Towards Modern Rule-Based Learning

Giovanni **Pagliarini**[1], Edoardo **Ponsanesi**[1], Guido **Sciavicco**[1] and Ionel Eduard **Stan**[2]

[1]*University of Ferrara, Italy*

[2]*University of Milano - Bicocca, Italy*

Abstract

Symbolic classification is a subfield of symbolic learning focused on extracting a collection of mutually exclusive logical rules for classification. This is typically achieved by learning intermediate models, such as decision trees or decision lists. In this paper, we present *Modal Sequential Covering* (*MSC*), a decision list learning algorithm that generalizes several existing proposals in the literature. We also provide an early open-source implementation of this algorithm in Julia, integrated within *Sole*, a comprehensive end-to-end framework for modern symbolic AI. An experimental comparison with available tools reveals that MSC allows us to learn simpler but equally performant models. The integration of MSC into *Sole* enables manipulating and visualizing the extracted knowledge in logical form. As *Sole* is designed for symbolic learning with modal and propositional logics, this work lays the foundation for further generalization to non-tabular data.

Keywords

Symbolic Learning, Decision List Learning, Rule Extraction

1. Introduction

Recent developments in machine learning have driven attention to black-box models, particularly for classification. This trend has led, in turn, to a demand for interpretation and explanation methods. However, some argue [1] that developing algorithms that inherently learn interpretable white-box models is preferable: symbolic learning, the subfield of machine learning concerned with extracting logical formulas from data, presents one alternative, offering intelligible and actionable models.

Symbolic classification seeks to extract a set of mutually exclusive logical rules for decision-making:

$$\Gamma = \{\varphi_1 \Rightarrow L_1, \ldots, \varphi_z \Rightarrow L_z\},$$

where each φ_i represents a rule's condition, and L_i is its corresponding class. The objective is to ensure that only one rule will fire for any pair of rules in any input instance. Barring minor variations, there are two high-level learning models in the symbolic context, that is, *decision trees* and *decision lists*; such simple models can be combined in several forms of ensemble models, stacked models, and similar pseudo-functional strategies. Decision trees resemble nestings of IF-ELSE constructs, and they are typically built in recursive manner; when branches are read from top to bottom, they are immediately interpreted as mutually exclusive rules, that is, a rule set. Decision lists are generally obtained as the result of some type of *sequential covering* algorithm, that produces a collection of rules, paired with a firing policy; typically, the firing policy is induced by an ordering of the rules, ultimately making the models similar to an IF-ELSEIF-ELSE construct. A rule set, as we have defined it, is then obtained by grafting the policy into the rules, so to render them mutually exclusive.

The current landscape of well-known packages that offer accepted implementations of decision trees and/or decision list learning algorithms includes, among others: *Weka*, in Java, which offers both decision tree (C4.5 [2]) and decision list learning algorithms (RIPPER [3], ONER [4]); *Scikit-learn*, in Python, which offers a decision tree learning algorithm (CART [5]); *Wittgenstein*, again in Python,
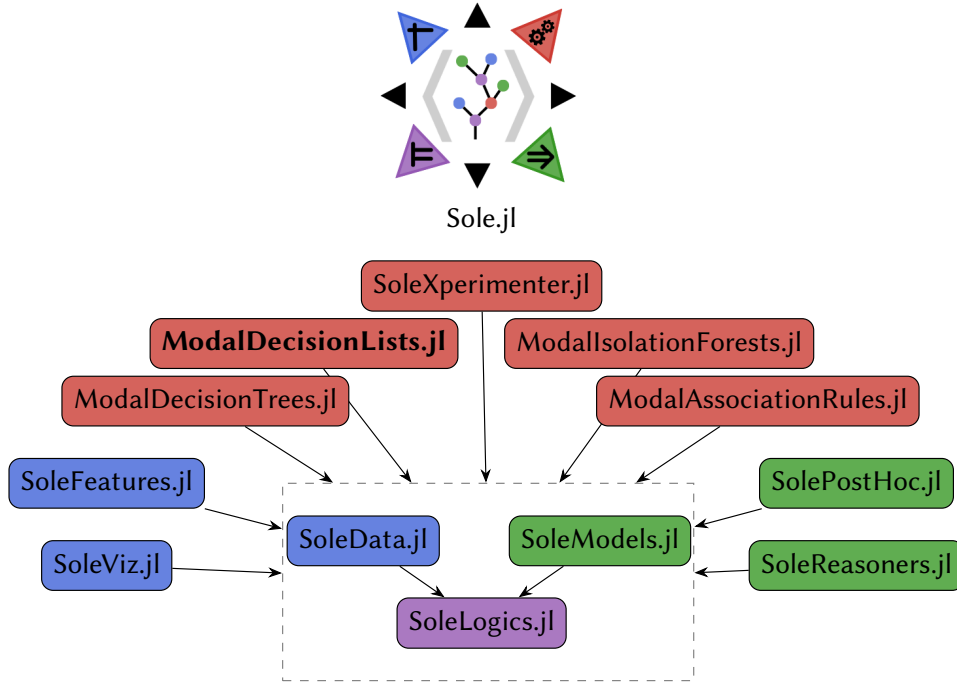
Figure 1: Structure of the Sole.jl framework for symbolic AI. Packages in green provide tools for manipulating logical formulas; packages in red provide tools for (symbolic) data processing; those in blue provide tools for learning symbolic models; those in purple provide tools for manipulating (symbolic) models. This work showcases an effort towards building the *ModalDecisionLists* package.

which provides a decision list algorithm (IREP [6], RIPPER); *Orange*, in Python, with both decision trees and lists (C4.5, CN2 [7]); *Chefboost*, in Python, with decision trees (ID3 [8]), *supervisedPRIM* and *oneR*, in R, with implementations for decision lists (PRIM [9], ONER), and *MLJ*, in Julia, which offers a decision list learning algorithm (CART). While decision trees and similar techniques have received attention even in recent years (see, e.g. [10], among many others), it does not seem to be the case with rule covering, with the possible exception of explanation methods (see, e.g. [11]) or Tsetlin machines (see, e.g. [12]). A common characteristics of all mentioned programming frameworks, symbolic algorithms, and implementations, is that they do not leverage nor highlight the logical representations of the extracted knowledge. As such, the learned rules cannot be manipulated within a logical framework, and the algorithms themselves are not easily generalizable to more expressive logical languages.

Sole.jl (or, simply, *Sole*) [13] is a open-source framework, written in the Julia programming language, that allows one to design and deploy end-to-end learning tasks, from data preprocessing and cleaning, filter-based feature extraction and selection, symbolic learning models training, testing, inspection, and post-hoc modification, and result visualization (a schema of Sole is shown in Fig 1). Sole is completely logic-based, and integrates several reasoning tools that can be paired up and used on the learned rules. As much as decision tree learning is concerned *Sole* already includes an implementation (MCART [14, 15]) that allows one to learn a decision tree from tabular and non-tabular data (using a suitable *modal logic*), and to manipulate the resulting rules in a comprehensive logical framework; modal logic extends propositional logic, and, as it emerges from several experiments on real-world data, can be applied to extract useful, non-trivial knowledge (see, e.g., [16, 17, 18]).

In this paper, we describe the implementation of the algorithm *Modal Sequential Covering* (*MSC*), which, in a nutshell, is the Julia implementation of a sequential covering algorithm that is based on, and generalizes several well-known algorithms, including RIPPER, CN2, ONER, and IREP. Decision lists learned with MSC may have rules whose literals are not just atomic statements but may be complex formulas; the covering can be heuristically driven by fully parameterizable beam search and be based on one of several available information measures; and, candidate literals can be explored via determinis-

Algorithm 1: Modal sequential covering algorithm. The input encompasses a dataset $\mathcal{I}$, a parametrization $\mathcal{B}$ (resp., Π, $\mathcal{G}$) for the beam search (resp., stopping conditions, search space and strategy). FindBestConjunction deploys a (CN2-like) beam search heuristic to learn conjunctions of formulas from a grammar; BestGuess computes the majority class of a dataset.

```
function MSC(I, Π, B, G):
    rules ← []                                          ◁ Initialize decision list
    I_uncov ← I                                         ◁ Initialize sub-dataset to be covered
    while |I_uncov| > 1 do
        (φ, I^φ_uncov, I^¬φ_uncov) ← FindBestConjunction(I_uncov, B, G)
        rule ← (φ ⇒ BestGuess(I^φ_uncov))               ◁ Create best rule
        if a Π-stopping condition applies then break
        rules.insert(rule)                              ◁ Add rule to list
        I_uncov ← I^¬φ_uncov                            ◁ Update sub-dataset to be covered
    end
    rule_default ← (⊤ ⇒ BestGuess(I_uncov))             ◁ Create best default rule
    rules.insert(rule_default)                          ◁ Add default rule to list
    return rules
end
```

tic/randomized search heuristics. An early implementation of MSC is available in the *ModalDecisionLists* package, which is a work-in-progress package part of the *Sole* learning suite. Moreover, consistently with the nature of *Sole*, a modal decision list is a rule-based classification model that is able to cope with non-tabular data as well, using a modal logic to replace the classical propositional one; while the current implementation of does not allow the use of modal logic yet, it contains all necessary elements for its immediate generalization.

2. Modal Sequential Covering

The most widely used algorithms for decision list learning follow the so-called *sequential covering* approach, that is, a separate-and-conquer method that learns rules sequentially. First, a set of uncovered instances is initialized as the set of all instances in the dataset; then, at each iteration of a loop, a single rule is learned on the uncovered instances by optimizing a loss function, and the instances covered by the rule is removed from the set of uncovered instances. Typically, a rule $\varphi \Rightarrow L$ built at a given iteration is an object whose antecedent φ is given by a conjunction of conditions over the attributes.

A conjunction is progressively built via heuristic approaches that explore new conjuncts to be added, starting with an empty formula. For example, CN2 [7] adopts a beam search approach over the space of conjunctions, while RIPPER [3] specializes a single conjunction by selectively choosing the best condition to add (which is equivalent to single-beam search approach). The methods mainly differ from each other in terms of the loss function to be optimized (e.g., minimum description length, information gain, Laplace accuracy), as well as the regularization strategies deployed (e.g., pruning and stopping criteria). RIPPER, for example, performs a training/validation split prior to the learning, and after iteratively growing each conjunction on the training set, it uses the validation set to post-prune it; moreover, it prioritizes the least numerous classes, so that classes are covered, in order, by their numerosity.

With a view of lifting the sequential covering approach to more-than-propositional logics (e.g., modal logics), these premises inspire the design of a more general sequential covering algorithm, namely *Modal Sequential Covering* (MSC), that iteratively learns conjunctions of type $\psi_1 \wedge \ldots \wedge \psi_n$, with ψ_i belonging to a (parametrizable) grammar; this extension is designed to trivially include the above algorithms where, for comparison, the conjuncts are limited to be atoms. MSC retains the list-level separate-and-conquer and rule-level beam search heuristic approaches, but extends the search space of conjuncts to a grammar, and deploys a parametrizable search heuristic, both of which are hyperparameters of the algorithm.

The pseudo-code of MSC is shown in Alg. 1.

Table 1
Cross-validation results showing κ coefficient, training time, number of atoms in the model, and average atoms per rule.

Dataset	Implementation	κ	time (seconds)	# atoms	# atoms/rule
Biopsy	Sole-RANDCOV	0.918 ± 0.02	1.85 ± 0.25	31	17.6
	Sole-CN2	0.915 ± 0.02	0.69 ± 0.06	14	8.3
	Orange-CN2	0.919 ± 0.02	0.19 ± 0.02	25	15.5
	Wittgenstein-RIPPER	0.901 ± 0.05	0.16 ± 0.00	14	2.3
	Scikit-CART	0.909 ± 0.04	0.001 ± 0.00	17	5.1
	MLJ-CART	0.872 ± 0.04	$1.11 \cdot 10^{-5} \pm 0.00$	13	37.0
Ionosphere	Sole-RANDCOV	0.795 ± 0.01	0.91 ± 0.24	15	9.1
	Sole-CN2	0.798 ± 0.04	21.98 ± 0.24	13	7.2
	Orange-CN2	0.822 ± 0.06	42.23 ± 0.15	22	13.6
	Wittgenstein-RIPPER	0.707 ± 0.01	0.50 ± 0.00	14	1.6
	Scikit-CART	0.799 ± 0.03	0.01 ± 0.00	3	1.7
	MLJ-CART	0.766 ± 0.04	$1.12 \cdot 10^{-5} \pm 0.00$	4	9.6
MobilePrice	Sole-RANDCOV	0.799 ± 0.02	45.17 ± 3.27	172	88.2
	Sole-CN2	0.795 ± 0.02	77.28 ± 0.16	213	104.1
	Orange-CN2	0.774 ± 0.02	31.76 ± 0.07	268	135.9
	Wittgenstein-RIPPER	$-$	$-$	$-$	$-$
	Scikit-CART	0.811 ± 0.00	0.02 ± 0.00	72	6.6
	MLJ-CART	0.821 ± 0.01	$1.12 \cdot 10^{-5} \pm 0.00$	123	484.3
Yeast	Sole-RANDCOV	0.465 ± 0.01	31.63 ± 4.81	157	84.2
	Sole-CN2	0.456 ± 0.02	2.54 ± 0.13	464	250.4
	Orange-CN2	0.448 ± 0.02	1.62 ± 0.00	307	181.9
	Wittgenstein-RIPPER	$-$	$-$	$-$	$-$
	Scikit-CART	0.443 ± 0.02	0.003 ± 0.00	13	1.7
	MLJ-CART	0.459 ± 0.04	$1.11 \cdot 10^{-5} \pm 0.00$	52	174.4
Abalone	Sole-RANDCOV	0.169 ± 0.02	38.29 ± 0.91	108	57.9
	Sole-CN2	0.163 ± 0.03	95.08 ± 0.23	984	537.6
	Orange-CN2	0.162 ± 0.01	628.10 ± 1.37	153	85.2
	Wittgenstein-RIPPER	$-$	$-$	$-$	$-$
	Scikit-CART	0.173 ± 0.01	0.002 ± 0.00	17	3.4
	MLJ-CART	0.162 ± 0.01	$1.14 \cdot 10^{-5} \pm 0.00$	97	334.8

3. Experiments and Results

We evaluated our implementation on five tabular datasets: Biopsy [19], Ionosphere [20], MobilePrice [21], Yeast [22], and Abalone [23]. All datasets, except MobilePrice, are publicly available on the UCI Machine Learning Repository [24], while MobilePrice is available via Kaggle.

We compared the following methods: MSC (Sole) with the RANDCOV profile (*Sole-RANDCOV*); MSC (Sole) with the CN2 profile (*Sole-CN2*); CN2 from the Orange Python package (*Orange-CN2*); RIPPER from the Wittgenstein Python package (*Wittgenstein-RIPPER*); CART from scikit-learn (*Scikit-CART*); CART from the DecisionTree.jl Julia package (*MLJ-CART*). While the first two methods are based on MSC, Sole-CN2 behaves similarly to Orange-CN2, and Sole-RANDCOV introduces novelty by generating random propositional conjuncts. For each method, hyperparameter tuning was performed using grid search, optimizing for Cohen's κ coefficient.

The results are summarized in Tab. 1. First, MSC achieves competitive accuracy (in terms of the κ coefficient) with state-of-the-art methods. Focusing on comparing, in particular, Sole-CN2 against Sole-RANDCOV, the latter produces classifiers with less atoms in 3 out of 5 cases; in one case, Ionosphere the number of atoms is similar (14 vs 15). In the cases in which the random search produces better results in terms of number of atoms, the difference is very relevant (172 vs 213, 157 vs 464, and 108 vs 984); in all of them, Sole-RIPPER outperforms Orange-CN2 as well. Moreover, observe that Wittgenstein-RIPPER fails in 3 out of 5 cases, as it cannot handle more-than-binary problems. Finally, decision trees such as those produced by MLJ-CART or Scikit-CART are presented for reference, as there is no direct comparison between the two approaches.

Overall, our results show that MSC is a robust addition to the existing symbolic learning approaches, providing a balance between performance and rule complexity. The implementation in Sole makes it highly flexible, allowing the exploration of advanced symbolic learning techniques in future work.

4. Conclusions

In this paper, we have introduced the Modal Sequential Covering (MSC) algorithm, a generalization of existing decision list learning approaches, and its implementation within the Sole framework. While MSC currently operates in propositional logic, it is designed to seamlessly extend to modal logic formulas, which is particularly relevant given the success of using modal logics in symbolic learning for extracting useful, non-trivial knowledge from real-world data, as demonstrated in [16, 17, 18].

The integration of MSC into Sole provides a highly flexible platform, not only for working with traditional decision lists but also for exploring complex symbolic learning tasks using modal logics.

Acknowledgments

References

[1] C. Rudin, Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, Nature Machine Intelligence 1 (2019) 206–215.

[2] J. R. Quinlan, Generating production rules from decision trees, in: Proceedings of the 10th International Joint Conference on Artificial Intelligence., Morgan Kaufmann, 1987, pp. 304–307.

[3] W. W. Cohen, Fast effective rule induction, in: Proc. of the 12th International Conference on Machine Learning, Morgan Kaufmann, 1995, pp. 115–123.

[4] R. C. Holte, Very simple classification rules perform well on most commonly used datasets, Machine Learning 11 (1993) 63–91.

[5] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and regression trees, Wadsworth Publishing Company, 1984.

[6] J. Fürnkranz, G. Widmer, Incremental reduced error pruning, in: Proc. of the 11th International Conference on Machine Learning„ Morgan Kaufmann, 1994, pp. 70–77.

[7] P. Clark, T. Niblett, The CN2 Induction Algorithm, Machine Learning 3 (1989) 261–283.

[8] J. Quinlan, Induction of Decision Trees, Machine Learning 1 (1986) 81–106.

[9] J. H. Friedman, N. I. Fisher, Bump hunting in high-dimensional data, Statistics and computing 9 (1999) 123–143.

[10] A. Schidler, S. Szeider, Sat-based decision tree learning for large data sets, Journal of Artificial Intelligence Research 80 (2024) 875–918.

[11] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, S. Lee, From local explanations to global understanding with explainable ai for trees, Nature machine intelligence 2 (2020) 56–67.

[12] K. Abeyrathna, O.-C. Granmo, R. Shafik, L. Jiao, A. Wheeldon, A. Yakovlev, J. Lei, M. Goodwin, A multi-step finite-state automaton for arbitrarily deterministic tsetlin machine learning, Expert Systems - the Journal of Knowledge Engineering 40 (2023).

[13] F. Manzella, G. Pagliarini, A. Paparella, G. Sciavicco, I. E. Stan, Sole.jl – Symbolic Learning in Julia, https://github.com/aclai-lab/Sole.jl, 2024.

[14] D. Della Monica, G. Pagliarini, G. Sciavicco, I. E. Stan, Decision trees with a modal flavor, in: Proc.

of the 21st International Conference of the Italian Association for Artificial Intelligence (AIxIA), volume 13796 of *LNCS*, Springer, 2022, pp. 47–59.

[15] F. Manzella, G. Pagliarini, G. Sciavicco, I. E. Stan, Efficient modal decision trees, in: Proc. of the 22th Conference of the Italian Association for Artificial Intelligence (AIxIA), volume 14318 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 381–395.

[16] F. Manzella, G. Pagliarini, G. Sciavicco, I. E. Stan, The voice of COVID-19: breath and cough recording classification with temporal decision trees and random forests, Artifcial Intelligence in Medicine 137 (2023) 102486.

[17] G. Pagliarini, G. Sciavicco, Interpretable land cover classification with modal decision trees, European Journal of Remote Sensing 56 (2023) 2262738.

[18] G. Pagliarini, S. Scaboro, G. Serra, G. Sciavicco, I. E. Stan, Neural-Symbolic Temporal Decision Trees for Multivariate Time Series Classification, in: Proceedings of the 29th International Symposium on Temporal Representation and Reasoning (TIME), volume 247 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 13:1–13:15.

[19] O. L. Mangasarian, W. H. Wolberg, Cancer diagnosis via linear programming, Technical Report, University of Wisconsin-Madison Department of Computer Sciences, 1990.

[20] V. Sigillito, S. Wing, L. Hutton, , K. Baker, Ionosphere, UCI Machine Learning Repository, 1989.

[21] A. Sharma, Mobile price classification dataset, https://www.kaggle.com/datasets, 2021.

[22] K. Nakai, Yeast, UCI Machine Learning Repository, 1996.

[23] W. Nash, T. Sellers, S. Talbot, A. Cawthorn, W. Ford, Abalone, UCI Machine Learning Repository, 1995.

[24] A. Asuncion, D. Newman, UCI machine learning repository, 2007.