WoodTapper: a Python package for explaining decision tree ensembles

Abdoulaye Sakho[A,B], Jad Aouad[A], Carl-Erik Gauthier[C], Emmanuel Malherbe[A] and Erwan Scornet[B]
[A] Artefact Research Center, Paris, France
[B]Laboratoire de Probabilités, Statistique et Modélisation Sorbonne Université and Université Paris Cité, CNRS, F-75005, Paris
[C]Société Générale, Paris, France

1. Summary

WoodTapper provides a dedicated Python toolbox to inspect, decompose, and explain predictions from tree-based classifiers using methods that directly leverage the discrete structure of trees. More specifically, one module converts any tree-based model of scikit-learn into a rule-based method, and a second module provides example-based explanations for a given input sample.

2. Statement of need

Interpretable machine learning has become an increasingly critical concern [14, 18] as predictive models are deployed in high-stakes settings such as healthcare [11], marketing [13] or finance [9, 17] which is moreover a regulated sector. While complex models, such as tree-based ensemble methods, often yield strong predictive performance, their opacity can pose challenges for accountability, trust and compliance. Popular explanation toolkits such as SHAP [12] and LIME [16] offer model-agnostic or surrogate-based attributions that are broadly applicable, but they can be computationally expensive for large ensembles, and their computation remains a black-box estimation. Another field of interpretable models are rule-based methods. They are especially attractive because they are in the form of "if-then" statements, which are often easier to audit and communicate than latent feature transformations.

3. State of the Field

The original SIRUS algorithm [3, 2] offered a principled approach to generate simple and stable rule-based models from random forests. However, its implementations have been limited to R and Julia [3, 10], creating accessibility barriers for the Python data science community. WoodTapper addresses this gap by offering a native Python implementation that integrates with the scikit-learn ecosystem. Furthermore, WoodTapper extends rules extraction (i) from all the tree-based models in scikit-learn (Random Forest [4], Gradient Boosting [6] and Extremely Randomized Trees [7]) and (ii) to the multiclass classification setting.

In addition, WoodTapper introduces an example-based explainability methodology that can be applied to all scikit-learn tree-based models. This approach associates predicted samples with representative samples from the training data set, explaining tree-based models' predictions through examples.

We compare below our Python implementation WoodTapper with the Julia, R and skgrf versions (see Tables 1 and 3) and observe that WoodTapper provides broader options for tree-based model

extraction, faster rule-extraction runtimes, and support for multiclass classification with unlimited tree depth.

4. Research impact statement

As a Python package, WoodTapper provides a practical interface for extracting decision rules with SIRUS and integrating those rules into downstream projects. The SIRUS rule-extraction procedure, originally developed for random forests, has been applied in more than 200 publications (according to Google Scholar 2025) and has since been extended to diverse domains, including microbiome analysis, time-series analysis, and hydrological process analysis. Python adaptation of SIRUS has been requested multiple times by practitioners and researchers to the authors, which WoodTapper addresses. Beyond rule extraction, WoodTapper offers an example-based auditing tool for black-box, tree-based models deployed in production, and has been already applied successfully in the context of Artefact's consulting missions with clients of different sectors, including banking.[1]

WoodTapper has demonstrated notable research impact and has grown its user and contributor communities since its initial release. It has been downloaded more than 1,500 times,[2] indicating strong demand for a Python implementation. The package has evolved through contributions from multiple developers, with community members able to add new features, report and fix bugs, and propose enhancements. Furthermore, the fully reproducible benchmarks described below show concrete improvements in both generalisation to all tree-based models and computation time.

5. Software design

WoodTapper package adheres to the scikit-learn [15] estimator interface. This design enables smooth integration with existing workflows involving pipelines, cross-validation, and model selection, and enables to efficiently benefit from future maintenance updates and improvements to scikit-learn. The implementation leverages NumPy [8] for numerical computation and joblib for parallel processing to optimize performance on large datasets (1). The code architecture uses a Mixin inherited by all tree-based models to improve code reuse and factorization. For each tree-based ensemble type, a subclass inherits both the original scikit-learn class and the Mixin. The standard `fit` and `predict` methods remain unchanged, while additional methods of WoodTapper are available.

Table 1 | Comparison of SIRUS implementations across softwares.

Feature	**WoodTapper (Python)**	**SIRUS (R)**	**SIRUS (Julia)**
Language	Python 3.x	R 4.x	Julia 1.x
Forest	`scikit-learn`	`ranger`	Own
Availability	PyPI (`woodtapper`)	CRAN (`sirus`)	General registry
Parallelism	✓ (via `joblib`)	Limited (via `parallel`)	✓ (native)
ML pipelines	✓	Partial	Partial
Tree models	All	random forest	random forest
Rules interface	Unified class methods	Function-based	Function-based
Tree depth ≥ 3	✓	✓	×
Classification	Multiclass	Binary	Multiclass

[1] The details of these deployments remain confidential and are beyond the scope of this paper.
[2] Counted on pepy in first 2 months.

6. Rules Extraction Module

6.1. Formulation

In this section, we present our `RulesExtraction` module and we specifically consider its application to a random forest classifier, which corresponds to the SIRUS algorithm introduced by [3]. The module is designed to extract a set of rules from an ensemble of trees and then aggregate them to produce the final output probabilities.

We suppose that we have a training set $\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^n$ composed of n pairs taking values in $\mathbb{R}^p$ and $\{0, 1\}$ respectively (binary classification). We denote by $x_i^{(j)}$ the j-th component of the i-th sample in $\mathcal{D}_n$. We suppose we have a set of trees $\{\mathcal{T}_m, m = 1, \ldots, M\}$ from a tree ensemble procedure, each grown with randomness Θ_m.

In a tree $\mathcal{T}_m$, we denote by $\mathcal{P}$ a path of successive splits from the root node. $\mathcal{P}$ defines thus a hyperrectangle in the input space, $\hat{H}(\mathcal{P}) \subset \mathbb{R}^p$. We associate $\mathcal{P}$ with a rule function $\hat{g}_{\mathcal{P}}$ returning the mean of Y from the training sample inside and outside $\hat{H}(\mathcal{P})$.

For a set of trees $\{\mathcal{T}_m, m = 1, \ldots, M\}$ and a path $\mathcal{P}$, we define:

$$\hat{p}\left(\mathcal{P}\right) = \frac{1}{M} \sum_{m=1}^{M} \mathbb{1}_{\{\mathcal{P} \in \mathcal{T}(\Theta_m, \mathcal{D}_n)\}},$$

which corresponds to the empirical probability that the path $\mathcal{P}$ belongs to the set of trees $\{\mathcal{T}_m, m = 1, \ldots, M\}$. The set of final rules is $\{\hat{g}_{\mathcal{P}}, \mathcal{P} \in \hat{\mathcal{P}}_{p_0}\}$ where $\hat{\mathcal{P}}_{p_0} = \{\mathcal{P},\ \hat{p}(\mathcal{P}) > p_0\}$ with $p_0 \in [0, 1)$. The final rules are aggregated as follows for building the final estimator:

$$\hat{\eta}_{p_0}(x) = \frac{1}{|\hat{\mathcal{P}}_{p_0}|} \sum_{\mathcal{P} \in \hat{\mathcal{P}}_{p_0}} \hat{g}_{\mathcal{P}}(x).$$

Beyond the binary classification detailed here, we also implemented the rule extractor for regression, where final rules are aggregated using weights learned via ridge regression.

6.2. Running time

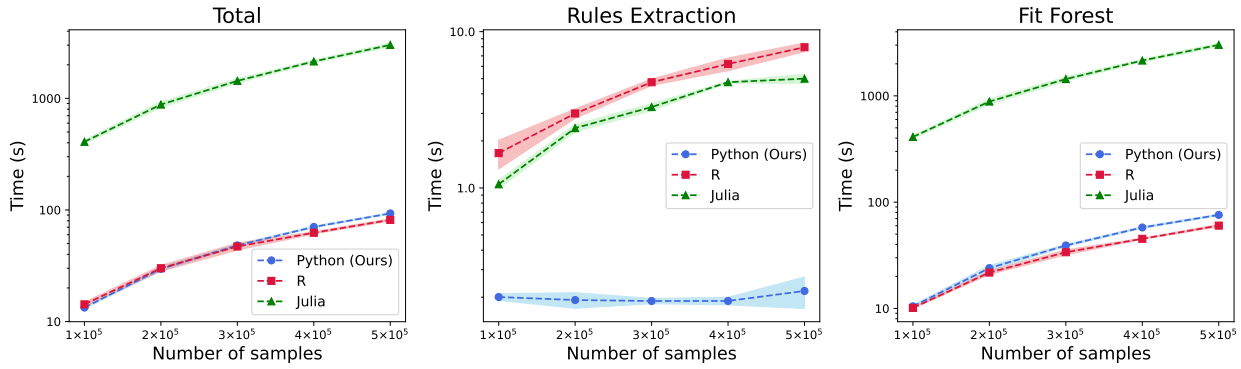


Figure 1 | SIRUS running time for simulated data using 5 threads, with d=200 and M=1000.

We compare the runtimes of SIRUS in Python (ours), R, and Julia using 5 threads on an AMD Ryzen Threadripper PRO 5955WX (16 cores, 4GHz) with 256GB RAM. We also experimented on

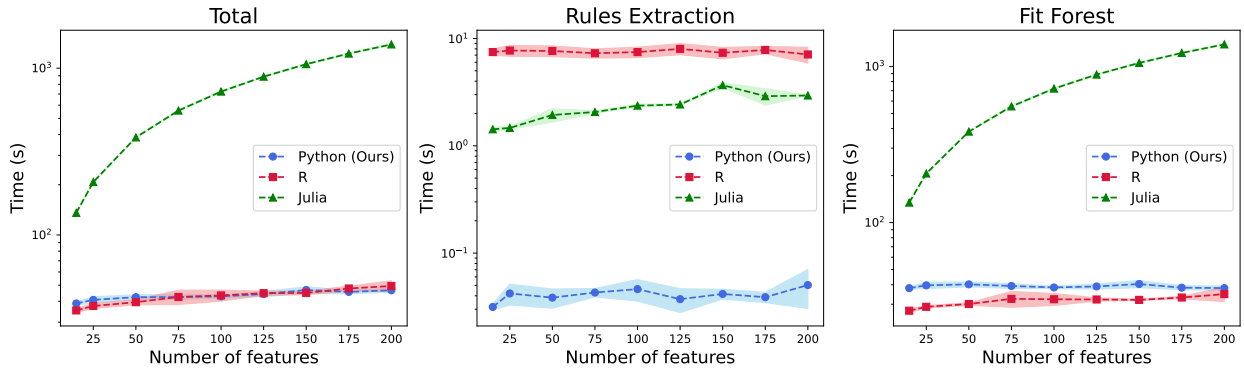


Figure 2 | SIRUS running time for simulated data using 5 threads, with n=300K and M=1000.

large-scale industrial data sets, including from the banking sector, and observed the same trends as displayed here. SIRUS.jl exhibits higher runtime compared to Python and R implementations. The R version, relying on ranger, is faster for tree construction on large datasets than scikit-learn. Our Python implementation, however, is considerably more efficient for rule extraction, regardless of sample size or feature dimensionality (see Figures 1 and 2).

6.3. Extracted rules and predictive performances

The rules generated by the original SIRUS (R) and our RulesExtraction implementation (Figures 3 and 4) on the Titanic dataset match for 9 out of 10 rules. The predictive performances reported in Table 2 are also very similar, confirming that our implementation faithfully reproduces the original algorithm. The remaining differences stem from the fact that the Python and R implementations do not use the exact same randomness when growing trees. As a result, rule occurrence frequencies can vary, which may change the ordering of the rules (Figures 3 and 4) and, in turn, affect the final rule selected (therefore, the predictive performance is not exactly identical).

```
 [1] "Proportion of class 1 = 0.386 - Sample size n = 887"
 [2] "if Sex in {male} then 0.19 (n=573) else 0.742 (n=314)"
 [3] "if Pclass < 3 & Sex in {female} then 0.947 (n=170) else 0.252 (n=717)"
 [4] "if Sex in {male} & Age < 14.8 then 0.535 (n=43) else 0.378 (n=844)"
 [5] "if Pclass < 2 & Sex in {male} then 0.369 (n=122) else 0.388 (n=765)"
 [6] "if Sex in {male} & Fare < 14.5 then 0.117 (n=332) else 0.546 (n=555)"
 [7] "if Sex in {male} & Fare < 22.2 then 0.132 (n=380) else 0.576 (n=507)"
 [8] "if Sex in {male} & Fare < 10.5 then 0.107 (n=270) else 0.507 (n=617)"
 [9] "if Sex in {male} & Fare < 27.7 then 0.151 (n=430) else 0.606 (n=457)"
[10] "if Sex in {male} & Fare < 39.7 then 0.169 (n=485) else 0.647 (n=402)"
[11] "if Sex in {male} & Fare < 78 then 0.179 (n=542) else 0.71 (n=345)"
[12] "if Sex in {female} & Siblings.Spouses.Aboard < 3 then 0.775 (n=293) el
se 0.194 (n=594)"
```

Figure 3 | SIRUS (R) rules on Titanic data set.

7. Example-based explainability module

7.1. Formulation

The `ExampleExplanation` module of `WoodTapper` is independent of the `RulesExtraction` module and provides an example-based explainability. It enables tree-based models to identify the most similar training samples to x, using the similarity measure induced by generalized random forests

```
   Condition                                THEN P(C1)      ELSE P(C1)
-------------------------------------------------------------------------
if   Sex is male                          then 19%                else 74%
if   Sex is female & Pclass <= 2.00       then 95%                else 25%
if   Sex is male & Age <= 14.80           then 53%                else 38%
if   Sex is male & Pclass <= 1.00         then 37%                else 39%
if   Sex is male & Fare <= 14.45          then 12%                else 55%
if   Sex is male & Fare <= 22.22          then 13%                else 58%
if   Sex is male & Fare <= 10.50          then 11%                else 52%
if   Sex is male & Fare <= 27.72          then 15%                else 61%
if   Sex is male & Fare <= 77.96          then 18%                else 71%
if   Sex is male & Fare <= 39.69          then 17%                else 65%
if   Sex is male & Fare <= 8.05           then 10%                else 48%
if   Sex is male & Pclass <= 2.00         then 27%                else 43%
```

Figure 4 | WoodTapper SIRUS (Ours) rules on Titanic data set.

Table 2 | Performance metrics for Titanic and House Sales datasets.

Dataset	Metric	SIRUS (original R)	Ours
Titanic	Accuracy	0.79 ± 0.03	0.78 ± 0.02
	(ROC) AUC	0.84 ± 0.04	0.84 ± 0.04
House Sales	MSE	0.35 ± 0.02	0.34 ± 0.01
	MAE	0.26 ± 0.01	0.26 ± 0.01

[4, 1]. For a new sample x with unknown label and a decision tree $\mathcal{T}_m$, let $\mathcal{L}_m(x)$ denote the set of training samples that share the same leaf as x. We define the similarity $w(x, x_i)$ between x and x_i as:

$$w(x, x_i) = \frac{1}{M} \sum_{m=1}^{M} \frac{\mathbb{1}_{\{x_i \in \mathcal{L}_m(x)\}}}{|\mathcal{L}_m(x)|}.$$

Finally, the l training samples with the highest $w(x, x_i)$ values, along with their target values y_i, are shown as the examples that best explain the prediction of x by the tree-based ensemble model.

In Python, the *skgrf* [5] package is an interface for using the R implementation of generalized random forest, focusing on classifiers for specific learning tasks (causal inference, quantile regression,...). For each task, the user can compute the kernel weights, equivalent to our leaf frequency match introduced above. Thus, we compare the kernel weights computation by *skgrf* and our module. We stress the fact that our `ExampleExplanation` is designed for usual tree-based models such as random forests or extra trees and not specifically in a context of causal inference or quantile regression. In particular, the tree building of our forest is different from the one in *skgrf*.

7.2. Running time

In figure 5, we compare the kernel weight computation runtime of `ExampleExplanation` and *skgrf* [5] using the same hardware as previous experiments. `ExampleExplanation` is consistently faster.

8. AI usage disclosure

Generative AI tools were used in this software only to implement the Cython function in the `ExampleExplanation` module and to draft certain docstring elements. For writing this manuscript and preparing supporting materials, generative AI was employed solely for formatting.

Table 3 | Comparison of GRF weight computations in several Python packages.

Feature	**WoodTapper (Py)**	**skgrf (Py)**
Forest implementation	`scikit-learn`	`ranger`
Language	Python	Python & R
Package availability	PyPI (`woodtapper`)	PyPI (`skgrf`)
scikit-learn API compatible	✓	✓
Tree-based models	All	Tree and random forest
GRF	×	✓

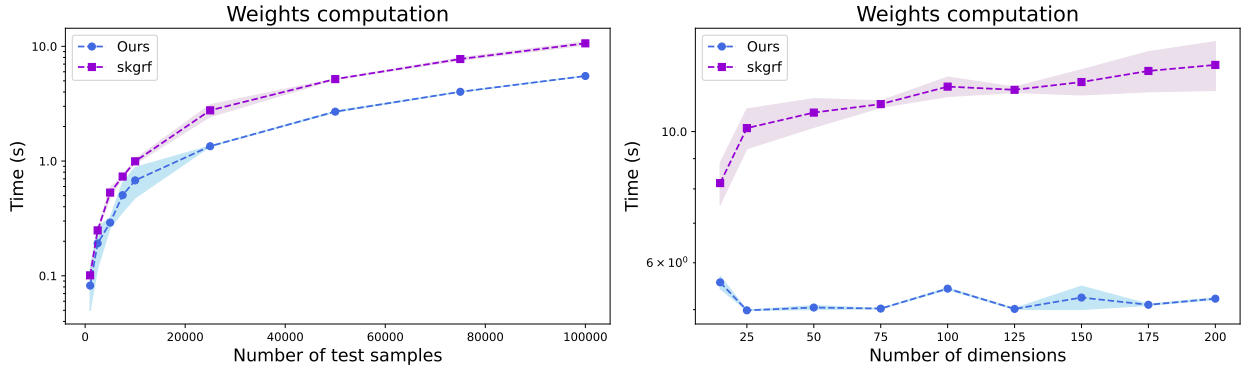


Figure 5 | Weights computation running time for simulated data using.

9. Acknowledgements

We would like to express our gratitude to Clément Bénard, Alexandre Chaussard and Vincent Auriau for their valuable help and feedback.

References

[1] Susan Athey, Julie Tibshirani, and Stefan Wager. Generalized random forests. 2019.

[2] Clément Bénard, Gérard Biau, Sébastien Da Veiga, and Erwan Scornet. Interpretable random forests via rule extraction. In *International conference on artificial intelligence and statistics*, pages 937–945. PMLR, 2021.

[3] Clément Bénard, Gérard Biau, Sébastien Da Veiga, and Erwan Scornet. Sirus: Stable and interpretable rule set for classification. 2021.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Christopher Flynn. skgrf - grf in Python. 2021.

[6] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[7] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[8] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[9] Amira Kamil Ibrahim Hassan and Ajith Abraham. Modeling insurance fraud detection using imbalanced data classification. In *Advances in Nature and Biologically Inspired Computing: Proceedings of the 7th World Congress on Nature and Biologically Inspired Computing (NaBIC2015) in Pietermaritzburg, South Africa, held December 01-03, 2015*, pages 117–127. Springer, 2016.

[10] Rik Huijzer, Frank Blaauw, and Ruud den Hartigh. Sirus. jl: Interpretable machine learning via rule extraction. *The Journal of Open Source Software*, 8(90):5786, 2023.

[11] Mohammed Khalilia, Sounak Chakraborty, and Mihail Popescu. Predicting disease risks from highly imbalanced data using random forest. *BMC medical informatics and decision making*, 11:1–13, 2011.

[12] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[13] Nam N Nguyen and Anh T Duong. Comparison of two main approaches for handling imbalanced data in churn prediction problem. *Journal of advances in information technology*, 12(1), 2021.

[14] Anne-Marie Nussberger, Lan Luo, L Elisa Celis, and Molly J Crockett. Public attitudes value interpretability but prioritize accuracy in artificial intelligence. *Nature communications*, 13(1):5821, 2022.

[15] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "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*, pages 1135–1144, 2016.

[17] Abdoulaye Sakho, Emmanuel Malherbe, Carl-Erik Gauthier, and Erwan Scornet. Harnessing mixed features for imbalance data oversampling: Application to bank customers scoring. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 247–264. Springer, 2025.

[18] Kacper Sokol and Peter Flach. Interpretable representations in explainable ai: from theory to practice. *Data Mining and Knowledge Discovery*, 38(5):3102–3140, 2024.