

BoolXAI: Explainable AI Using Expressive Boolean Formulas

Serdar Kadioğlu^{1,2}, Elton Yechao Zhu³, Gili Rosenberg⁴, John Kyle Brubaker⁴, Martin J. A. Schuetz^{4,5}, Grant Salton^{4,5,6}, Zhihuai Zhu⁴, Helmut G. Katzgraber⁴

¹ AI Center of Excellence, Fidelity Investments, MA, USA

² Department of Computer Science, Brown University, RI, USA

³ Fidelity Center for Applied Technology, FMR LLC, MA, USA

⁴ Amazon Quantum Solutions Lab, WA, USA

⁵ AWS Center for Quantum Computing, CA, USA

⁶ Institute for Quantum Information and Matter, California Institute of Technology, CA, USA

{serdark@cs.brown.edu}

Abstract

In this paper, we design, develop, and release BoolXAI, an interpretable machine learning classification approach for Explainable AI (XAI) based on expressive Boolean formulas. The Boolean formula defines a logical rule with tunable complexity according to which input data are classified. Beyond the classical conjunction and disjunction, BoolXAI offers expressive operators such as AtLeast, AtMost, and Choose and their parameterization. This provides higher expressiveness compared to rigid rules- and tree-based approaches. We show how to train BoolXAI classifiers effectively using native local optimization to search the space of feasible formulas. We provide illustrative results on several well-known public benchmarks that demonstrate the competitive nature of our approach compared to existing methods. Our work is embodied in the open-source BoolXAI library with a high-level user interface to serve researchers and practitioners. BoolXAI can be used either as a standalone interpretable classifier or for post-hoc explanations of other black-box models or observed behavior. We highlight several desirable benefits of our tool, especially in industrial settings where rapid experimentation, reusability, reproducibility, deployment, and maintenance are of great interest. Finally, we showcase a deployed service powered by BoolXAI as an enterprise application.

Code — <https://github.com/fidelity/boolxai>

Introduction

Machine Learning (ML) models exhibit ever-increasing complexity due to their sophisticated inner workings and the sheer scale of their parameter space. On the one hand, this contributes to the astonishing performance of ML models. On the other hand, it makes it difficult to understand and interpret their predictions. Explainability is highly desirable in many applications and is even mandatory in several domains such as finance and healthcare due to industry regulations (Freitas 2014; Holzinger et al. 2019; Stöger, Schneberger, and Holzinger 2021). From the perspective of Responsible AI, explainable models help discover superfluous patterns and avoid unwanted bias, and from the perspective of AI Product Lifecycle, explainable models are easier to deploy, debug, reproduce, maintain and improve over time.

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Explainable AI (XAI) has been studied extensively, providing the community with a rich literature. These include explanations in natural language processing (Liu, Yin, and Wang 2019; Strobelt et al. 2019), robotics (Brandão et al. 2021), reasoning (Junker 2004), planning (Borgo, Cashmore, and Magazzeni 2018), and vision (Kendall and Gal 2017). Our paper focuses on supervised machine learning, specifically, classification from tabular data, which is ubiquitous in high-stakes industrial applications.

In this paper, we present BoolXAI, an interpretable classification approach based on expressive Boolean formulas. In its simplest form, given a supervised dataset, the main idea behind BoolXAI is to find the smallest logical rule that satisfies the truth labels of the maximum number of samples. Under the hood, we formulate this as an optimization problem to design inherently interpretable classification models with tunable complexity. The formulation allows expressive Boolean operators that capture the classical conjunction (AND) and disjunction (OR) operators as well as ATMOST(), ATLEAST(), and CHOOSE() to generate logical rules that best explain the given dataset. To solve this optimization problem efficiently, we search the feasible space of all possible formulas using native local optimization.

More broadly, our approach is connected to integer linear programming (ILP) and Quadratic Unconstrained Binary Optimization (QUBO). As such, BoolXAI can be seen as an integration technology that brings together Boolean Satisfiability (SAT), Stochastic Local Search, ILP, and QUBO to provide a scalable and deployment-ready approach for XAI.

BoolXAI is designed and developed through a unique collaboration between academia (Brown & CalTech), financial technology (AI Center at Fidelity & Fidelity Center of Applied Technology), and the high-tech industry (Amazon Quantum Solutions & AWS Center of Quantum Computing). This collaboration allows us to meet the practical demands of industrial applications. We highlight several important built-in features offered by BoolXAI to address real-world scenarios often omitted in the research literature.

We showcase the high-level design and functionality of BoolXAI. We present the optimization formulation and our solution. We provide results from public datasets. Finally, we demonstrate an application powered by BoolXAI and deployed as a service to facilitate AI solutions in the enterprise.

High-Level Usage Example

Let us start by presenting a simple usage example to make the idea behind BoolXAI more concrete.

```
from boolxai import BoolXAI, Operator

# Supervised binary classification
X_train, y_train, X_test, y_test = data

# BoolXAI classifier with maximum depth,
# complexity, and Boolean operators
rule = BoolXAI.RuleClassifier(
    max_depth=3,
    max_complexity=6,
    operators=[Operator.And,
               Operator.Or,
               Operator.Choose,
               Operator.AtMost,
               Operator.AtLeast],
    metric=balanced_accuracy)

# Learn the best rule within constraints
rule.fit(X_train, y_train)

# Best Boolean rule
best_rule = rule_classifier.best_rule_
best_score = rule_classifier.best_score_
depth = best_rule.depth()
complexity = best_rule.complexity()

# Predict and evaluate
y_pred = rule.predict(X_test)
score = balanced_accuracy(y_test, y_pred)

# Visualization
best_rule.plot() # as in Figure 1
networkx_digraph = best_rule.to_graph()
```

In this example, given a sample of binary data X paired with binary labels y , we are tasked with a supervised binary classification problem. Let us discuss the extensions to numeric features, multi-class, and multi-label classification later on. Given this input, BoolXAI provides a high-level interface to train a rule-based classifier that is aimed at achieving the best score evaluated by the given `metric` within the constraints of `max_depth` and `max_complexity`. The Boolean rule can be composed of different operators including the classical conjunction (`And`) and disjunction (`Or`) as well as `AtMost`, `AtLeast`, and `Choose`. When run using only conjunction and disjunction, BoolXAI subsumes the previous literature on Boolean formulas for explainability specified in Conjunctive Normal Form (CNF).

Figure 1 shows the syntax tree of one formula, $\text{And}(\text{Choose2}(a, b, c, d), \sim e, f)$ to illustrate a BoolXAI rule. This rule contains six literals, i.e., features/columns of X , and two Boolean operators, `And` and `Choose` ($k = 2$).

As our usage example shows, the rule generation is constrained by parameters `max_depth` and `max_complexity`. The complexity is the number of operators and literals/features in the rule. The depth is the number of edges in the longest path from the root of the syntax tree of the rule to any leaf/literal. The example in Figure 1 has a complexity of eight with a depth of two.

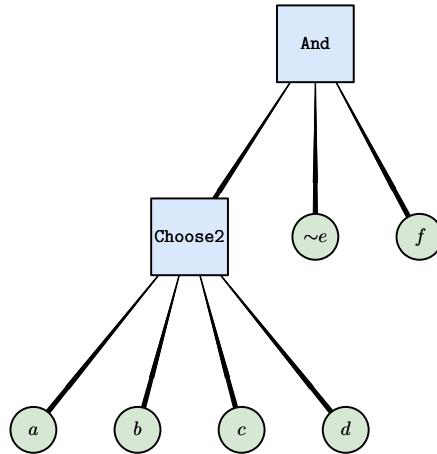


Figure 1: Visualization of an expressive Boolean formula, $\text{And}(\text{Choose2}(a, b, c, d), \sim e, f)$, that contains six literals (features) and two operators. It has a depth of two and a complexity of eight (the sum of literals and operators).

The `fit-and-predict` paradigm of BoolXAI follows the design pattern of the well-known scikit-learn library (Pedregosa et al. 2011). This should be familiar and easy to use for practitioners with a general background in data science. Our `RuleClassifier` is implemented by inheriting from the scikit-learn `BaseEstimator`. This makes it interoperable with all scikit-learn pipelines for hyper-parameter tuning and rapid experimentation. The `fit` step solves the underlying optimization problem to find the `best_rule`. The `best` is defined by the evaluation `metric` achieved on the given data respecting the maximum depth and complexity. The `predict` step evaluates the rule on a given sample(s) and returns the prediction(s).

Finally, users can plot the rules, as shown in Figure 1, or generate a directed `networkx` graph object with leaves corresponding to features connected with directed edges to the operator acting on them. In the prediction step, evaluating a rule on a given sample is accomplished by starting at the leaves, substituting the values from the features of the sample, and then applying the operators until reaching the root of the tree (formula).

Beyond this usage example, the `RuleClassifier` exposes several other parameters that power users can configure. This is not required for the out-of-the-box usage pattern showcased here. All BoolXAI parameters are set to sensible defaults. We discuss advanced usage later on when covering lessons learned and practical considerations.

Expressiveness of Boolean Operators

Now that we have demonstrated how to use BoolXAI, let's use a motivational example to highlight the expressive nature of our Boolean representations. Consider a simple scenario with five binary features that lead to a positive label only if at least three of the features hold. In BoolXAI, this is straightforward to state with a single rule as `AtLeast3(f0, ..., f4)`.

Decision Trees are generally considered highly interpretable, but they fail in this case. The result of training a decision tree for this simple scenario, leading to a surprisingly large tree with *19 split nodes*. The decision tree is deep and difficult to interpret. If a user faces it, it is hard to comprehend the global view of *AtLeast3* that the tree is trying to enforce.

Alternatively, instead of our expressive operators, if we only consider propositional formulas given in CNF using disjunctions and conjunctions, then we need a CNF over *13 clauses* using *11 variables* and a total of *29 literals* across the clauses. In comparison, our expressive Boolean formula offers a concise representation, as illustrated in this motivational example. Similarly, *AtMost* (k) provides an upper bound on the number of selected features, and *Choose* (k) enforces upper and lower bounds simultaneously.

One of our main motivations for studying XAI from the lens of logical formulas stems from their highly comprehensible nature. For instance, the inclusion of operators such as *AtLeast* is motivated by the idea of checklists (Zhang et al. 2024), as in medical lists used for symptoms for a particular condition where a minimum number of symptoms must be present for diagnosis (Christov et al. 2016). BoolXAI brings these succinct representations in a readily available tool for explainability in downstream applications.

BoolXAI Approach

So far, we only covered usage and motivation examples of BoolXAI. To be more precise, let us first define our problem statement and then present the algorithm behind our tool.

Originally, we covered this problem definition and the solution approach in (Rosenberg et al. 2023). In parallel, we released high-level industry blog posts to share the idea with practitioners^{1,2}. While our focus in this paper is on the BoolXAI as a tool, for completeness, we present technical details following the notation from (Rosenberg et al. 2023).

Definition 1 (Rule Optimization Problem (ROP)) Given a binary feature matrix \mathbf{X} and a binary label vector \mathbf{y} , the goal of the Rule Optimization Problem (ROP) is to find the optimum rule R^* that balances the score S of the rule R on classifying the data, and the complexity of the rule C , which is given by the total number of features and operators in R , bounded by a parameter C' .

Mathematically, our optimization problem can be stated at a high level as:

$$\begin{aligned} R^* = \arg \max_R [S(R(\mathbf{X}), \mathbf{y}) - \lambda C(R)] \\ \text{s.t. } C(R) \leq C' \end{aligned} \quad (1)$$

where R is any valid rule, and R^* , the optimized rule is the solution. The regularization parameter λ is a real number that controls the trade-off between the score and the complexity. This ROP definition is more generic than BoolXAI, which only considers Boolean representations for R .

¹<https://fcatalyst.com/blog/june2023/explainable-ai-using-expressive-boolean-formulas>

²<https://aws.amazon.com/blogs/quantum-computing/explainable-ai-using-expressive-boolean-formulas>

Objective: There are two competing objectives in this problem: maximizing performance (measured by S), and minimizing the complexity (measured by total number of literals and operators). The intuition is introducing more literals and operators makes rules less interpretable, hence the objective penalizes complexity. In BoolXAI, the objective is controlled by the `max_complexity`, `max_depth`, and `lambda` parameters. A common best practice is to explore the score vs. complexity trade-off by varying C' and λ over a range of values, producing a series of rules in the score-complexity space to discover the Pareto frontier.

Evaluation: In our usage example, we employed balanced accuracy for the performance metric, S . More precisely, it is equal to the mean of the accuracy of predicting each class:

$$S = \frac{1}{2} \left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right) \quad (2)$$

where T/F and P/N are the number of true/false positives/negatives. The motivation behind this metric is that many datasets are imbalanced in practice. BoolXAI `metric` parameter allows any scikit-learn style evaluation function.

Native Local Optimization for Solving ROP

As proposed in (Rosenberg et al. 2023), BoolXAI implements a native local optimization to solve ROP. Here, “native” refers to optimization in the natural search space of the problem, i.e., all valid expressive Boolean formulas. Our approach contrasts with reformulating the original problem in a fixed format such as MaxSAT, ILP, or QUBO. Next, “local” refers to exploring the search space via stochastic local search, i.e., by performing a series of moves that make relatively small changes to the current configuration.

Initialization: The initial BoolXAI rule is constructed by randomly choosing between two and $C' - 1$ literals without replacement. Literals are chosen for negation via a coin flip. Once the literals have been chosen, an operator and valid parameter (if the operator chosen is parameterized) are selected randomly. This is followed by applying local moves.

Local Moves: Given the initial configuration, the main search loop starts. We only consider feasible local moves via rejection sampling as follows. A node (literal or operator) in the current rule is chosen randomly. Then, a move type is chosen by cycling over the respective move types for the chosen literal/operator. Next, a random move of the chosen type is drawn. The process is restarted if the random move is invalid until a valid move is found. Typically, only a few iterations are needed.

Figure 2 illustrates the local moves when applied to the rule *And* (*Choose2* (a, b, c, d), $\sim e, f$) from Figure 1. The top row shows moves perturbing the literals, and the bottom row shows moves perturbing the operators.

For literal moves, BoolXAI implements the following:
L1: Remove literal—removes the chosen literal (d in Figure 2) only if the parent operator would not end up with fewer than two subrules. If the parent is a parameterized operator, it adjusts the parameter down (if needed) to remain valid after removing the chosen literal.

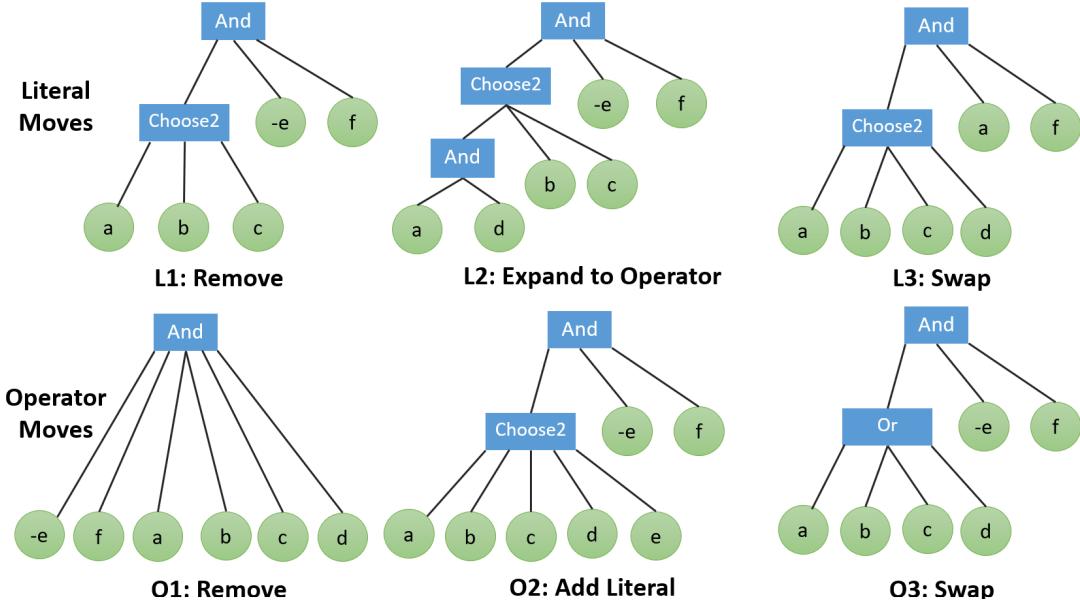


Figure 2: Local moves applied to literal nodes (top row) and operator nodes (bottom row) of the Boolean formula in Figure 1.

L2: Expand literal to operator—expands a chosen literal to an operator (*a* into *And* in Figure 2), moving a randomly chosen sibling literal to that new operator. It proceeds only if the parent operator includes at least one more literal. If the parent is a parameterized operator, it adjusts the parameter down (if needed) to remain valid after removing the chosen and sibling literal.

L3: Swap literal—replaces the chosen literal with a random literal ($\sim e$ with *a* in Figure 2) that is either the negation of the current literal or is a (possibly negated) literal that is not already included under the parent operator.

For operator moves, BoolXAI implements the following:

O1: Remove operator—removes an operator (*Choose2* in Figure 2) and any operators and literals under it. It only proceeds if the operator has a parent (i.e., it is not the root) and if the parent operator has at least three subrules such that the rule is still valid after the move has been applied.

O2: Add literal to operator—adds a random literal (possibly negated) to a given operator, but only if that variable is not already included in the parent operator (*e* added in Figure 2).

O3: Swap operator—swaps an operator for a randomly selected operator and parameter if the new operator is parameterized (*Choose2* with *Or* in Figure 2). It proceeds only if the new operator type or parameter is different.

BoolXAI Native Local Solver

Bringing the initial rule generation and local moves together, the BoolXAI solver starts by generating a new random rule `num_starts` times, which diversifies the search and is embarrassingly parallelizable. The main meta-heuristic behind our solver is based on Simulated Annealing (Kirkpatrick, Gelatt Jr, and Vecchi 1983). Each start begins by generating an initial random rule and continues with the proposal of `num_iterations` local moves.

Moves are accepted based on a Metropolis criterion, such that the acceptance probability of a proposed move $P(\Delta F, T) = \min(1, e^{-\Delta F/T})$ depends only on the objective function change ΔF and temperature T . Initially, the temperature is high, leading to most moves being accepted (exploration), regardless of the ΔF value. The temperature is decreased on a geometric schedule, such that in the latter stages of each start, the solver accepts only descending or “sideways” moves—moves that do not change the objective function (exploitation), i.e., moves for which $\Delta F \leq 0$.

BoolXAI QUBO & ILP Solvers

In (Rosenberg et al. 2023), we expand this local solver with “non-local” moves, similar to LNS (Shaw 1998), that optimize an *entire subtree* of the current formula based on exact QUBO and ILP formulations. Further, these non-local moves are compatible with quantum algorithms, potentially powered by special-purpose hardware or quantum devices. We omit these details here and focus on the practical aspects.

Illustrative Results

Table 1 shows the rules found by applying BoolXAI native local solver with `max_complexity = 4` to well-known UCI ML datasets from the airline industry, healthcare, finance, marketing, real estate, and e-commerce. These problems have varied characteristics, with the number of samples ranging from 195 to 130,000 and the number of binarized features ranging from 67 to 300.

When we compare BoolXAI with the single-best feature, most-frequent feature, decision trees, CNF rules, ILP rules, and QUBO rules, it performs competitively regarding runtime, striking a neat balance between model performance and complexity.

Dataset	Expressive BoolXAI Formula	Accuracy
Airline Cust. Satisfaction	And (Inflight entertainment \neq 5, Inflight entertainment \neq 4, Seat comfort \neq 0)	76%
Breast Cancer	AtMost1 (worst concave \leq 0.15, worst radius \leq 16.43, mean texture \leq 15.30)	95%
Credit Card Default	Or (PAY_2 > 0, PAY_0 > 0, PAY_4 > 0)	71%
Credit Risk	Choose1 (status = no checking, status < 200, property_magnitude = real estate)	70%
Customer Churn	AtMost1 (tenure > 5, Contract \neq Month-to-month, InternetService \neq Fiber optic)	76%
Direct Marketing	Or (duration > 393, nr.employed \leq 5076.2, month = mar)	86%
Home Equity Default	Or (DEBTINC > 41.62, DELINQ \neq 0.0, CLNO \leq 11)	71%
Online Shopper Intent	AtMost1 (PageValues \leq 5.55, PageValues \leq 0, BounceRates > 0.025)	86%
Parkinson's	AtMost1 (spread1 > -6.30, spread2 > 0.19, Jitter:DDP > 0.06)	90%

Table 1: BoolXAI rules obtained by the native local solver with `max_complexity = 4` across well-known UCI ML datasets.

On average, *BoolXAI* achieves 80% balanced accuracy with a single Boolean formula of complexity four (one operator and three features). The details of each dataset, the experimental setup, and the comparison with other methods can be found in (Rosenberg et al. 2023).

Practical Considerations & A Case Study

The tool support for XAI trails behind the growing research literature, primarily focusing on methodology. This section highlights several practical aspects that BoolXAI considers for end-users. Thanks to our unique industry/academia partnership, we built BoolXAI to ease development, deployment, and maintenance, emphasizing scalability, extensive documentation, heavy testing, sensible defaults, and error handling to guide users.

The primary users of BoolXAI are data scientists and practitioners with a general background in machine learning. The library is indexed in PyPI for general availability, allowing a seamless installation with the `pip install boolxai` command. At Fidelity, it is available to 100+ data scientists and has been downloaded 2000+ times in the broader community since its recent launch in Q4 2024, and is growing.

Let us share our key findings and lessons from a particular BoolXAI case study. The State of Wealth Mobility³ is recently published to better understand public opinion on wealth management and financial planning. One main takeaway from this extensive study is the importance of financial literacy, as the majority (81%) say they would have benefited from financial education earlier. In accordance with this, the Learn Center⁴ aims to provide financial articles, webinars, newsletters, and more to the broader public. Specifically, long-form viewpoint articles offer a free newsletter with news and insights from finance professionals about markets, investing, and personal finance. Further details on the case study for article recommendations can be found in (Verma et al. 2023).

We applied BoolXAI to this dataset led by a small team of data scientists to understand the factors in engagement better. Our key findings and lessons learned are:

³<https://preview.thenewsmarket.com/Previews/FINP/DocumentAssets/682671.pdf>

⁴<https://www.fidelity.com/learning-center/overview>

Complexity: As expected, among the most critical parameters for the interpretability were depth (`max_depth`) and complexity (`complexity`). Users mostly kept complexity less than 15 with a max depth of less than 5.

Robustness: One confusion of the users stemmed from BoolXAI finding various rules that quite differ from each other but perform similarly in terms of metrics (e.g., balanced accuracy). Empirically, users found that it stabilizes the best rule when we first identify frequent features exhibited in the top-10 best results and then run BoolXAI only with those features. We recommend this method in practice.

Visualization: Users reported relying on BoolXAI's visualization functionality often (as in Figure 1) to interpret the resulting rules and share findings with non-technical users.

Runtime: Running BoolXAI on a dataset of 500,000 rows and 100 columns takes about 60 seconds on a modern laptop with an Intel i7 2.20 GHz processor using a single core.

By default, this runs 2000 samples (`max_samples`) for 500 iterations (`max_iterations`) with 10 restarts (`num_starts`). This translates into ~ 0.01 sec per rule optimization iteration. Most users did not utilize parallelization (`n_jobs`), explained by quick iteration time, and most of the time was spent interpreting the generated rules.

Predefined User Rules: To our surprise, users were seldom interested in running BoolXAI on the given data as-is. Defining a pre-determined rule based on their existing business understanding mattered more. Given this feedback, we designed a BoolXAI functionality to optimize only part of a rule (`base_rule`). Users select the features of the base rule, and then BoolXAI optimizes the *rest of the formula*, keeping the base rule fixed. This is akin to bootstrapping but human-guided, allowing users to test hypotheses quickly. BoolXAI can also be fed into `BoostingClassifier` for boosting and iterative training of multiple rules focusing on the most challenging samples to predict next.

Incremental Rule Generation: A similar usage pattern was typical to build formulas incrementally. By default, BoolXAI reports the single best feature (`baseline`). Users leveraged the single best rule as the predefined rule and optimized the remainder of the formulae. This yielded the subsequent feature(s) to include in the next iteration. Users enjoyed the controlled nature of iterative formula generation, where they introduced features incrementally, decided the

rule’s complexity, and when to stop. The fast runtime of BoolXAI helped these iterative scenarios.

MLOps Pipelines: Data scientists highly praised the scikit-learn interoperability of BoolXAI. This allowed leveraging BoolXAI models in existing pipelines and hyper-parameter tuners as the base sklearn classifiers. Unlike existing methods that require a specialized optimization solver in the backend, our native local optimization is designed to depend only on numpy and sklearn. This eased installation, deployment and maintenance over time.

Non-Binary Data: Users benefited from the default binarization via `BoolXAIKBinsDiscretizer` to deal with numerical features. This allowed configuring the discretization behavior via the encoding method (e.g., one-hot, one-hot-dense, or ordinal) and binning strategy (e.g., uniform, quantile, kmeans). The advantage of this built-in utility, in contrast to standard discretizers, is the preservation of feature names, which is crucial in visualization and interpretation. Beyond binary tasks, BoolXAI classifiers can be fed into built-in scikit-learn utilities. When multi-class input was necessary, users relied on `1vs1`, `1vsRest`, `OutputCode` classifiers from scikit-learn. Likewise, for multi-label input, `MultiOutputClassifier` was used thanks to scikit-learn compatibility. To further increase interpretability, BoolXAI automatically post-processes the best rule to flatten the logic by removing nested operations, where possible, similar to De Morgans’ law.

Coding Standards & Testing: Let us note that the library adheres to the PEP-8 style guide for Python coding standards and complies with `numpydoc` documentation standards. All available functionality is tested via standalone unit tests to verify the correctness of the algorithms, including invalid cases. The test suite provides more than 95% code coverage. The source code is peer-reviewed for both architecture and implementation. The implementation pays special attention to ensuring reproducibility, a highly desirable feature in industrial applications. Sensible defaults and strict error checking of input parameters helped users avoid simple mistakes.

Documentation & Examples: Users benefited from the well documented⁵ library. Public methods have source code documentation, including their arguments, default parameter settings, return values, and exception cases. The library supports typing to provide the user with argument hints and annotations.

EASE: Explainability-as-a-Service

Even with our best efforts on software usability, BoolXAI, as a library, remains as a tool for practitioners. Yet, explainability is in high demand by business stakeholders. To make XAI accessible to broader audiences, we built EASE: Explainability-as-a-Service, powered by BoolXAI.

EASE is an interactive web service that does not require programming. Users can upload their input data (or its path on Amazon S3) and examine the BoolXAI rules and visualizations. As explained earlier, users can seed their assump-

tions or extract base rules from previous runs to incrementally re-optimize the rest of the formula. One of the strong assumptions of BoolXAI is that the input features are inherently interpretable. EASE relaxes this assumption by offering a set of attributes commonly used in segmentation and analytics to its users out of the box.

For non-technical users, EASE allowed business partners to input their observational data and discover correlations based on readily available common interpretable attributes. This helped our business users form hypotheses to be tested further to explain observed behaviors.

For technical users, who have access to sophisticated machine learning methods operating on high-dimensional data, an unintended behavior emerged. BoolXAI provided data scientists with a lower bound on expected performance as in Table 1. This lower bound was then utilized as a delta to validate the performance achieved by complex methods. The delta gap provided a principled approach to reconsider additional complexity when making deployment decisions.

Related Work

Broadly, the two main approaches to XAI are explaining black boxes and training interpretable models. Most state-of-the-art models, particularly in deep learning, are sophisticated with many parameters. For these models, a common approach is to provide post-hoc explanations in a model-agnostic manner. These methods include local and global explanations. For local explanations, prominent tools include LIME (Ribeiro, Singh, and Guestrin 2016), SHAP (Lundberg and Lee 2017), and MUSE (Lakkaraju et al. 2019). These methods work with a local approximation based on a group of instances, hence, lack global robustness and are vulnerable to adversarial attacks (Slack et al. 2020; Lakkaraju, Arsov, and Bastani 2020). Global explanations mimic black-box models using interpretable models (Craven and Shavlik 1995; Bastani, Kim, and Bastani 2017). A drawback of global methods is ambiguity when widely different interpretations yield similar fidelity. To train interpretable models, apart from the well-known decision trees and linear regression, many approaches have been studied, including rule lists (Letham et al. 2015), falling-rule lists (Wang and Rudin 2015a), decision sets (Lakkaraju, Bach, and Leskovec 2016), and scoring systems (Ustun and Rudin 2016).

The closest to our approach are combinatorial methods such as learning CNF rules using MaxSAT (Malioutov and Meel 2018; Ghosh and Meel 2019), ILP (Su et al. 2015; Wang and Rudin 2015b; Lawless et al. 2021), and LP (Malioutov et al. 2017; Akyüz and Birbil 2021). BoolXAI is based on local moves instead of exact methods that create a computational bottleneck on large datasets. Our solution performs native optimization without dependency on a specialized solver in the backend, which eases the deployment. BoolXAI is designed to be interoperable with the Python ML stack for experimentation. In numerous cases, interpretable models have been shown to yield comparable results to black-box models (Rudin 2019), and our work contributes to this line of research.

⁵<https://github.com/fidelity/BoolXAI/tree/main/notebooks>

References

- Akyüz, M. H.; and Birbil, S. I. 2021. Discovering Classification Rules for Interpretable Learning with Linear Programming. *CoRR*, abs/2104.10751.
- Bastani, O.; Kim, C.; and Bastani, H. 2017. Interpreting blackbox models via model extraction. *arXiv 1705.08504*.
- Borgo, R.; Cashmore, M.; and Magazzeni, D. 2018. Towards Providing Explanations for AI Planner Decisions. *CoRR*, abs/1810.06338.
- Brandão, M.; Canal, G.; Krivic, S.; and Magazzeni, D. 2021. Towards providing explanations for robot motion planning. In *IEEE Int. Conf. on Robotics and Automation, ICRA 2021*.
- Christov, S. C.; Conboy, H. M.; Famigletti, N.; Avrunin, G. S.; Clarke, L. A.; and Osterweil, L. J. 2016. Smart Checklists to Improve Healthcare Outcomes. In *2016 IEEE/ACM Int. Workshop on Software Engineering in Healthcare*.
- Craven, M.; and Shavlik, J. 1995. Extracting tree-structured representations of trained networks. *Advances in neural information processing systems*, 8.
- Freitas, A. A. 2014. Comprehensible classification models: a position paper. *SIGKDD Explor. Newsl.*, 15(1): 110.
- Ghosh, B.; and Meel, K. S. 2019. IMLI: An incremental framework for MaxSAT-based learning of interpretable classification rules. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 203–210.
- Holzinger, A.; Langs, G.; Denk, H.; Zatloukal, K.; and Müller, H. 2019. Causability and explainability of artificial intelligence in medicine. *WIREs Data Mining and Knowledge Discovery*, 9(4): e1312.
- Junker, U. 2004. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In McGuinness, D. L.; and Ferguson, G., eds., *AAAI*, 167–172.
- Kendall, A.; and Gal, Y. 2017. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In *Advances in Neural Information Processing Systems*.
- Kirkpatrick, S.; Gelatt Jr, C. D.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science*, 220(4598): 671–680.
- Lakkaraju, H.; Arsov, N.; and Bastani, O. 2020. Robust and stable black box explanations. In *International Conference on Machine Learning*, 5628–5638. PMLR.
- Lakkaraju, H.; Bach, S. H.; and Leskovec, J. 2016. Interpretable decision sets: A joint framework for description and prediction. In *Proc. of ACM SIGKDD international conference on knowledge discovery and data mining*, 1675–1684.
- Lakkaraju, H.; Kamar, E.; Caruana, R.; and Leskovec, J. 2019. Faithful and customizable explanations of black box models. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 131–138.
- Lawless, C.; Dash, S.; Gunluk, O.; and Wei, D. 2021. Interpretable and Fair Boolean Rule Sets via Column Generation. *arXiv preprint*, 2111.08466.
- Letham, B.; Rudin, C.; McCormick, T. H.; and Madigan, D. 2015. Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 9(3): 1350–1371.
- Liu, H.; Yin, Q.; and Wang, W. Y. 2019. Towards Explainable NLP: A Generative Explanation Framework for Text Classification. In *Proc. of ACL*, 5570–5581.
- Lundberg, S. M.; and Lee, S.-I. 2017. A unified approach to interpreting model predictions. *NeurIPS*, 30.
- Malioutov, D.; and Meel, K. S. 2018. MLIC: A MaxSAT-based framework for learning interpretable classification rules. In *CP*, 312–327. Springer.
- Malioutov, D. M.; Varshney, K. R.; Emad, A.; and Dash, S. 2017. Learning interpretable classification rules with Boolean compressed sensing. In *Transparent Data Mining for Big and Small Data*, 95–121. Springer.
- Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *JMLR*, 12: 2825–2830.
- Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. Why should I trust you? Explaining the predictions of any classifier. In *Proc. of ACM SIGKDD*, 1135–1144.
- Rosenberg, G.; Brubaker, J. K.; Schuetz, M. J. A.; Salton, G.; Zhu, Z.; Zhu, E. Y.; Kadolu, S.; Borujeni, S. E.; and Katzgraber, H. G. 2023. Explainable Artificial Intelligence Using Expressive Boolean Formulas. *Machine Learning and Knowledge Extraction*, 5(4): 1760–1795.
- Rudin, C. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5): 206–215.
- Shaw, P. 1998. Using Constraint Programming and Local Search Methods to Solve VPR. In *CP*, 417–431.
- Slack, D.; Hilgard, S.; Jia, E.; Singh, S.; and Lakkaraju, H. 2020. Fooling LIME and SHAP: Adversarial attacks on post hoc explanation methods. In *Proc. of ACM AIES*, 180–186.
- Stöger, K.; Schneeberger, D.; and Holzinger, A. 2021. Medical artificial intelligence: the European legal perspective. *Commun. ACM*, 64(11): 34–36.
- Strobelt, H.; Gehrmann, S.; Behrisch, M.; Perer, A.; Pfister, H.; and Rush, A. M. 2019. Seq2seq-Vis: A Visual Debugging Tool for Sequence-to-Sequence Models. *IEEE Trans. Vis. Comput. Graph.*, 25(1): 353–363.
- Su, G.; Wei, D.; Varshney, K. R.; and Malioutov, D. M. 2015. Interpretable two-level Boolean rule learning for classification. *arXiv preprint*, 1511.07361.
- Ustun, B.; and Rudin, C. 2016. Supersparse linear integer models for optimized medical scoring systems. *Machine Learning*, 102(3): 349–391.
- Verma, G.; Sengupta, S.; Simanta, S.; Chen, H.; Perge, J. A.; Pillai, D.; McCrae, J. P.; and Buitelaar, P. 2023. Empowering RecSys using automatically generated KG and RL. *arXiv:2307.04996*.
- Wang, F.; and Rudin, C. 2015a. Falling rule lists. In *Artificial intelligence and statistics*, 1013–1022. PMLR.
- Wang, T.; and Rudin, C. 2015b. Learning optimized Or's of And's. *arXiv preprint*, 1511.02210.
- Zhang, H.; Morris, Q.; Ustun, B.; and Ghassemi, M. 2024. Learning optimal predictive checklists. In *NeurIPS*, 2021.