgrande, Javier Romero & Torsten Schaub (2015): *Asprin: Customizing Answer Set Preferences without a Headache*. In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI '15, AAAI Press, pp. 1467–1474.

[5] Tianqi Chen & Carlos Guestrin (2016): *XGBoost: A Scalable Tree Boosting System*. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, ACM Press, San Francisco, California, USA, pp. 785–794, doi:10.1145/2939672.2939785.

[6] William W. Cohen (1995): *Fast Effective Rule Induction*. In: *Proceedings of the Twelfth International Conference on International Conference on Machine Learning*, ICML '95, Morgan Kaufmann, pp. 115–123, doi:10.1016/b978-1-55860-377-6.50023-2.

[7] Houtao Deng (2019): *Interpreting Tree Ensembles with Intrees*. International Journal of Data Science and Analytics 7(4), pp. 277–287, doi:10.1186/s12864-017-4340-z.

[8] Finale Doshi-Velez & Been Kim (2017): *Towards A Rigorous Science of Interpretable Machine Learning*. arXiv:1702.08608 [cs, stat].

[9] Dheeru Dua & Casey Graff (2017): *UCI Machine Learning Repository*. https://archive.ics.uci.edu/ml/index.php.

[10] Jerome H. Friedman (2001): *Greedy Function Approximation: A Gradient Boosting Machine*. Annals of statistics, pp. 1189–1232, doi:10.1214/aos/1013203451.

[11] Jerome H. Friedman & Bogdan E. Popescu (2008): *Predictive Learning via Rule Ensembles*. The Annals of Applied Statistics 2(3), pp. 916–954, doi:10.1214/07-AOAS148.

[12] Martin Gebser, Thomas Guyet, René Quiniou, Javier Romero & Torsten Schaub (2016): *Knowledge-Based Sequence Mining with ASP*. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI 2016, IJCAI/AAAI Press, pp. 1497–1504.

[13] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2014): *Clingo = ASP + Control: Preliminary Report*. CoRR abs/1405.3694.

[14] Michael Gelfond & Vladimir Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In: *ICLP/SLP*, 88, pp. 1070–1080.

[15] Tias Guns, Siegfried Nijssen & Luc De Raedt (2011): *Itemset Mining: A Constraint Programming Perspective*. Artificial Intelligence 175(12-13), pp. 1951–1983, doi:10.1016/j.artint.2011.05.002.

[16] Thomas Guyet, Yves Moinard & René Quiniou (2014): *Using Answer Set Programming for Pattern Mining*. In: *Actes Des Huitièmes Journées de l'Intelligence Artificielle Fondamentale (JIAF'14)*.

[17] Satoshi Hara & Kohei Hayashi (2018): *Making Tree Ensembles Interpretable: A Bayesian Model Selection Approach*. In: *International Conference on Artificial Intelligence and Statistics*, pp. 77–85.

[18] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen & Bart Baesens (2011): *An Empirical Evaluation of the Comprehensibility of Decision Table, Tree and Rule Based Predictive Models*. Decision Support Systems 51(1), pp. 141–154, doi:10.1016/j.dss.2010.12.003.

[19] Matti Järvisalo (2011): *Itemset Mining as a Challenge Application for Answer Set Enumeration*. In: *Logic Programming and Nonmonotonic Reasoning*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 304–310, doi:10.1007/978-3-642-20895-9_35.

[20] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye & Tie-Yan Liu (2017): *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*. In: *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., pp. 3146–3154.

[21] Himabindu Lakkaraju, Stephen H. Bach & Jure Leskovec (2016): *Interpretable Decision Sets: A Joint Framework for Description and Prediction*. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, ACM Press, San Francisco, California, USA, pp. 1675–1684, doi:10.1145/2939672.2939874.

[22] Vladimir Lifschitz (2008): *What is answer set programming?* In: *AAAI-08/IAAI-08 Proceedings - 23rd AAAI Conference on Artificial Intelligence and the 20th Innovative Applications of Artificial Intelligence Conference*, pp. 1594–1597.

[23] Bing Liu, Wynne Hsu & Yiming Ma (1998): *Integrating Classification and Association Rule Mining*. In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD '98, AAAI Press, New York, NY, pp. 80–86.

[24] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal & Su-In Lee (2020): *From Local Explanations to Global Understanding with Explainable AI for Trees*. Nature Machine Intelligence 2(1), pp. 56–67, doi:10.1038/s42256-019-0138-9.

[25] Scott M Lundberg & Su-In Lee (2017): *A Unified Approach to Interpreting Model Predictions*. In: *Advances in Neural Information Processing Systems*, pp. 4765–4774.

[26] Benjamin Negrevergne, Anton Dries, Tias Guns & Siegfried Nijssen (2013): *Dominance Programming for Itemset Mining*. In: *Proceedings of the 2013 IEEE 13th International Conference on Data Mining*, ICDM '13, IEEE, pp. 557–566, doi:10.1109/ICDM.2013.92.

[27] Sergey Paramonov, Daria Stepanova & Pauli Miettinen (2019): *Hybrid ASP-Based Approach to Pattern Mining*. Theory and Practice of Logic Programming 19(4), pp. 505–535, doi:10.1007/978-3-642-20895-9_35.

[28] 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 & E. Duchesnay (2011): *Scikit-Learn: Machine Learning in Python*. Journal of Machine Learning Research 12, pp. 2825–2830.

[29] Marco Tulio Ribeiro, Sameer Singh & Carlos Guestrin (2016): *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, ACM Press, San Francisco, California, USA, pp. 1135–1144, doi:10.1145/2939672.2939778.

[30] Farhad Shakerin & Gopal Gupta (2019): *Induction of Non-Monotonic Logic Programs to Explain Boosted Tree Models Using LIME*. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI '19 33, pp. 3052–3059, doi:10.1609/aaai.v33i01.33013052.

[31] Ian H. Witten, Eibe. Frank & Mark. A. Hall (2016): *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. Morgan Kaufmann.

[32] Hongyu Yang, Cynthia Rudin & Margo I. Seltzer (2017): *Scalable Bayesian Rule Lists*. In: *Proceedings of the 34th International Conference on Machine Learning*, ICML 2017 70, PMLR, pp. 3921–3930.