Do Tree-based Models Need Data Preprocessing?

Hubert Ruczyński[1] **Anna Kozak**[1]

[1]Warsaw University of Technology

Abstract The number of machine learning (ML) algorithms, and ML-related methodologies, which increase the overall performance, grows every year. The abundance of possibilities makes it impossible for the data scientists to test all of them every time, thus the need for best practices evaluation studies exists. In the scope of this paper, we attempt to evaluate the impact of preprocessing strategies on tree-based models. To conduct this study we prepare 38 different preprocessing strategies and train almost one million tree-based models. Furthermore, we analyze the impact of different data preparation strategies and outline the best-performing ones with the usage of newly introduced preprocessibility measure.

1 Introduction

Data preprocessing is an essential part of the machine learning pipeline (García et al., 2015; Alasadi and Bhaya, 2017; Çetin and Yıldız, 2022), as it greatly influences data quality (Famili et al., 1997) and the discovery of relationships that can optimize machine learning models (Obaid et al., 2019). Despite being a time-consuming process (Anaconda, 2022) it's fundamental, especially for large datasets where dimensionality reduction can save time in subsequent processes (García et al., 2016). Data preprocessing encompasses not only quality checks but also key elements like transformation, filling in missing data, outlier detection, and variable selection for the model.

Although it's a common belief that tree-based models do not require preprocessing as they can handle it without any changes, experiments suggest that we may achieve even better results with proper preprocessing (Caruana et al., 2008; Grinsztajn et al., 2022). This understanding can be beneficial for automated machine learning (AutoML) pipelines, allowing us to optimize and implement an automated machine learning process that preprocesses the dataset appropriately for the selected model to yield better results.

This paper presents an extensive experiment involving 38 data preprocessing strategies for binary and multiclass classification and regression tasks. We use five tree-based models: decision tree, random forest, XGBoost, LightGBM, and CatBoost. We expanded the *forester*[1] software to include more interference in preprocessing for automated model learning. More information about the tool is available in Appendix A.

2 Datasets Description

For binary and multiclass classification tasks, we utilized datasets from the OpenML-CC18 benchmark (Bischl et al., 2019), whereas the regression tasks were sourced from OpenML (Gijsbers et al., 2019). Furthermore, to guarantee that the preprocessing module encounters significant challenges in some cases, we introduced modified versions of certain datasets where data quality was intentionally diminished. For each task type we chose two datasets, one highly dimensional, and the other with a small amount of columns to ensure various scenarios, and applied the modifications described in Appendix B, where you can find all details concerning the dataset selection. Eventually we ended up with 25 datasets, where 6 of them are the modified versions with lowered data quality.

[1]https://github.com/ModelOriented/forester

3 Preprocessing Strategies

The *forester's* custom preprocessing module focuses on three major data preprocessing areas, namely the heuristic removals, data imputation, and feature selection (more details in Appendix C). Each of those sub-modules consists of methods that use hyperparameters, such as k for the KNN algorithm. To make the study computationally feasible, we decided to use only the default parameters for each method. Furthermore, we aggregate the removal strategies to the following approaches:

- **Minimal** - it removes the observations that do not have the target value,
- **Medium** - it removes duplicate, id-like, static (threshold = 0.99), and sparse (threshold = 0.3) columns and corrupted rows with too many missing values (threshold = 0.3),
- **Maximal** - it is medium approach with the removal of highly correlated columns.

In order to minimize the computational overhead, we decided to omit some presumably costly combinations, which resulted in carrying on with 38 different preprocessing strategies, described in Appendix D. Eventually, after conducting the data preprocessing, we ended up with 950 different datasets, spanning over all the aforementioned possibilities.

4 Model Training

The next step is the model training, where we also incorporate the *forester* package. The `train()` function is used for all 38 preprocessed versions of 25 datasets. Each data frame was split into training, testing, and validation samples, with the proportions 60%, 20%, and 20%, respectively. To ensure reproducibility of the results, we set the split seed to 123. To limit the computational time, we only use random search as a tuning method, which is similar to conducting the experiments multiple times.
Eventually, for each preprocessed dataset (950 tasks) and every engine (5 models: decision tree, random forest, XGBoost, CatBoost, and LightGBM), we train 21 configurations, where 1 of them has default parameters, and the other 20 have their parameters chosen by the random search algorithm. We save the achieved outcomes, and measure the time of the training. Finally, we ended up with 999 750 models ($950 \times 5 \times 21$) for the whole study. More details are available in the Appendix E.

5 Results

To analyze the impact of data preprocessing strategies and not the particular trained models, for each dataset we considered only the best performing solution in terms of accuracy for classification, and R^2 for the regression. Furthermore, we do not focus on the raw performance scores, but rather its comparison to the baseline results.
The **baseline preprocessing strategy** is a preprocessing strategy which consists of minimal removal, median-other imputation, and lack of feature selection. The **baseline dataset** for the dataset D is the dataset created from D with the **baseline preprocessing strategy**. The **baseline model** for the dataset D is the best performing model trained on the **baseline dataset**.
To validate how often given strategies improve or worsen the results, we analyzed the number of wins, ties and losses of all strategies in comparison to the baseline. For the dataset D we compared the results of the best model of each preprocessing strategy for D ($\theta(m)$) to the performance of it's **baseline model** ($\theta(m_B)$). Afterwards, we counted the number of wins ($\theta(m) > \theta(m_b)$), ties ($\theta(m) = \theta(m_b)$) and losses ($\theta(m) < \theta(m_b)$).
Furthermore, we introduce the **preprocessibility** measure, based on tunability from (Probst et al., 2018). It describes how much performance can we gain or lose for a dataset D by using various preprocessing strategies. The Equation (1) describes **positive preprocessibility**, and Equation (2) the **negative preprocessibility**.

$$P^{+}(D) = \max_{d_i \in D}(\max_{m_j(d_i)} (\theta(m_j))) - \max_{m_j(B)} (\theta(m_j)), \tag{1}$$

$$P^{-}(D) = \min_{d_i \in D}(\min_{m_j(d_i)} (\theta(m_j))) - \min_{m_j(B)} (\theta(m_j)), \tag{2}$$

where D is a set preprocessed datasets, $d_i \in D$ is a dataset from D, θ is the performance measurement metric which values have probabilistic interpretation (in our case it's accuracy or R^2), $m_j(d_i)$ is the model trained on d_i dataset, and B is a baseline dataset for D.

Table 1: The impact of preprocessing methods on the tree-based models predictive quality.

Statistic	All strategies	Is FS Used?		Feature Selection Methods				Removal Strategy			Imputation Method			
		No	Yes	Boruta	MCFS	MI	VI	Minimal	Medium	Maximal	MICE	Median-other	Median-frequency	KNN
Wins [%]	15.5%	12.3%	17.3%	22.5%	8.0%	14.4%	22.7%	10.0%	13.3%	13.0%	29.2%	29.2%	27.5%	58.3%
Ties [%]	56.6%	70.6%	48.5%	53.0%	76.0%	40.4%	36.0%	85.0%	71.4%	55.0%	12.5%	41.6%	40.0%	25.0%
Losses [%]	27.9%	17.1%	34.2%	24.5%	16.0%	45.2%	41.3%	5.0%	15.3%	32.0%	58.3%	29.2%	32.5%	16.7%
Average positive preprocessibility	0.009	0.006	0.009	0.008	0.001	0.008	0.003	0.005	0.005	0.005	0.005	0.004	0.002	0.017
Average negative preprocessibility	-0.048	-0.013	-0.047	-0.013	-0.010	-0.044	-0.021	-0.002	-0.003	-0.011	-0.021	-0.019	-0.016	-0.011

The Table 1 presents us with the results depending on various aspects of data preprocessing strategies. The first column shows us the general impact of data preprocessing on tree-based models, and we can see that only for 15.5% of cases this step contributed positively to the outcomes. Although it is not much, delving deeper into the results helps us outline which data preparation techniques work best with considered model family.

At first, we focus on the feature selection (FS) methods being the most influential aspect of data preprocessing. We can notice that the incorporation of such algorithms drastically lowers the number of ties, and we observe more wins and losses. Unfortunately, the general results show us that although the positive preprocessibility grows when FS is used, the contribution to the negative preprocessibility is even larger. However, the analysis of particular methods shows us that Boruta algorithm works very well, whereas the methods like Mutual Information (MI) or Variable Importance (VI) are rather poor.

The analysis of removal strategies unravels that we should not remove highly correlated features before using tree-based models, as the maximal approach, which uses this method performs far worse than the simpler counterparts.

Finally, the outcomes indicate that KNN imputation works best among the selected methods with a remarkably high win percentage exceeding 50%. Quite surprisingly, MICE algorithm, which is another advanced imputation method, losses severely with much simpler median-other, and median-frequency approaches.

Additionally we decided to analyze which models are more influenced by preprocessing. The 'All preprocessing strategies' column of Table 2 shows us the differences between considered architectures. The decision tree, and LightGBM mostly yield a negative impact, whereas random forest, XGBoost and CatBoost benefit a lot. Random forest has the highest average positive preprocessibility, however, if we consider the baseline performance, we will see that it also is the worst performing one, thus it has more opportunities to improve.

To validate our findings that describe which preprocessing methods work best with tree-based models, we decided to check how well the optimal strategy works out. To do that we considered the preprocessing pipeline which includes medium removal strategy, KNN as an imputation method and Boruta or lack of feature selection. Furthermore, we limited the results to the models that benefit from preprocessing the most, being XGBoost, CatBoost and random forest. The outcomes

Table 2: The behaviour of different tree-based models on preprocessing pipelines and the validation of best strategy.

Statistic	All preprocessing strategies					Best preprocessing strategies			
	Decision tree	Random forest	XGBoost	LightGBM	CatBoost	XGBoost	CatBoost	Random forest	All
Wins [%]	13.7%	18.2%	17.0%	10.7%	22.0%	18.0%	36.0%	36.0%	30.0%
Ties [%]	60.6%	52.5%	55.5%	62.8%	50.5%	64.0%	52.0%	58.0%	58.0%
Losses [%]	25.7%	29.3%	27.5%	26.5%	27.5%	18.0%	12.0%	6.0%	12.0%
Average positive preprocessibility	0.007	0.014	0.010	0.004	0.010	0.006	0.007	0.008	0.007
Average negative preprocessibility	-0.063	-0.039	-0.043	-0.066	-0.070	-0.001	-0.001	-0.001	-0.001
Average maximal score (Accuracy/R^2)	0.752	0.694	0.845	0.795	0.866	0.840	0.862	0.688	0.797

are presented in the second column of Table 2. They prove that the optimal preprocessing strategy works well, as the average negative preprocessibility is a few times smaller than the positive one, and the models prepared that way beat the baseline more often than they lose.
To delve deeper into the analysis of results for the optimal strategy, we take a closer look at the Figure 1. Previous analyses showed us that the average positive preprocessibility achieved much higher values than the negative ones, however from the middle subplot we can see how exactly it was distributed. Eventually it comes out that in the case of 11 datasets the performance does not change at all. Moreover, the highest positive preprocessibility values reach up to 0.05, and are achieved mostly for the modified datasets. It indicates that preprocessing strategies are more useful when we deal with real-life data that is not perfect.

6 Limitations and Broader Impact Statement

The main limitation of our study is related to computational feasibility and regards the limitations to 38 out of 60 possible strategies. Covering the whole search space would be more beneficial and provide more insights. Furthermore, to minimize costs, we arbitrary chose the parameters of preprocessing algorithms, such as the desired number of columns for feature selection methods. Covering larger space of those parameters, for example by random searching them, would provide more reliable results. One of the future works following this study could be focusing only on the real-world data or adding more data preprocessing techniques related to feature engineering (e.g., Principal Component Analysis (PCA), scaling).
After careful reflection, the authors have determined that this work presents no notable negative impacts to society or the environment. On the contrary, it provides an insightful analysis of data preprocessing techniques for tree-based models outlines the best practices in this area, so the ML specialists do not have to conduct trial and error on their own.
Based on the conclusions, which shows that the majority of improvements occurred for the modified datasets, we advocate for creating the real-life data benchmark for ML and AutoML solutions. Those results show that using well-prepared datasets limits the impact of preprocessing methods, and does not reflect the real world, where it remains a crucial component.

7 Conclusion

In this work we analyzed the impact of preprocessing strategies on tree-based models, and pointed out how use them in an efficient way. We provide evidence that although in the majority of preprocessing scenarios, the obtained results were the same (56.6%) or worse (27.9%) than the baseline, for some examples (15.5%) we were able to increase model performance. Furthermore,

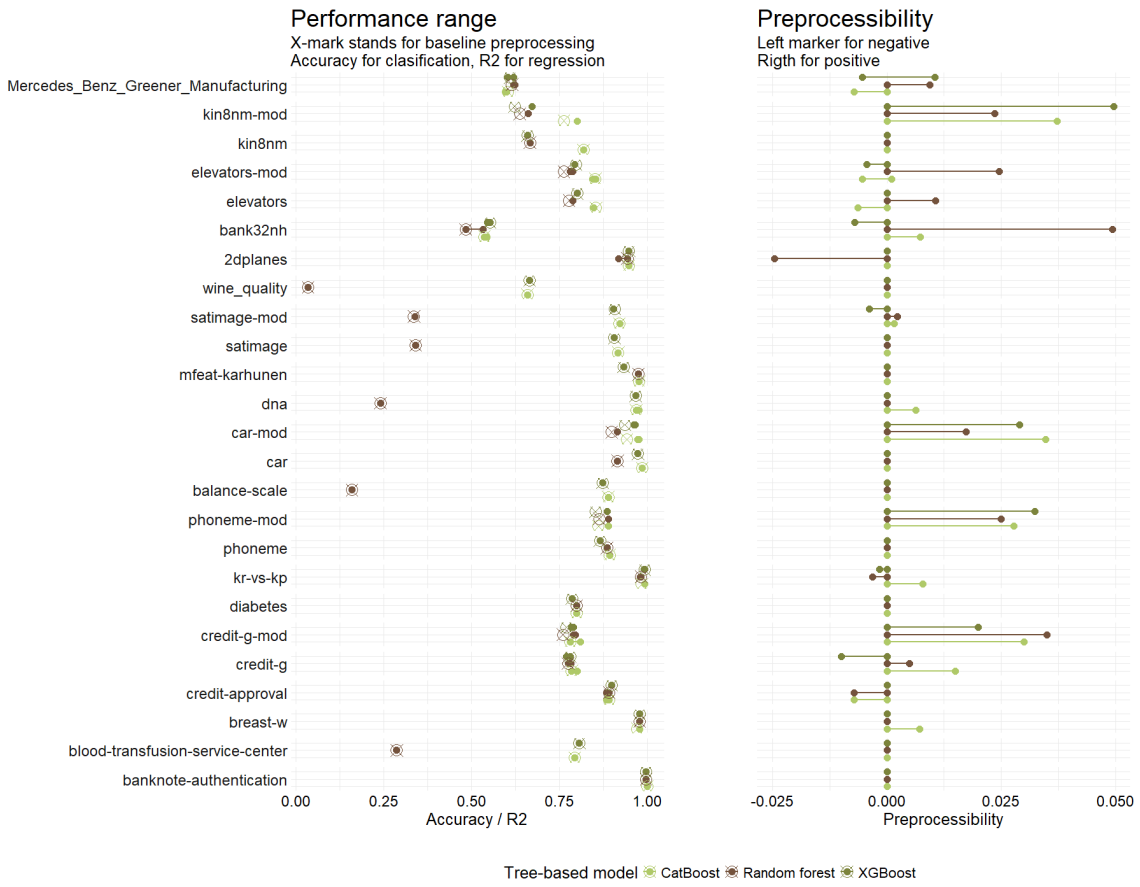

Figure 1: Detailed analysis of results for the optimal strategy. The left subplot presents the performance range, with the baseline marked as an X. The second one shows the positive and negative preprocessibility values.

we analyzed the impact of three major components of preprocessing heuristic removals, data imputation, and feature selection. The outcomes showed us that the most beneficial strategies for tree-based models are: the removal of duplicate, id-like, static, and sparse columns with the deduction of rows with too many missing values; imputation with KNN algorithm; and using Boruta or none feature selection methods. Additionally, we checked which tree-based models are the most influenced by the data preprocessing, which showed us that it has a negative impact on decision trees, and LightGBM, and a positive one on random forest, XGBoost, and CatBoost. Eventually, we validated the theoretical outcomes, by proper filtering of data, and the results further underlined our findings. Lastly, we observed that the data preprocessing was the most beneficial for the dataset with artificially worsened quality, which indicates its significance while dealing with real-life data.

Acknowledgements. Part of this study was financed from the competition CyberSummer@WUT-3 organized by Centrum Badawcze POB Cyberbezpieczeństwo i Analiza Danych, (CB POB Cyber&DS) at Warsaw University of Technology. Conducting this study on such a scale would not be possible without provided resources. The experiments would be infeasible without the support from PLGrid grants. We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2024/017150, and grant no. PLG/2024/017059.

References

Alasadi, S. A. and Bhaya, W. S. (2017). Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, 12(16):4102–4107.

Anaconda (2022). State of Data Science 2022. Anaconda's user study.

Batista, G. and Monard, M.-C. (2002). A study of k-nearest neighbour as an imputation method. volume 30, pages 251–260.

Bischl, B., Casalicchio, G., Feurer, M., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. (2019). OpenML Benchmarking Suites. *arXiv:1708.03731v2 [stat.ML]*.

Buuren, S. and Groothuis-Oudshoorn, C. (2011). MICE: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software*, 45.

Caruana, R., Karampatziakis, N., and Yessenalina, A. (2008). An empirical evaluation of supervised learning in high dimensions. *Proceedings of the 25th International Conference on Machine Learning*, pages 96–103.

Dramiński, M., Rada-Iglesias, A., Enroth, S., Wadelius, C., Koronacki, J., and Komorowski, J. (2008). Monte Carlo feature selection for supervised classification. *Bioinformatics (Oxford, England)*, 24:110–7.

Estévez, P. A., Tesmer, M., Perez, C. A., and Zurada, J. M. (2009). Normalized mutual information feature selection. *IEEE Transactions on neural networks*, 20(2):189–201.

Famili, A., Shen, W.-M., Weber, R., and Simoudis, E. (1997). Data preprocessing and intelligent data analysis. *Intelligent Data Analysis*, 1(1):3–23.

García, S., Luengo, J., and Herrera, F. (2015). *Data preprocessing in data mining*, volume 72. Springer.

García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., and Herrera, F. (2016). Big data preprocessing: methods and prospects. *Big Data Analytics*, 1(1).

Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B., and Vanschoren, J. (2019). An open source AutoML benchmark. *arXiv preprint arXiv:1907.00909*.

Grinsztajn, L., Oyallon, E., and Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Kowarik, A. and Templ, M. (2016). Imputation with the R package VIM. *Journal of Statistical Software*, 74(7):1–16.

Kozak, A. and Ruczyński, H. (2023). forester: A Novel Approach to Accessible and Interpretable AutoML for Tree-Based Modeling. In *2nd International Conference on Automated Machine Learning 2023*.

Kratzer, G. and Furrer, R. (2018). varrank: an r package for variable ranking based on mutual information with applications to observed systemic datasets.

Kursa, M. B. and Rudnicki, W. R. (2010). Feature Selection with the Boruta Package. *Journal of Statistical Software*, 36(11):1–13.

Louppe, G., Wehenkel, L., Sutera, A., and Geurts, P. (2013). Understanding variable importances in forests of randomized trees. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Obaid, H. S., Dheyab, S. A., and Sabry, S. S. (2019). The impact of data pre-processing techniques and dimensionality reduction on the accuracy of machine learning. In *2019 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference (IEMECON)*, pages 279–283.

Peng, H., Long, F., and Ding, C. (2005). Feature Selection Based On Mutual Information: Criteria of Max-Dependency,Max-Relevance, and Min-Redundancy. *IEEE transactions on pattern analysis and machine intelligence*, 27:1226–38.

Probst, P., Bischl, B., and Boulesteix, A.-L. (2018). Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *Journal of Machine Learning Research*.

Ruczyński, H., Kozak, A., Słowakiewicz, P., Grudzień, A., and Biecek, P. (2023). *forester*. R package version 1.1.4.

Sulaiman, M. A. and Labadin, J. (2015). Feature selection based on mutual information. In *2015 9th International Conference on IT in Asia (CITA)*, pages 1–6.

Çetin, V. and Yıldız, O. (2022). A comprehensive review on data preprocessing techniques in data analysis. *Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi*, 28(2):299–312.

Submission Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The main claims from the abstract mention that this paper studies the impact of preprocessing strategies on tree-based models, and inform about the outline of the study. We describe our methodology in detail in Sections 2, 3, 4, and the results are presented in Section 5.

   (b) Did you describe the limitations of your work? [Yes] See Section 6.

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6.

   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? `https://2022.automl.cc/ethics-accessibility/` [Yes] We believe that our paper conforms to the guidelines.

2. If you ran experiments...

   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources)? [Yes] Each preprocessing strategy was evaluated on the same datasets, and with the usage of the same measures, namely the win-ratio, and preprocessibility (based on accuracy and R^2.)

   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning)? [Yes] The evaluation details are mentioned in the Sections 3, 4, and Appendices C, D, E.

   (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] We did not explicilty repeat the experiments, but for each preprocessing strategy we trained 21 models of each engine with random search, which can be treated as repetitions of experiments.

   (d) Did you report the uncertainty of your results (e.g., the variance across random seeds or splits)? [N/A] We did not analyze the uncertainty of results, although the raw results are available via the Appendix E.

   (e) Did you report the statistical significance of your results? [No] Reporting the statistical significance is of no use for this study.

   (f) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] We used a tabular OpenML-CC18 benchmark for part of considered datasets, as described in Section 2, and Appendix B.

   (g) Did you compare performance over time and describe how you selected the maximum duration? [No] We did not compare the performance over time.

   (h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The computational resources are described in the Appendix F.

   (i) Did you run ablation studies to assess the impact of different components of your approach? [Yes] This study can be treated as the ablation study of preprocessing strategies of the *forester* package.

3. With respect to the code used to obtain your results…

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit versions), random seeds, an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] Such materials are present on our GitHub repository, for the details see Appendix E.

   (b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [No] We did not include such an example, although the user is capable of doing so on their own, by modifying our notebooks.

   (c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [Yes] The code is available on the package's GitHub repository in the form of R Markdown notebooks, for the details see Appendix E.

   (d) Did you include the raw results of running your experiments with the given code, data, and instructions? [Yes] The semi-raw results are available on the package's GitHub repository, for the details see Appendix E.

   (e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [Yes] The code is available on the package's GitHub repository in the form of R Markdown notebooks, for the details see Appendix E.

4. If you used existing assets (e.g., code, data, models)…

   (a) Did you cite the creators of used assets? [Yes] A full list of the cited papers/tools is described in the references.

   (b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [Yes] See Section 2, we are using OpenML-CC18 and its data. We cited all data sources according to the guidelines of datasets on OpenML (and in OpenML-CC18).

   (c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Our data does not contain personally identifiable information or offensive content.

5. If you created/released new assets (e.g., code, data, models)…

   (a) Did you mention the license of the new assets (e.g., as part of your code submission)? [Yes] The new version of the *forester* package is shortly described in Appendix A, which includes its license.

   (b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [Yes] The new version of the *forester* package is shortly described in Appendix A, which includes a link to projects GitHub repository.

6. If you used crowd-sourcing or conducted research with human subjects…

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We did not do research with human subjects.

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] We did not do research with human subjects.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] We did not do research with human subjects.

7. If you included theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [N/A] We have no theoretical results.

(b) Did you include complete proofs of all theoretical results? [N/A] We have no theoretical results.

A The forester package

This study is entirely based on the *forester* package (Ruczyński et al., 2023; Kozak and Ruczyński, 2023), being Open-source (GPL-3 License) AutoML tool created for R users. For the sake of this study we modified it, mainly by adding the multiclass classification task and custom preprocessing module, described in detail in Appendix C. The current version of *forester* and other supplementary materials regarding the project are available on our GitHub repository[2], and the AutoML'24 branch[3] contains the package version 1.6.1 that was used for the study.

B Datasets

In this appendix we present the details concerning the dataset selection and modifications, mentioned in the Section 2.

Binary classification

Table 3 summarizes the characteristics of the employed binary classification tasks. Most of the tasks are imbalanced, and none of them originally contain identifier-like columns. The artificially modified datasets are *credit-g-mod*, and *phoneme-mod*.

Table 3: **The quantitative description of a subset of OpenML-CC18 benchmark**. These are the binary classification tasks with small to medium size, and moderate preprocessing issues used for the study.

Data set	Observations	Features	Static	Duplicate pairs	Missing fields	Dimensional issues	Correlation pairs	Imbalance	ID-like
banknote-authentication	1372	5	0	0	0	No	1	No	No
blood-transfusion-service-center	748	5	0	0	0	No	1	Yes	No
breast-w	699	10	0	0	16	No	9	Yes	No
credit-approval	690	16	0	0	37	No	1	No	No
credit-g	1000	21	0	0	0	No	0	Yes	No
diabetes	768	9	0	0	0	No	0	Yes	No
kr-vs-kp	3196	37	4	0	0	Yes	0	No	No
phoneme	5403	6	0	0	0	No	0	Yes	No
credit-g-mod	1000	26	2	2	271	No	2	Yes	Yes
phoneme-mod	5403	11	2	2	1477	No	2	Yes	Yes

Multiclass classification

Table 4 presents multiclass classification tasks, which are of moderate size, and stand out from the previous ones because of generally higher dimensionality. Due to this fact, in this case, we can

[2] https://github.com/ModelOriented/forester

[3] https://github.com/ModelOriented/forester/tree/AutoML'24

also observe more highly correlated columns than before. The artificially modified datasets are *satimage-mod*, and *car-mod*.

Table 4: **The quantitative description of a subset of OpenML-CC18 benchmark datasets**. These are the multiclass classification tasks with moderate size, but high dimensionality, and moderate preprocessing issues used for the study.

Data set	Observations	Features	Static	Duplicate pairs	Missing fields	Dimensional issues	Correlation pairs	Imbalance	ID-like
balance-scale	625	5	0	0	0	No	0	Yes	No
mfeat-karhunen	2000	65	0	0	0	Yes	0	No	No
satimage	6430	37	0	0	0	Yes	285	Yes	No
car	1728	7	0	0	0	No	0	Yes	No
dna	3186	191	0	0	0	Yes	0	Yes	No
satimage-mod	6430	42	2	2	1753	Yes	320	Yes	Yes
car-mod	1728	12	2	2	476	No	2	Yes	Yes
wine_quality	6497	12	0	0	0	No	1	Yes	No

Regression

Table 5 details the characteristics of the regression tasks used in the study. These tasks are significantly larger in size compared to the binary classification tasks and exhibit more instances of data corruption. In general, regression tasks also introduce a greater number of scenarios with dimensionality issues (excessive features) and imbalanced target variables. The artificially modified datasets are *elevators-mod*, and *kin8nm-mod*.

Table 5: **The quantitative description of a subset of OpenML datasets**. These are the regression tasks with medium to large size and major preprocessing issues used for the study.

Data set	Observations	Features	Static	Duplicate pairs	Missing fields	Dimensional issues	Correlation pairs	Imbalance	ID-like
2dplanes	40768	11	0	0	0	No	0	No	No
bank32nh	8192	33	0	0	0	Yes	0	Yes	No
elevators	16599	19	2	0	0	No	11	Yes	No
kin8nm	8192	9	0	0	0	No	0	No	No
Mercedes-Benz Greener-Manufacturing	4209	378	145	134	0	Yes	522	No	Yes
elevators-mod	16599	24	4	2	4519	No	15	Yes	Yes
kin8nm-mod	8192	14	2	2	2231	No	2	No	Yes

Dataset modifications

As mentioned in the Section 2 in order to include more real-life data, we artificially worsened the quality of 6 benchmark datasets in the following way:

- We added a numeric column id (ID-like column), whose values are in the range from 0 to the number of rows,
- We added two static columns, where the first one consists of repeated number 1, whereas the second one mostly has letter *a*, and sometimes letter *b*,
- We randomly (with seed) sampled two original columns, and created duplicates of them,
- We randomly selected three original columns (with the exclusion of the ones used in the previous step) and introduced missing values to them. The amount of missing elements per column is 5%, 10%, and 15%.

The outcoming dataset is named after the original one with the suffix '-mod', for example the modification of *credit-g* is *credit-g-mod*.

C Custom preprocessing

The *forester's* custom preprocessing focuses on three major data preparation areas, being the heuristic removal of corrupted features or observations, missing data imputation, and feature selection. In this Appendix we describe the methods available for the users via considered functionality.
The sub-module responsible for removing irrelevant features employs up to six separate criteria to determine whether a column or row should be eliminated. These criteria are user-configurable by setting the appropriate threshold parameters:

1. Duplicated columns - identify and remove columns that are identical to others,
2. Id-like columns - identify and remove columns resembling identifiers based on either a provided or a default list,
3. Static columns - remove features exhibiting a high percentage (governed by user-defined threshold k) of identical values,
4. Sparse columns - remove features with a low percentage (governed by user-defined threshold l) of non-empty values,
5. Corrupted rows - remove observations with either a low percentage (governed by user-defined threshold m) of non-empty values or an empty target value,
6. Highly correlated columns - iteratively remove the minimal number of features exceeding a user-defined correlation threshold (n) to achieve the desired outcome of reducing multicollinearity.

Parameters k, l, m, n are threshold parameters provided by the user, corresponding to the appropriate criterion.
The second sub-module addresses missing values within the dataset by selecting an appropriate imputation method based on user selection. Four algorithms are available. With the median-other, and median-frequency approaches the numeric features are imputed with the median value, while categorical features are imputed with either the "other" string or the most frequent value, depending on the chosen method. It is also possible to choose more advanced algorithms such as K-Nearest Neighbors (KNN), (Batista and Monard, 2002) from *VIM* package (Kowarik and Templ, 2016), or Multivariate Imputation by Chained Equations (MICE) (Buuren and Groothuis-Oudshoorn, 2011) and determine their major parameters.
The final component of the preprocessing module focuses on feature selection. The package offers four state-of-the-art (SOTA) algorithms, being Mutual Information (MI) (Sulaiman and Labadin, 2015) from *varrank* package (Kratzer and Furrer, 2018), Boruta (Kursa and Rudnicki, 2010), Monte Carlo Feature Selection (MCFS) (Dramiński et al., 2008), and Variable Importance (VI) (Louppe et al., 2013). The interface of `custom_preprocessing()` function allows users to fine-tune the most crucial parameters of these methods. The selection of algorithms caters to varying time complexities, with MI and MCFS being faster compared to the more time-consuming Boruta and VI.

D Preprocessing strategies

In this section we describe the reasoning which stands behind achieving 38 preprocessing strategies stated in the Section 3. If we considered all the possible options, including the aggregation of removal methods, we would end up with 60 (3 removal x 4 imputation x 5 feature selection) combinations. However, due to the limited time and resources we decided to narrow it down to 38 strategies, which exclude presumably the most expensive configurations. This way we ended up with 7 exploration areas described in the list below:

- **RM** - tests all removal strategies, where other modules are set to the most basic options: median-other for imputation, and none for feature selection,
- **IMP** - tests all imputation methods, where other modules are set to the most basic options: minimal for removals, and none for feature selection,
- **FS** - tests all feature selection methods, where other modules are set to the most basic options: minimal for removals, and median-other for imputation,
- **RM + IMP** - tests medium and maximum removal options combined with all imputation methods,
- **RM + FS** - tests medium and maximum removal options combined with all feature selection methods,
- **IMP + FS** - tests all imputation methods with MI and Boruta feature selection,
- **RM + IMP + FS** - tests medium and maximum removal options combined with median-frequency, median-other and KNN imputation, and MI and Boruta feature selection.

We have to notice, that when the MI-based feature selection method was used, we selected the 'estevez' (Estévez et al., 2009) method for binary classification, and 'peng' (Peng et al., 2005) for regression tasks. To ensure a detailed insight into particular strategies, in Table 6 we present every configuration. The horizontal lines represent the segments described above.

E Resources

In this Appendix we include the information regarding the reproducibility, focusing on the initial data, source codes, and raw results. These files are available from the *foresters* GitHub repository[4]. The directory regarding this study contains:

- RData files containing datasets used in the study,
- Rmd notebooks, starting with 01-08 which indicating the order of their execution,
- MSc_processed_results directory containing the final results after the model training.

The structure of files is described in detail in README.md file included in the aforementioned directory.

F Computational resources

During the thesis development process, two different machines were used. The first is author's private PC with 32GB of RAM, CPU: 11th Gen Intel(R) Core(TM) i7-11700KF @ 3.60GHz (16 cores), and the GPU: NVIDIA GeForce RTX 3070 Ti. The package development, testing, and the results analysis, including plots were conducted there.
For heavier computations, such as data preprocessing, model training, and results aggregations we were able to use the PLGrid[5] infrastructure. We managed to acquire the proper computational grant there, which enabled conducting such a wide study. The computational grants we used, were both on Ares supercomputer. The first one was a pilot grant where we used all available 1000 hours of computational time. The main grant however has a limit of 25 000 hours where additional 1 300 hours were consumed. Large storage capabilities also enabled us to save over 150 GB of results there.

[4] https://github.com/ModelOriented/forester/tree/AutoML'24/docs/articles/AutoML24Workshop_MScThesis

[5] https://www.plgrid.pl

According to their website PLGrid is a nationwide computing infrastructure designed to support scientific research and experimental development across a wide range of scientific and economic fields. PLGrid provides access to supercomputers, quantum computers, specialized accelerators for artificial intelligence, cloud computing, disk storage, optimized computing software and assistance from experts from the entire Poland.
Ares is a state-of-the-art supercomputer with a computing power of more than 3.5 PetaFlops (CPU) and 500 TFlops (GPU). With its innovative direct liquid-cooling technology for CPUs and RAM modules, it ranks among the most energy-efficient computing systems of its class in the world. It offers: disk system consisting of more than 11 PB, 17 824 computing cores, a total of 147.7 TB RAM, and computing servers with the latest generation of Intel Xeon Platinum processors, divided into three groups with the following configurations: 532 servers supplied with 192 GB of RAM, 256 servers with 384 GB of RAM, 9 servers supplied with 8 NVIDIA Tesla V100 cards.

Table 6: Compact representation of the preprocessing strategies. The abbreviations from the first column are: RM - removal, IMP - imputation, and FS - feature selection.

Strategy	Removal	Imputation	Feature Selection
RM IMP	Minimal	Median-other	None
RM RM + IMP	Medium	Median-other	None
RM RM + IMP	Maximal	Median-other	None
IMP	Minimal	KNN	None
IMP	Minimal	Median-frequency	None
IMP	Minimal	MICE	None
FS IMP + FS	Minimal	Median-other	Boruta
FS IMP + FS	Minimal	Median-other	MI
FS	Minimal	Median-other	MCFS
FS	Minimal	Median-other	VI
RM + IMP	Medium	KNN	None
RM + IMP	Medium	Median-frequency	None
RM + IMP	Medium	MICE	None
RM + IMP	Maximal	KNN	None
RM + IMP	Maximal	Median-frequency	None
RM + IMP	Maximal	MICE	None
RM + FS RM + IMP + FS	Medium	Median-other	Boruta
RM + FS RM + IMP + FS	Medium	Median-other	MI
RM + FS RM + IMP + FS	Maximal	Median-other	Boruta
RM + FS RM + IMP + FS	Maximal	Median-other	MI
RM + FS	Medium	Median-other	MCFS
RM + FS	Medium	Median-other	VI
RM + FS	Maximal	Median-other	MCFS
RM + FS	Maximal	Median-other	VI
IMP + FS	Minimal	KNN	Boruta
IMP + FS	Minimal	Median-frequency	Boruta
IMP + FS	Minimal	MICE	Boruta
IMP + FS	Minimal	KNN	MI
IMP + FS	Minimal	Median-frequency	MI
IMP + FS	Minimal	MICE	MI
RM + IMP + FS	Medium	KNN	Boruta
RM + IMP + FS	Medium	Median-frequency	Boruta
RM + IMP + FS	Medium	KNN	MI
RM + IMP + FS	Medium	Median-frequency	MI
RM + IMP + FS	Maximal	KNN	Boruta
RM + IMP + FS	Maximal	Median-frequency	Boruta
RM + IMP + FS	Maximal	KNN	MI
RM + IMP + FS	Maximal	Median-frequency	MI