The Journal of Open Source Software

Balsa: A Fast C++ Random Forest Classifier with Command-line and Python Interface

Tobias Borsdorff [1¶], **Denis de Leeuw Duarte** [2], **Joris van Zwieten** [2], **Soumyajit Mandal** [1], **and Jochen Landgraf** [1]

1 SRON Space Research Organization Netherlands **2** Jigsaw B.V., The Netherlands ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

DRAFT

Summary

Random Forest classifiers are widely used machine learning methods that combine multiple decision trees to improve predictive accuracy and reduce overfitting (Breiman, 2001). While implementations like scikit-learn (Pedregosa et al., 2011) are popular in the Python ecosystem, operational processing environments often require high-performance C++ implementations that can handle large datasets efficiently while maintaining low memory footprints.

Balsa is a high-performance, open-source (BSD 3-Clause License) C++ implementation of the Random Forest classifier, designed with runtime efficiency and memory optimization as core design priorities. The implementation follows the modern C++17 standard and a complete API documentation is provided with the package. Originally developed for the cloud-clearing classification in the operational processing of Copernicus Sentinel-5 Precursor (S5P) methane data (Lorente et al., 2021, 2023), Balsa addresses the strict performance requirements for satellite data processing (Borsdorff, Martinez-Velarte, et al., 2024). The library has been successfully integrated into ESA's operational data processing framework (Borsdorff, Mandal, et al., 2024a, 2024b), where it currently runs in both the offline and near real-time S5P methane products, processing large volumes of satellite observations with stringent latency requirements.

Statement of Need

Balsa was developed by SRON Netherlands Institute for Space Research in cooperation with Jigsaw B.V. to meet the demanding performance requirements of operational satellite data processing. During initial development phases, the scikit-learn implementation (Pedregosa et al., 2011) was used, but operational integration required a C++ implementation with significantly improved runtime and memory efficiency. The transition to near real-time processing for S5P methane data further emphasized the need for a solution that could handle millions of data points with minimal latency and memory overhead. While Balsa was developed for S5P methane processing, it is designed as a general-purpose Random Forest classifier applicable to diverse machine learning tasks beyond satellite data processing.

Balsa offers several key advantages over existing implementations:

- **Performance**: Balsa demonstrates superior runtime performance during the training and prediction phase compared to both scikit-learn and the C++-based Ranger implementation (Wright & Ziegler, 2017) (Figure 1). This advantage is particularly critical for operational applications where classification speed directly impacts processing throughput.
- **Memory efficiency**: Balsa consistently shows lower memory footprint during both training and prediction phases, making it particularly suitable for processing large datasets

(Figure 2). Benchmarks demonstrate scalability to datasets with millions of data points.

- **Accuracy**: All three implementations (Balsa, scikit-learn, and Ranger) produce essentially identical prediction accuracy (Figure 3), ensuring performance improvements stem from optimization rather than algorithmic compromises.
- **Flexible integration**: Balsa's compact binary format enables seamless workflows where models trained in Python can be efficiently loaded and used in operational C++ environments.
- **Distributed training**: Multiple machines can train Random Forest models independently on the same or different datasets, with trained models easily merged to create stronger classifiers without requiring centralized coordination.

The performance comparisons presented in Figure 1, Figure 2, and Figure 3 were conducted using the TROPOMI cloud-clearing classification problem as a real-world benchmark, with datasets derived from TROPOMI satellite measurements as described in Borsdorff et al. (Borsdorff, Martinez-Velarte, et al., 2024).

The library provides three levels of user interaction: a comprehensive C++ API for direct integration into applications, command-line tools for standalone training and classification tasks, and Python bindings installable via pip that simplify development while maintaining access to the high-performance C++ core. Balsa is cross-platform, supporting Linux, macOS, and Windows environments. This multi-layered approach supports both rapid prototyping in Python and deployment in performance-critical production environments. Balsa supports both single- and double-precision arithmetic, allowing memory optimization as needed.

Key Features

Balsa provides a complete ecosystem for Random Forest classification:

Core Library: The C++ library supports both binary and multi-class classification with multithreaded training capabilities. Models can be trained in parallel across multiple cores and even across multiple independent machines, with the resulting forests merged to create stronger classifiers. The library uses an efficient binary format for model storage, enabling fast loading and minimal disk usage.

Command-Line Tools: The package includes utilities for the complete machine learning workflow: `balsa_generate` creates synthetic datasets for testing, `balsa_train` trains models with configurable parameters, `balsa_classify` performs batch classification, `balsa_measure` calculates comprehensive performance metrics (including accuracy, precision, recall, F-scores, P4 metric, diagnostic odds ratio, and confusion matrices), `balsa_featureimportance` analyzes feature contributions following a permutation based method, `balsa_merge` combines independently trained models for distributed training workflows, and `balsa_test` runs unit tests to verify installation and functionality.

Python Interface: Python bindings provide NumPy integration and a familiar interface for Python developers, while maintaining the performance benefits of the underlying C++ implementation. The package is easily installable via pip, making it readily accessible to the Python machine learning community. Models trained via Python can be directly used by the C++ tools and vice versa, facilitating hybrid workflows where development occurs in Python and deployment in high-performance C++ environments.

Performance Analysis Tool: The `rfcperf` benchmarking utility enables systematic comparison of Random Forest implementations across different dataset sizes, ranging from thousands to millions of samples. It measures system performance (CPU time, memory usage, wall-clock time) and classification quality (accuracy, precision, recall, F-scores) while generating comparative visualization reports. This tool was used to generate the performance comparisons presented in Figure 1, Figure 2, and Figure 3, and allows users to reproduce these benchmarks on their own systems and datasets.

Comprehensive Documentation: The package includes detailed documentation covering installation, theoretical background, optimization guidelines, and extensive examples for both command-line and programmatic usage.

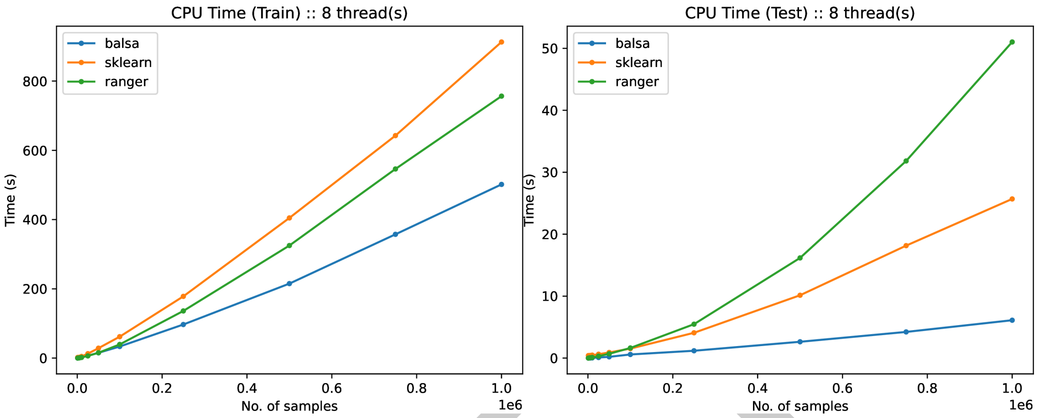

Figure 1: Runtime comparison during RFC training (left) and prediction (right) for scikit-learn (orange), Ranger (green), and Balsa (blue) as a function of dataset size, evaluated on TROPOMI cloud-clearing data. Balsa demonstrates superior prediction performance, which is critical for operational applications including near real-time processing.

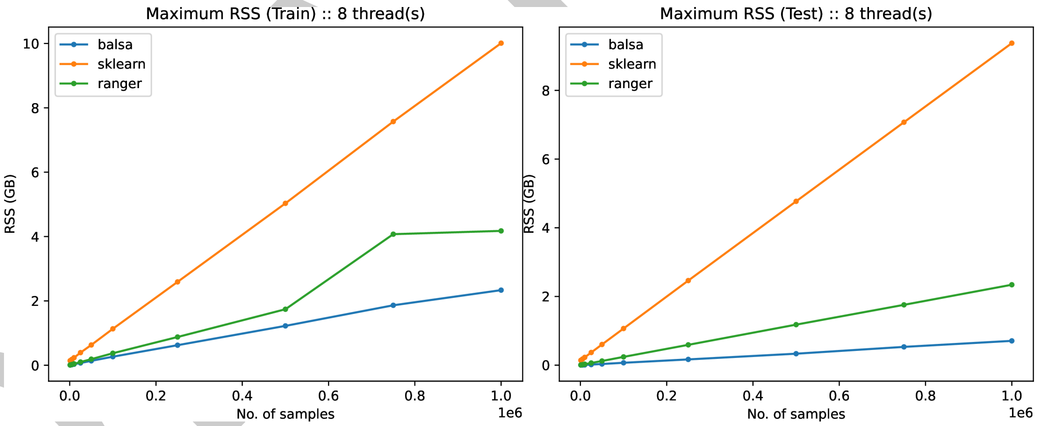

Figure 2: Memory usage during RFC training (left) and prediction (right) for scikit-learn (orange), Ranger (green), and Balsa (blue) as a function of dataset size, evaluated on TROPOMI cloud-clearing data. Balsa maintains consistently lower memory footprint across dataset sizes ranging from thousands to millions of samples, enabling processing of larger datasets in memory-constrained environments.

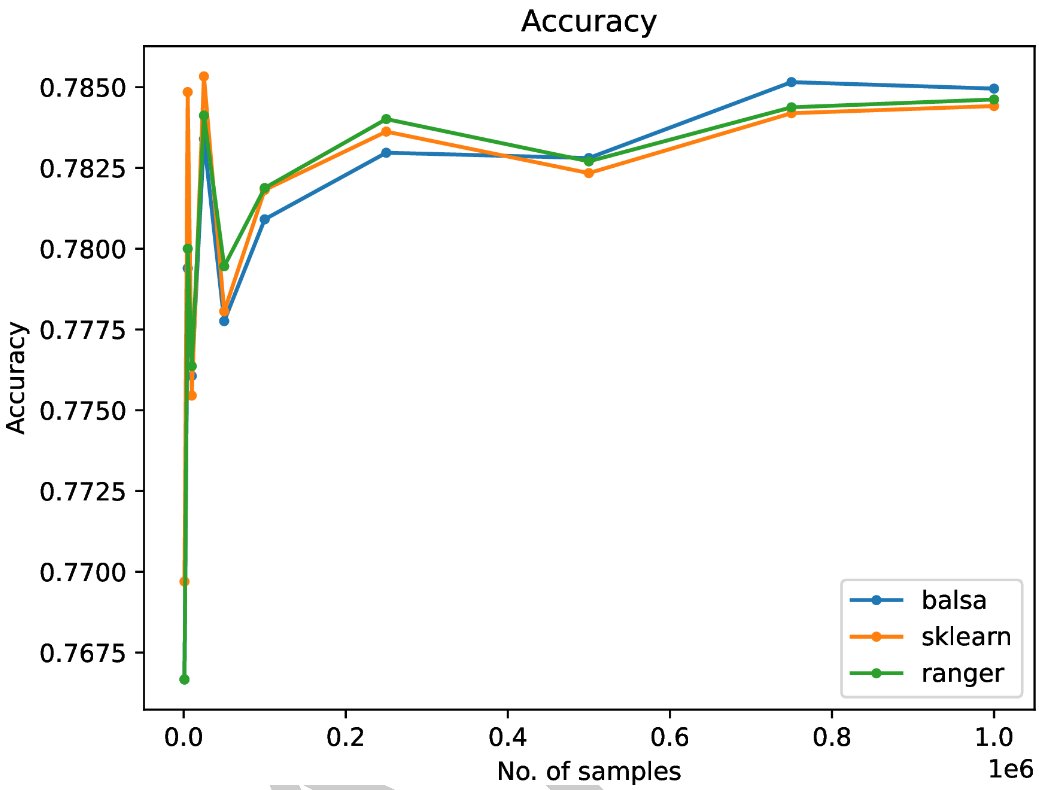

Figure 3: Classification accuracy for scikit-learn (orange), Ranger (green), and Balsa (blue) as a function of dataset size, evaluated on TROPOMI cloud-clearing data. All three implementations achieve comparable accuracy, confirming that Balsa's performance gains do not compromise prediction quality.

Availability

Balsa is publicly available under the BSD 3-Clause License at Balsa GitHub Repository

Authors Contribution

T. Borsdorff led the project and coordinated the overall development. He contributed to the conceptual design of the software, performed the verification and validation activities together with J. Landgraf and S. Mandal, and wrote the manuscript. J. van Zwieten and D. de Leeuw Duarte were responsible for the main implementation of the Balsa library, including the core C++ codebase and associated tools. All authors contributed to discussions, refinement of the software, and preparation of the manuscript, and all agree on the order of authorship.

Acknowledgements

Balsa development was funded by the European Space Agency.

References

Borsdorff, T., Mandal, S., Martinez Velarte, M., Barr, A., & Landgraf, J. (2024a). *Algorithm theoretical baseline document for sentinel-5 precursor: TROPOMI CH4 random forest cloud filter*. https://doi.org/10.5281/zenodo.14186320

Borsdorff, T., Mandal, S., Martinez Velarte, M., Barr, A., & Landgraf, J. (2024b). *Product user manual for the random forest classifier (RFC) c++ implementation to be used for TROPOMI CH4* (Version 1.0.0) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.14186406

Borsdorff, T., Martinez-Velarte, M. C., Sneep, M., Linden, M. ter, & Landgraf, J. (2024). Random forest classifier for cloud clearing of the operational TROPOMI XCH4 product. *Remote Sensing*, *16*(7). https://doi.org/10.3390/rs16071208

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Lorente, A., Borsdorff, T., Butz, A., Hasekamp, O., Brugh, J. aan de, Schneider, A., Wu, L., Hase, F., Kivi, R., Wunch, D., Pollard, D. F., Shiomi, K., Deutscher, N. M., Velazco, V. A., Roehl, C. M., Wennberg, P. O., Warneke, T., & Landgraf, J. (2021). Methane retrieved from TROPOMI: Improvement of the data product and validation of the first 2 years of measurements. *Atmospheric Measurement Techniques*, *14*(1), 665–684. https://doi.org/10.5194/amt-14-665-2021

Lorente, A., Borsdorff, T., Martinez-Velarte, M. C., & Landgraf, J. (2023). Accounting for surface reflectance spectral features in TROPOMI methane retrievals. *Atmospheric Measurement Techniques*, *16*(6), 1597–1608. https://doi.org/10.5194/amt-16-1597-2023

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., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Wright, M. N., & Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, *77*(1), 1–17. https://doi.org/10.18637/jss.v077.i01

DRAFT