The Journal of Open Source Software

WoodTapper: a Python package for explaining decision tree ensembles

Abdoulaye SAKHO[1,2], **Jad AOUAD**[1], **Carl-Erik GAUTHIER**[3], **Emmanuel MALHERBE**[1], **and Erwan SCORNET**[2]

1 Artefact Research Center, Paris, France **2** Laboratoire de Probabilités, Statistique et Modélisation Sorbonne Université and Université Paris Cité, CNRS, F-75005, Paris **3** Société Générale, Paris, France

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Fangzhou Xie

Reviewers:

- @jncraton
- @jungtaekkim

Submitted: 15 November 2025
Published: unpublished

Statement of need

Interpretable machine learning has become an increasingly critical concern (Nussberger et al., 2022; Sokol & Flach, 2024) as predictive models are deployed in high-stakes settings such as healthcare (Khalilia et al., 2011), marketing (Nguyen & Duong, 2021) or finance (Hassan & Abraham, 2016; Sakho et al., 2025) 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 and LIME 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.

`WoodTapper` complements these ecosystems by providing a dedicated Python toolbox to inspect, decompose, and explain predictions 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 explains example-based explanations given an input sample. We describe these two modules below.

State of the Field

The original SIRUS algorithm (Bénard et al., 2021a, 2021b) 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 (Bénard et al., 2021b; Huijzer et al., 2023), 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, Gradient Boosting and Extremely Randomized Trees) 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 Table 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

The Journal of Open Source Software

unlimited tree depth.

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) 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 researcher 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, reporting and fixing bugs, and proposing enhancements. Furthermore, the fully reproducible benchmarks described below show concrete improvements in both generalisation to all tree-based models and computation time.

Software design

`WoodTapper` package adheres to the scikit-learn (Pedregosa et al., 2011) 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 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.

Rules Extraction Module

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 Bénard et al. (2021b).

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 $\hat{H}(\mathcal{P})$.

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

For a set of trees $\{\mathcal{T}_m, m = 1, \dots, M\}$ 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 $\{\mathcal{T}_m, m = 1, \dots, M\}$. 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.

Running time

Table 1: Comparison of SIRUS implementations across softwares.

Feature	`WoodTapper` (Py)	SIRUS (R)	SIRUS (Jl)
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	✓	✓	$\times$
Classification	Multiclass	Binary	Multiclass

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 250GB RAM. We also experimented on 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).

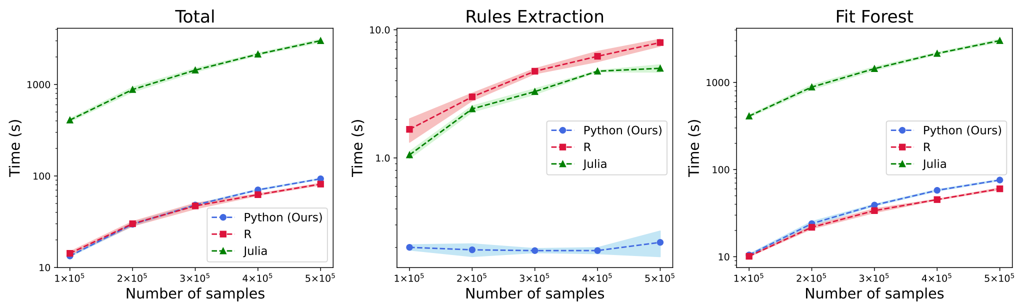

Figure 1: SIRUS running time for simulated data using 5 threads, with d=200 and M=1000.

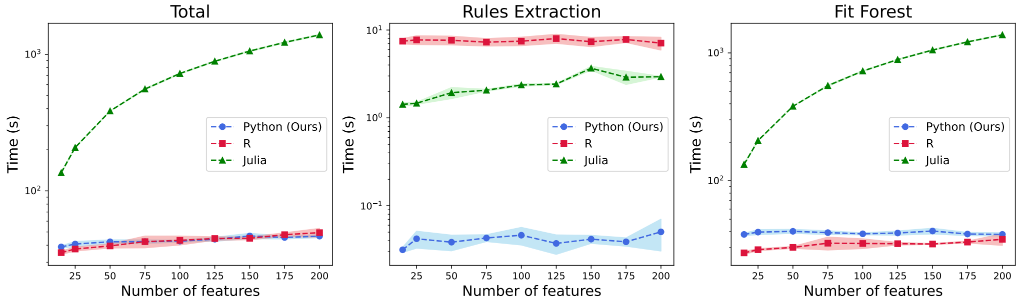

Figure 2: SIRUS running time for simulated data using 5 threads, with n=300K and M=1000.

Extracted rules and predictive performances

The rules produced by the original SIRUS (R) and our `RulesExtraction` in 3 and 4 on the Titanic data set are identical, and predictive performances in Table 2 are very similar, confirming that our implementation faithfully reproduces the original algorithm.

```
 [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.

```
   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

The Journal of Open Source Software

Example-based explainability module

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 (Athey et al., 2019; Breiman, 2001). 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* (Flynn, 2021) package is an interface for using the R implementation of generalized random forest, focusing on classifiers for specifics learning tasks (causal inference, quantile regression,…). For each task, the user can compute the kernel weights, equivalently to our leaf frequency match introduce above. Thus, we compare the kernel weights computation by *skgrf* and our module. We stress on the fact that our `ExampleExplanation` is designed for usual tree-based models such as random forest of 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*.

Running time

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	×	✓

In figure 5, we compare the kernel weight computation runtime of `ExampleExplanation` and *skgrf* (Flynn, 2021) using the same hardware as previous experiments. `ExampleExplanation` is consistently faster.

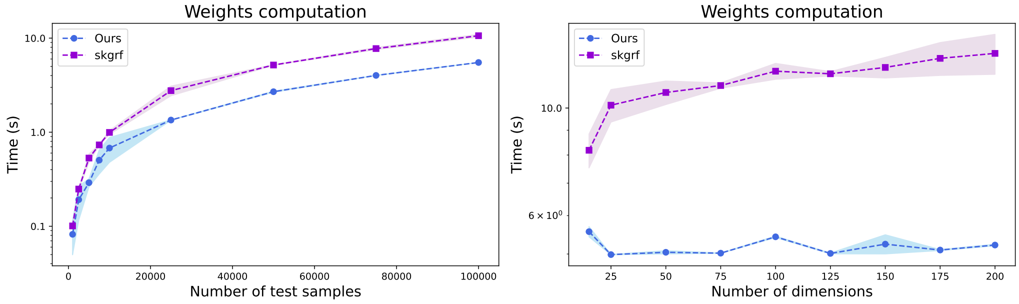

Figure 5: Weights computation running time for simulated data using.

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.

Acknowledgements

We would like to express our gratitude to Clément BENARD, Alexandre CHAUSSARD and Vincent AURIAU for their valuable help and feedbacks.

References

Athey, S., Tibshirani, J., & Wager, S. (2019). *Generalized random forests*. https://doi.org/10.1214/18-aos1709

Bénard, C., Biau, G., Da Veiga, S., & Scornet, E. (2021a). Interpretable random forests via rule extraction. *International Conference on Artificial Intelligence and Statistics*, 937–945.

Bénard, C., Biau, G., Da Veiga, S., & Scornet, E. (2021b). *Sirus: Stable and interpretable rule set for classification*. https://doi.org/10.1214/20-ejs1792

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1007/978-3-030-56485-8_3

Flynn, C. (2021). *Skgrf - grf in python*. https://github.com/crflynn/skgrf/tree/main?tab=readme-ov-file

Hassan, A. K. I., & Abraham, A. (2016). Modeling insurance fraud detection using imbalanced data classification. *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*, 117–127. https://doi.org/10.1007/978-3-319-27400-3_11

Huijzer, R., Blaauw, F., & Hartigh, R. den. (2023). Sirus. Jl: Interpretable machine learning via rule extraction. *The Journal of Open Source Software*, *8*(90), 5786. https://doi.org/10.21105/joss.05786

Khalilia, M., Chakraborty, S., & Popescu, M. (2011). Predicting disease risks from highly imbalanced data using random forest. *BMC Medical Informatics and Decision Making*, *11*, 1–13. https://doi.org/10.1186/1472-6947-11-51

Nguyen, N. N., & Duong, A. T. (2021). Comparison of two main approaches for handling imbalanced data in churn prediction problem. *Journal of Advances in Information Technology*, *12*(1). https://doi.org/10.12720/jait.12.1.29-35

Nussberger, A.-M., Luo, L., Celis, L. E., & Crockett, M. J. (2022). Public attitudes value interpretability but prioritize accuracy in artificial intelligence. *Nature Communications*, *13*(1), 5821. https://doi.org/10.1038/s41467-022-33417-3

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., & others. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, *12*, 2825–2830.

Sakho, A., Malherbe, E., Gauthier, C.-E., & Scornet, E. (2025). Harnessing mixed features for imbalance data oversampling: Application to bank customers scoring. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 247–264. https://doi.org/10.1007/978-3-032-06118-8_15

Sokol, K., & Flach, P. (2024). Interpretable representations in explainable AI: From theory to practice. *Data Mining and Knowledge Discovery*, *38*(5), 3102–3140. https://doi.org/10.1007/s10618-024-01010-5