The impact of data preparation on the quality of tree-based models created with AutoML forester package

Hubert Ruczyński[1]

[1]Warsaw University of Technology

Abstract Automated Machine Learning (AutoML) solutions are increasingly popular, as they allow data scientists to train high-quality machine learning models with minimal effort. However, the majority of AutoML solutions are developed in Python, leaving R users with limited, and overly complex tools. In this paper, we introduce *forester*, an open-source AutoML package implemented in R that is designed for training high-quality tree-based models on tabular data. The *forester* supports regression, binary classification, and newly implemented survival analysis tasks. The focus put on a single model family, gives us an opportunity to derive conclusions about its behaviour.

Additionally, we introduce a custom preprocessing module, which creates an opportunity to validate a common belief that tree-based models do not require any data preprocessing. We answer this question by conducting a thorough ablation study, of the *forester* package, where we evaluate the impact of dozens of preprocessing strategies. Obtained results let us believe that in the case of the tree-based models family some methods, prove to be more efficient than, others. Finally, we cannot fully agree with the presented thesis, and we provide the reasons supporting our belief.

1 Introduction

The use of machine learning is rapidly expanding in a variety of fields, including medicine, finance, and tourism. For example, machine learning models are being used to assist doctors in cancer diagnosis (Shimizu and Nakayama, 2020), financiers in credit evaluation (Jorge et al., 2022), and tourists in personalized recommendation systems (Fararni et al., 2021). The increasing demand for machine learning models has led to research on automatically developing tools to build artificial intelligence based solutions.

Machine learning models come in a variety of forms, from simple decision rules to complex neural network structures, or even large language models (LLMs), such as ChatGPT (Bavarian et al., 2022). When applied to tabular data, a wide range of models are available, including decision trees, linear or logistic regression, random forests, support vector machines (SVMs), and neural networks. However, tree-based models are the most widely used, primarily due to their high predictive efficiency. A simple decision tree model cannot produce satisfactory results, however, using multiple trees to create a random forest can significantly improve predictive power which makes them superior even to complex neural networks (Caruana et al., 2008; Grinsztajn et al., 2022). The efficiency of tree-based models is due to their ability to learn non-linear relationships between the features and the target variable. Additionally, tree-based models are relatively easy to interpret, which can be helpful for understanding the factors that influence the target variable.

The automation of the machine learning process can make it accessible to those who are new to data analysis and modeling. It can also speed up the work of experienced data scientists, who can produce at least baseline models with a single line of code.

The objective of AutoML is not limited to producing models. It can also include many other components. For example, the CRoss Industry Standard Process for Data Mining (CRISP-DM)

(Wirth and Hipp, 2000) is the most common methodology for data mining, analytics, and data science projects. However, the basic framework of an automatic machine learning system is the preparation of models based on data entered by the user. This process can be extended in various directions, such as preliminary analysis of the data, optimization of hyperparameters, or data preprocessing.
Another aspect of AutoML, which is often discarded by larger packages is the impact of data preprocessing methods on outcomes. It is often repeated that most of the data scientists' work concerns data preprocessing and preparation stage, and it is the first step for obtaining meaningful outcomes. There are only a few solutions such as the Atlantic (Santos and Ferreira, 2023) package in Python that attempt to automate this part of the machine learning pipeline. Moreover, there are only a few papers that try to evaluate various preprocessing strategies in order to show which ones tend to be more effective. Studies like (Symeonidis et al., 2018), evaluate the preprocessing impact on sentiment analysis tasks, but classical tabular data problems, lack formal evaluation of various methods.
In this paper, we present a list of enhancements made to the *forester* package (Kozak and Ruczyński, 2023), such as; an improved reporting function that provides more detailed information about the performance of the model, the implementation of a new machine learning task being the survival analysis, which lacks AutoML solutions, especially in R, and the introduction of an advanced preprocessing function that allows for more control over the data cleaning and feature selection (FS) process. Moreover, we use the advanced preprocessing function to conduct a thorough ablation study of preprocessing components, which shows us the impact of each component on the performance of tree-based models. This way we attempt to fill a rather unexplored field of automated preprocessing and its evaluation. The implementation of the *forester* package is available in our GitHub repository[1]. The software is open source and contains comprehensive documentation with examples of use.

2 Related works

AutoML has become increasingly popular in Python, with *Auto-WEKA* (Thornton et al., 2013) being one of the first solutions, and *Auto-sklearn* (Feurer et al., 2015, 2022) with *TPOT* (Tree-Based Pipeline Optimization Tool) (Olson et al., 2016) following this trend. However, there are only a few AutoML packages available in R.
One of these packages is *H2O* (LeDell et al., 2022). It is an open-source, in-memory, distributed, fast, and scalable machine learning and predictive analytics platform that produces a ranked list of models that can be easily exported for use in a production environment. *H2O* provides an easy-to-use interface that automates the training of multiple candidate models with various engines. Similarly to the *forester* package, *H2O* offers model training, tuning, and extensive evaluation of the results. However, there are some key distinctions between the two packages.
The *forester* is distinguished from *H2O* by its custom preprocessing module and focus on tree-based models. The *H2O* only includes target encoding, which is in the experimental stage, whereas, in the *forester*, we have more accurate, extensive, and highly customizable preprocessing.
Another widely-used R AutoML package is *mlr3* (Lang et al., 2019). It provides a framework for classification, regression, survival analysis, and other machine learning tasks. The *mlr3* lets us also perform hyperparameter tuning and feature selection. The package is well-documented, contains many functions and models, and provides many capabilities. However, it is different from a typical AutoML package because it requires knowledge of building machine learning models.
A more detailed comparison of the *forester* package with *H2O* and *mlr3* is presented in the previous paper (Kozak and Ruczyński, 2023).

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

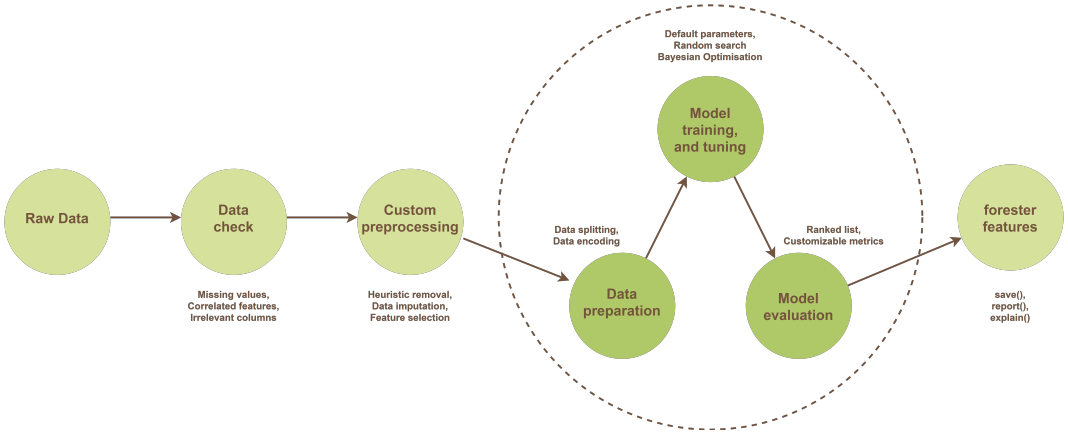


Figure 1: A diagram presenting the *forester* pipeline. With the *forester* you can analyze poor-quality data with the in-built data check, which points to possible issues, and based on the outcomes select proper preprocessing strategies from custom preprocessing module. The train function begins with data preparation, so there is no need for manual data encoding. In the next step, the models are trained with default and random searched parameters and tuned with a Bayesian optimization algorithm. In the end, trained models are evaluated and presented as a ranked list. In addition, the package offers the user additional features.

3 *forester* AutoML

The *forester* is an AutoML package automating the machine learning pipeline, starting from the data preparation, through model training, to the interpretability of the results, which is represented in Figure 1. This way, it minimizes the user's time performing basic and often repetitive activities related to the machine-learning process. In this section we will summarize major characteristics of the *forester* described in the previous paper, and provide information about latest improvements of the package.

3.1 ML pipeline

Despite the high automation of the pipeline shown in Figure 1, it exposes multiple parameters which advanced data scientists can use to customize the model creation. The latest improvements to the pipeline is an addition of custom preprocessing step, and providing a support for survival analysis task. The whole package relies on the six pillars described in this section.

1. **Data check**
   The first one, called data check, concerns a data preprocessing phase. This module provides a list of possible issues existing within used dataset. The analysis of it's results might help us choose proper settings for the custom preprocessing module.

2. **Custom preprocessing**
   The second pillar is the major contribution in this paper in terms of improving the *forester* package. The `custom_preprocessing()` function is a high level module which simplifies writing the code for well-suited prepossessing pipeline. The module covers three major parts of data quality enhancements: the removal of corrupted features, imputation of missing data, and feature selection.

   The sub-module responsible for the deletion of irrelevant features uses up to 6 separate criterion which determine whether a columns or row should be removed or not. The user can determine which rules to use. With this module the user can remove:

(a) Duplicated columns,

(b) Id-like columns, determined in provided, or default list,

(c) Static columns - the features which have more than $k\%$ of the same values,

(d) Sparse columns - the features which have less than $k\%$ of not empty values,

(e) Corrupted rows - these are the observations which have less than $k\%$ of not empty values, or have an empty target value,

(f) Highly correlated columns - the features which have correlation higher than k. The method iteratively removes minimal possible number of columns to achieve the desired outcome.

Parameter k is a threshold parameter provided by the user, corresponding to appropriate criterion.

Second sub-module determines which imputation method to use if there are any missing values inside the dataset. The user can choose from 4 algorithms. With the median-other, and median-frequency approaches the numeric features are imputed with median value, whereas the categorical ones with the 'other' string or the most frequent value. It is also possible to choose more advanced algorithms such us 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 last part of preprocessing module is the choice of feature selection method. 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 obviously enables the user to fine tune the most important parameters of these methods. The selection of algorithms covers various time-complexity, where MI, and BORUTA are relatively fast, whereas MCFS and VI are time-consuming.

The main strengths of discussed module are high customization to the user's need provided by a wide range of tunable parameters, and the independence from main training pipeline. Due to the fact that `custom_preprocessing()` function is used outside of the `train()` function, its applicability isn't limited to the *forester* package only, and the users are free to use it with other solutions. The Listing 1 shows us how the interface of the function looks like. The parameteres are grouped into three lists corresponding to described modules. In order to design proper preprocessing strategy it is advised to firstly run a `data_check()`, and later turn on proper modules.

3. **Data preparation**
   As previous step is not obligatory, at the beginning of this stage the package conducts the most basic preprocessing, if it is needed. Moreover, every model in the *forester* package requires a different data format which is also prepared inside the main function.

4. **Available tasks**
   Originally, the *forester* focused only on binary classification, and regression tasks, as those two are the most common problems solved by data scientists. Current version however, aims to be a more versatile tool, thus we allowed the users to solve survival analysis tasks, with the usage of random survival forests (Ishwaran et al., 2008), (Ishwaran and Kogalur, 2007) from *randomForestSRC* package (Ishwaran and Kogalur, 2023). The module is in early development stage, but it provides all necessary means, to automatically solve survival analysis tasks. An example of usage is available in our GitHub repository[2].

[2] `https://github.com/ModelOriented/forester/misc/experiments/survival_analysis_demo.html`

```
preprocessed_data ← custom_preprocessing(
    data          = lisbon,
    y             = 'Price',
    na_indicators = c(''),
    removal_parameters = list(
        active_modules = c(
            duplicate_cols = TRUE, id_like_cols    = TRUE,
            static_cols    = TRUE, sparse_cols     = TRUE,
            corrupt_rows   = TRUE, correlated_cols = TRUE),
        id_names = c('id', 'nr', 'number', 'idx',
                     'identification', 'index'),
        static_threshold           = 0.99,
        sparse_columns_threshold   = 0.3,
        sparse_rows_threshold      = 0.3,
        high_correlation_threshold = 0.7
    ),
    imputation_parameters = list(
        imputation_method = 'median-other',
        k = 10,
        m = 5
    ),
    feature_selection_parameters = list(
        feature_selection_method = 'VI',
        max_features       = 'default',
        nperm              = 1,
        cutoffPermutations = 20,
        threadsNumber      = NULL,
        method             = 'estevez'
    ),
    verbose = FALSE)
```

Listing 1: Exemplary presentation of the custom preprocessing interface, showing all selectable parameters. The functions design underlines the division into 3 sub-modules being parameters removal, data imputation, and feature selection.

5. **Model training and tuning**
   The third and most important pillar of the *forester* package is model training and tuning. The solution focuses on the tree-based model family, which has been shown to have high-quality performance for various tabular data tasks. We've limited the choice of engines to: decision tree (Quinlan, 1986) from *partykit* Hothorn and Zeileis (2015), random forest (Breiman, 2001) from *ranger* (Wright and Ziegler, 2017), XGBoost (Chen and Guestrin, 2016), LightGBM (Ke et al., 2017), and CatBoost (Prokhorenkova et al., 2018) for binary classification, and regression tasks. These models are trained with three approaches: using the default parameters, performing the random search algorithm within the predefined parameter space, and running an advanced Bayesian Optimization algorithm for fine-grained tuning.

6. **Model evaluation**
   The last pillar is the automatic evaluation of the trained models. The *forester* package evaluates trained models with multiple metrics, such as accuracy, area under the receiver operating characteristic curve (AUC), F1 score, recall, precision, sensitivity, specificity, and balanced accuracy for the binary classification tasks. For the regression models it calculates Mean Squared

Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), R^2, and Mean Absolute Deviation (MAD). The survival analysis task is evaluated with the usage of Brier Score, and Concordance Index. The results are later presented as a ranked list sorted by the selected metric.

3.2 Extensive features

One of the most important goals for the *forester* package is the convenience of use and helping the users to focus more on analyzing the results instead of writing the code. To obtain such a user-friendly environment, the *forester* offers plenty of additional features useful for data scientists.

1. **Model explanations**
   One of the most popular, emerging topics in machine learning landscape is explainable artificial intelligence (XAI). Various tools, such as *DALEX* (Biecek, 2018) or *iml* (Molnar et al., 2020) allow to create explanations of how trained models work. To improve the understanding of created models, and increase the level of trust in the outcomes, the *forester* provides a wrapper for the DALEX explainer compatible with returned models.

2. **Saving the outcomes**
   The package also contains a custom save function which lets us easily save whole objects provided during the training.

3. **Automated report**
   The proposed solution includes a module that automatically generates a report to help users quickly and easily analyze the training results. The main goal of this feature is to ensure that every user can easily assess the quality of the trained models. The report consists of basic information about the dataset, a data check report, a ranked list of the top ten models, visualizations concerning model quality, and thorough descriptions of how to read the presented plots. An example report for the *Compas* dataset is provided in Appendix I.

   The visualizations are divided into two groups: the first group compares the outcomes of different models, which helps users to decide which model is the best. For example, the train vs test plot, lets us choose model which does not have the best performance, but is less overfitted.

   The second group of visualizations focuses on the model with the best performance, and its most prominent feature is a feature importance plot. This visualization helps users to understand which variables are the most important for the model, which can help them to evaluate its correctness. It is worth noting that the reports, are different for binary classification and regression tasks, due to dissimilar tasks characteristics.

3.3 Performance

The paper introducing the *forester* also conducts a study on package's performance. It is compared to the *H2O* framework on 8 small to medium binary classification tasks from the OpenML-CC18 benchmark, and 7 medium to large regression tasks from OpenML (Vanschoren et al., 2013). The details describing the experiment, can be found in Section 6 of the previous paper, however we will describe the most important conclusions of this study, and the datasets will be thoroughly described in the next section.
The results presented in Figure 2 and Figure 3 indicate that the *forester* creates not only competitive, but even slightly better results in comparison to *H2O*. This trend is especially visible for the regression tasks. Moreover, the observed results were achieved two times faster than the one's of *H2O*. However, it is worth noticing that presented results were achieved with version 1.2.1, which didn't include the module with advanced preprocessing, so the results could have been even better now.

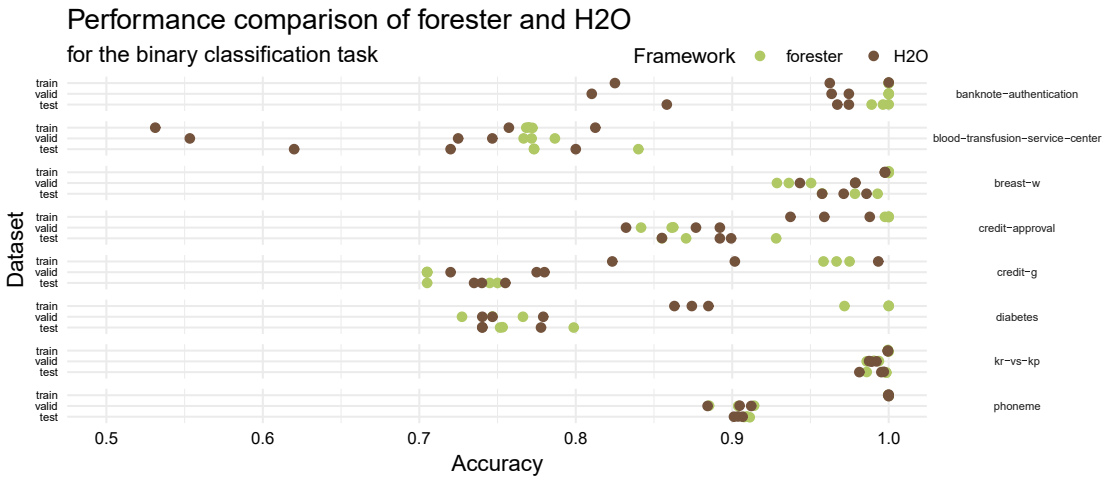


Figure 2: Performance comparison for *forester* and *H2O* frameworks for the datasets described in Table 1. Every experiment is conducted 3 times, which results are three observations visible on the plot for each dataset. Note that in some cases the dots might overlap. This plot clearly shows us that the *forester* performs better than the *H2O* package on the provided tasks, which confirms that it is a highly competitive framework.

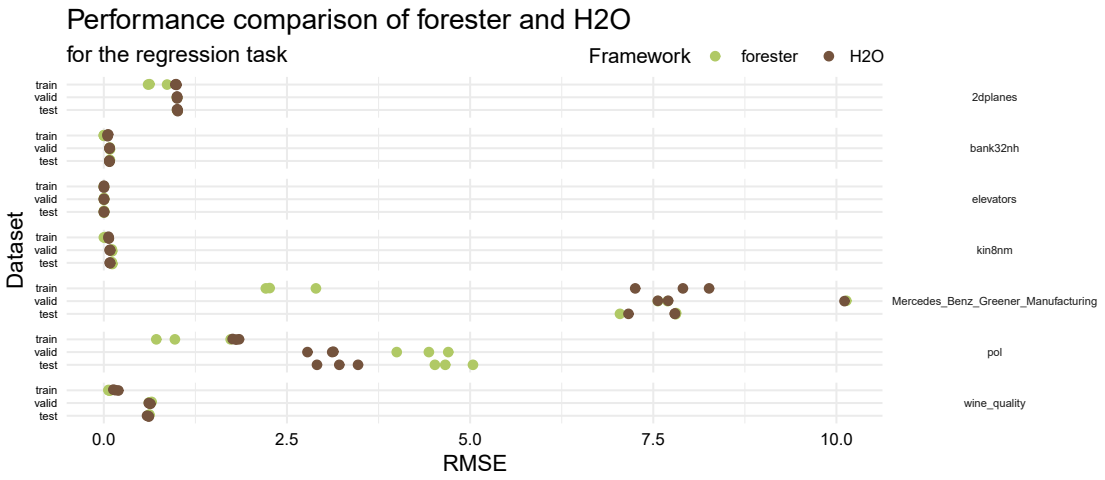


Figure 3: Performance comparison for *forester* and *H2O* frameworks for the datasets described in Table 2. Every experiment is conducted 3 times, which results are three observations visible on the plot for each dataset. Note that in some cases the dots might overlap. This plot shows us that the *forester* performs comparably to the *H2O* package on the provided tasks, which confirms that it is a highly competitive framework.

4 Ablation study

Although the performance comparison to other AutoML tools is of utmost importance and provides a necessary assessment of achievable results, it cannot grasp the nuances such as the impact of preprocessing on achieved outcomes. Such evaluation is extremely important, as advanced data preparation is one of the most prominent features missing in other available solutions. The ablation study presented in this section focuses on the impact of various preprocessing scenarios generated with `custom_prepossessing()` function. The experiments were conducted on 15 machine learning tasks, the same that was used for the performance evaluation, and it focuses on two major aspects of modeling: time complexity, and models' performance.

4.1 Datasets description

To provide continuity to the previous article, we have decided to use the same datasets for the ablation study. Moreover, the datasets suited the study perfectly, as it focuses on small to medium datasets mostly, which was crucial due to the limited computational resources described in Appendix A. The selection of binary classification tasks comes from the OpenML-CC18 benchmark and regression tasks from OpenML. Described datasets, especially the regression ones contained various issues lowering the data quality which is a perfect environment for testing the preprocessing methods. The data frames are described in Table 1 for binary classification, and in Table 2 for regression tasks. The raw data and process of obtaining them are described in Appendix B.

Binary classification
As presented in Table 1 the binary classification tasks were small to medium sizes, with *kr-vs-kp* being the largest. Only one dataset had static columns, and not a single task suffered from the duplicates, however, two of them contained missing fields. Additionally, the aforementioned *kr-vs-kp* had a problem with too big a number of columns, which will be discarded by tree-based models. Unfortunately, half of the tasks had at least one pair of highly correlated columns, where *breast-w* had 9 highly correlated pairs. Finally, most of the tasks were imbalanced, but none of them contained id-like columns.

Data set	Rows	Columns	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

Table 1: The quantitative description of a subset of OpenML-CC18 benchmark datasets used for the ablation study, and the evaluation of *forester* package. These are the binary classification tasks with small to medium size, and moderate preprocessing issues.

Regression

Table 2 presents regression tasks, which were much bigger than the previous ones, and presented more examples of corrupted data. The most problematic datasets were *Mercedes_Benz_Greener_Manufacturing*, and *pol*. These tasks were the largest ones and contained lots of static columns, as well as an abundance of duplicate ones. *Mercedes_Benz_Greener_Manufacturing* additionally had a huge amount of highly correlated columns, and at least one id-like column. In general regression tasks also introduced more scenarios with dimensionality issues, as well as the imbalanced target variables.

Data set	Rows	Columns	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
pol	15000	49	22	156	0	Yes	2	Yes	No
wine_quality	6497	12	0	0	0	No	1	Yes	No

Table 2: The quantitative description of a subset of OpenML datasets used for the ablation study, and the evaluation of *forester* package. These are the regression tasks with medium to large size, and major preprocessing issues.

4.2 Preprocessing strategies

In order to conduct the ablation study we had to prepare various preprocessing strategies. However, we were not able to cover the full space of possibilities. If we wanted to do so, even without tuning the parameters such as threshold, we would end up with 576 approaches, derived from 36 removal, 4 imputation, and 4 feature selection strategies. To end up with a reasonable amount of outcoming datasets, we had to limit this number significantly, thus we prepared 3 removal strategies:

- **Minimal (min)** - It is a baseline method that removes only the observations that do not have the target value,
- **Medium (med)** - 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 (max)** - It behaves the same as the medium option, however, it also removes the columns with a correlation higher than 0.7.

With such strategies, we were able to obtain the results where removal does not have a major impact (min), where we use safe and reasonable options (med), and where we additionally apply unstable highly correlated column removal. With those limitations, we would end up with 48 datasets. Even though, we have decided to limit it even further. The feature selection method is in some cases extremely time-consuming, thus we limited the number of experiments with those. Finally, we ended up with 39 preprocessing strategies 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. (3)
- **Imp** - tests all imputation methods, where other modules are set to the most basic options: min for removals, and none for feature selection. (4)

- **FS** - tests all feature selection methods, where other modules are set to the most basic options: min for removals, and median-other for imputation. (4)
- **RM + Imp** - tests med and max removal options combined with all imputation methods. (8)
- **RM + FS** - tests med and max removal options combined with all feature selection methods. (8)
- **Imp + FS** - tests median-other and KNN imputation methods with VI and BORUTA feature selection. (4)
- **RM + Imp + FS** - tests med and max removal options combined with median-other and KNN imputation, and VI and BORUTA feature selection. (8)

Additionally, we have to notice, that when the Mutual Information (MI) feature selection method was used, we selected the 'estevez' method for binary classification, and 'peng' for regression tasks. The table presenting the preprocessing strategies more compactly, the source code, raw preprocessing results, and raw duration of each step are described in Appendix C.

4.3 Model training

The next step was to conduct the model training. We used the *forester* package for this purpose. The `train()` function was used for all 39 preprocessed versions of 15 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 used random search as a tuning method, because random search is more beneficial for ablation studies than Bayesian optimization. Bayesian optimization would take too long to run, and the results could have been too perfect, making it difficult to see the effects of the preprocessing.
After the model training, we aggregated the results to create a convenient dataset for further analysis. For each dataset, we trained many models, but we only analyzed the measures aggregated by maximum, mean, median, and minimum values. Binary classification tasks were evaluated using accuracy, area under the curve (AUC), and F1 score. Regression tasks were evaluated using mean squared error (MSE), root mean squared error (RMSE), and mean absolute deviation (MAE). The results were also differentiated by the engine used for training. However, for the final analysis, we only used the results from the aggregation of all model types. The raw training results, as well as the preprocessed data and source code, are described in Appendix D.

4.4 Time analysis

An important aspect of our analysis is the time complexity of different approaches. The extended preprocessing module leads to more time-consuming computations, which could be spent on training the models. On the other hand, a thorough preparation step might result in removing a lot of unnecessary columns, so the model should be able to learn faster and achieve better results. Despite the absolute preprocessing time, another important aspect is the relative duration of training time. For example, if the training takes 1000 seconds, then preprocessing lasting 100 seconds is not as much as in the case when training takes 100 seconds. This section delves deeply into *foresters* time complexity. The source codes of both time and performance analysis are presented in Appendix E.

General time complexity

The first question to answer in time complexity analysis is whether different preprocessing strategies significantly affect the computational time of prepossessing and training. In order to determine that, we will say that the difference is significant if the values of 1st and 3rd quantiles differ more than 2 times. The results presented in Figure 4 indicate that preprocessing duration differs a lot for each

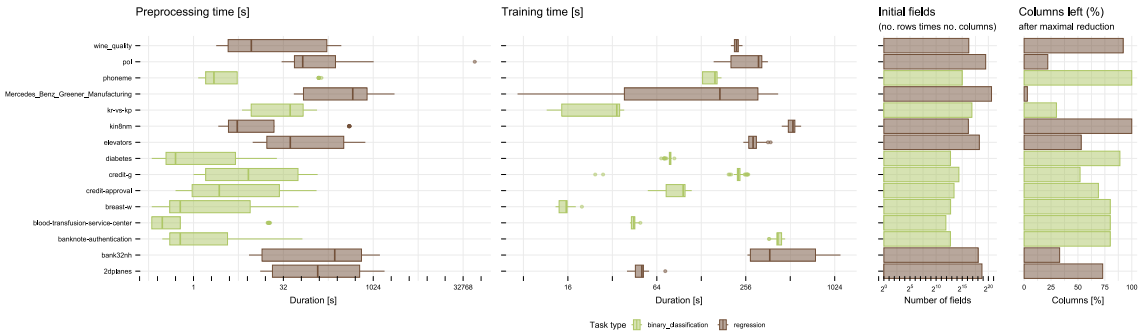


Figure 4: Visualization presenting the durations of preprocessing and training times for different regression and binary classification tasks. The first two subplots present major outcomes, whereas the last two are supplements, that describe each dataset shortly. The plots show that preprocessing time depends heavily on the size of the data, and its times are more varying than the ones of the training. Moreover, training durations depend on both task complexity and the number of columns left after preprocessing strategies.

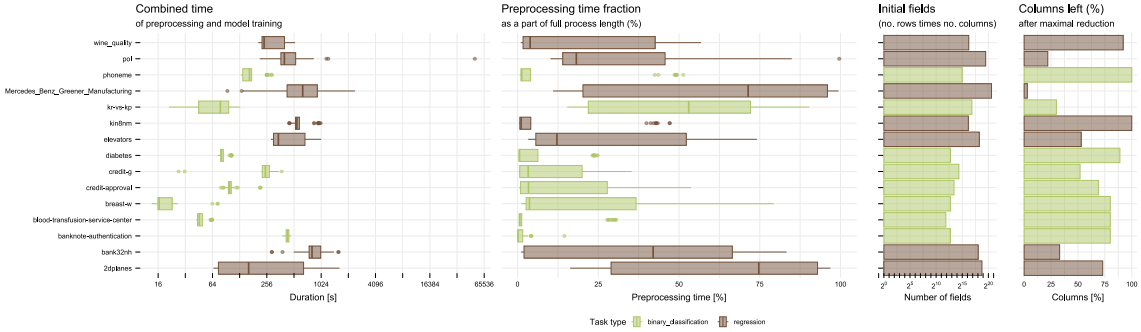


Figure 5: Visualization presenting the combined durations of preprocessing and training times, as well as the fraction of preprocessing in full process for different regression and binary classification tasks. The first two subplots present major outcomes, whereas the last two are supplements, that describe each dataset shortly. The plots show that combined time is less varying than the preprocessing time from Figure 4. Moreover, the second subplot indicates that in some cases preprocessing lasts disproportionally longer than the training.

dataset, however, the training times, are more similar to each other. Major training time differences happen only for the datasets that had the most columns removed during the preprocessing strategy that yielded the biggest dimensionality reduction. It indicates that vast dimensionality reduction leads to shorter training times. Additionally, we can witness that the preprocessing times are much shorter for the binary classification tasks, due to the smaller number of initial fields.
According to Figure 5 we can notice that the combined preprocessing and training time level off and the final sum of them are less varying than the preprocessing time itself. It proves, that longer preprocessing with more data removal leads to shorter training times. Additionally, the second subplot measuring the fraction of preprocessing time in the whole process shows us that in the case of smaller datasets without major feature removal, even extensive preprocessing strategies last much shorter than the training, whereas, for large tasks with stronger column removal, the data preparation can be much longer than the training. Moreover, for most dataframes, these relative proportions differ a lot, which means that there are some extremely time-consuming preprocessing strategies, which might be not optimal and we should be careful while selecting those.

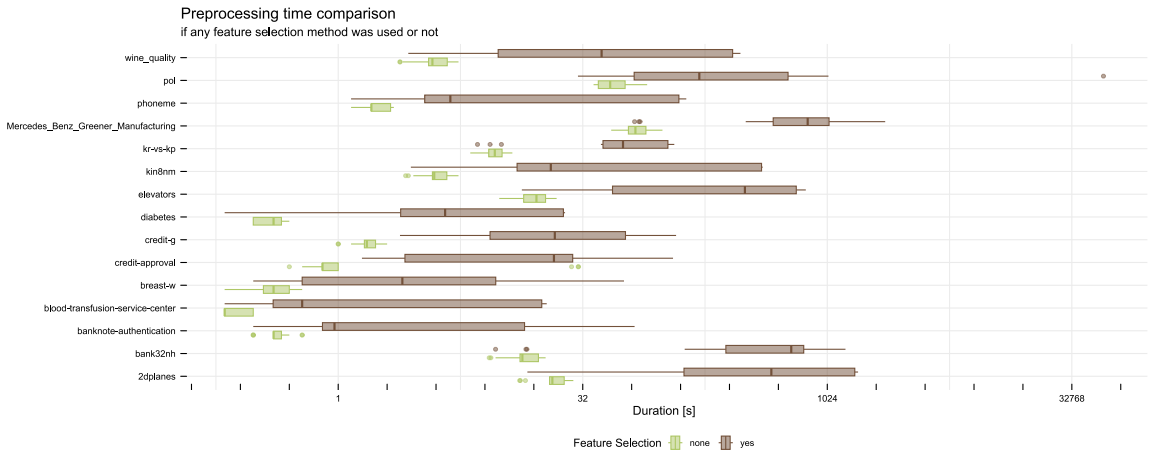


Figure 6: Preprocessing time comparison depending on the presence of any feature selection method. The plot shows that the most time-consuming part of the preprocessing pipeline is feature selection.

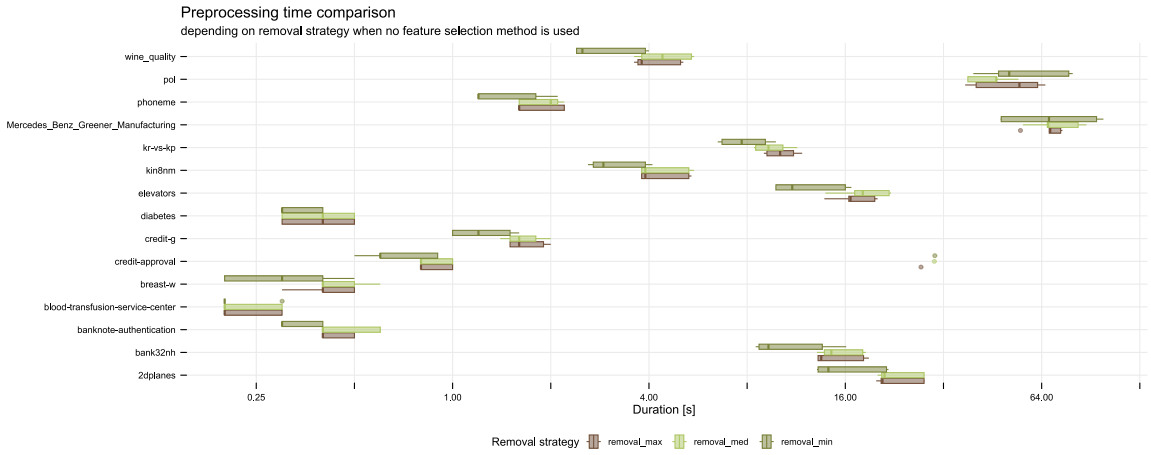


Figure 7: Preprocessing time comparison depending on the removal strategy, when no feature selection methods were used. The plot shows that the min strategy is slightly shorter than the other two and that there are no major differences between the medium and max approaches, even though the last one contains the removal of highly correlated columns.

Preprocessing time complexity

To follow up on the impact of the highly varying time complexity of different preprocessing strategies, we will delve deeply into different approaches and their durations. Based on those results we can evaluate which methods might be useful in real-life modeling, and which are too expensive. In Figure 6 we are presented with the durations divided by the presence of any feature selection method. It shows us, that lack of these elements leads to much shorter, and less varying times. Additionally, according to Figure 7 we can witness that in the case of strategies not involving feature selection, there aren't any major differences between different removal approaches. The min strategy is definitely shorter than the two others, which last roughly the same, even though the max approach includes the removal of highly correlated columns. A short analysis of the impact of imputation methods is present in the Appendix F, as there were only 2 datasets that had missing values.

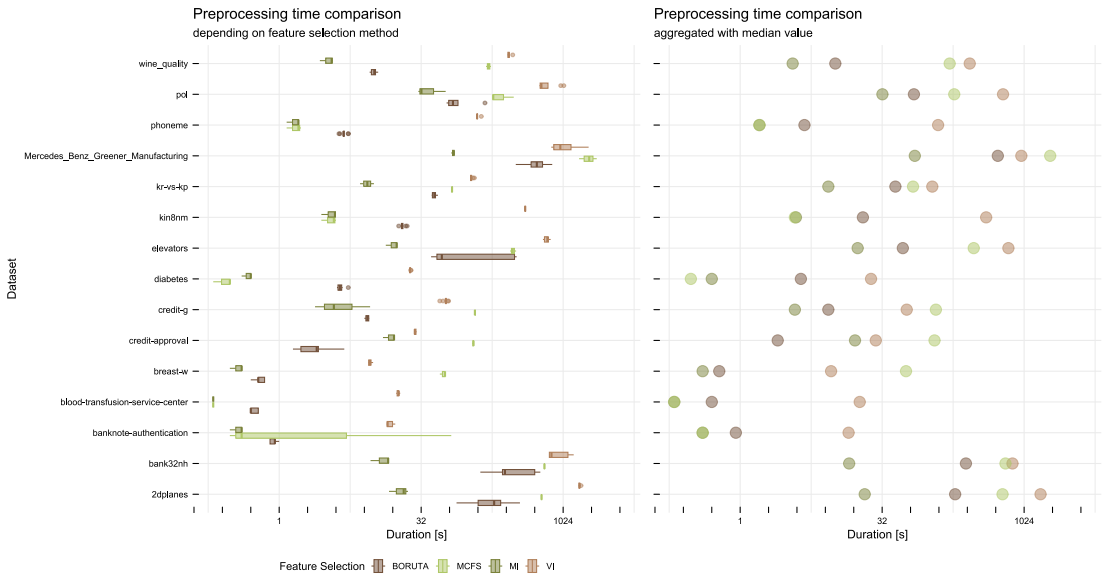


Figure 8: Preprocessing time comparison depending on feature selection with raw results.

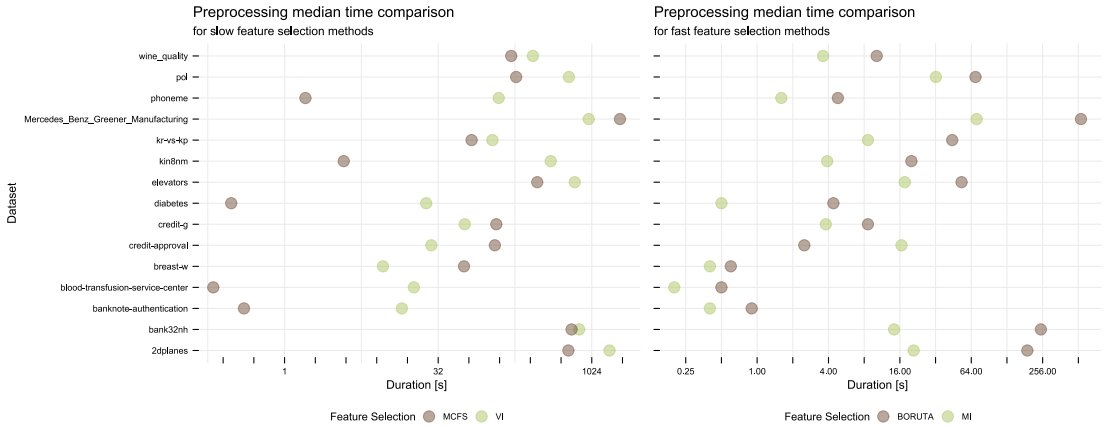


Figure 9: Preprocessing time comparison depending on feature selection with aggregated results.

Feature selection time complexity

As feature selection methods are the most time-consuming part of the preprocessing pipeline, we should take a closer look at them to discover more nuances about their performance. In Figure 8 we are presented with the preprocessing times depending on the used feature selection method as raw and aggregated times. As the first subplot is a bit unreadable, we decided to aggregate those times by its median value. This visualization shows us that we can clearly distinguish two slow methods: MCFS, and VI, and two fast ones in the form of MI, and BORUTA. As we can see in a more detailed Figure 9, our previous assumption that all feature selection methods lead to much longer computations is invalid. The longest execution of MI-based feature selection took only one minute, whereas BORUTA ended within 4 minutes. The slow methods, however, are much longer and can last up to 32 minutes, which is far too long for efficient usage. We can also summarize that the order from fastest to slowest feature selection method is: MI, BORUTA, MCFS, VI (10's of seconds, 10's - 100's, 100's - 1000's, 100's - 1000's).

4.5 Performance analysis

In this section, we will delve deeply into the models' performance depending on various preprocessing approaches.

Advanced preprocessing vs Baseline

Similarly to the time analysis, the first step is to determine whether there are any significant differences between all preprocessing strategies. To check that we will use the visualization from Figure 10 for classification, and Figure 11 for regression tasks.

The first plot from Figure 10 shows us that for small binary classification tasks, there were not many differences between maximal scores, however, they occurred for both mean and median aggregations. From those two subplots we can see that in the case of wide boxplots, the X-mark describing the baseline model with minimal preprocessing, in most cases landed on the right side of the box, It means, that in general, advanced preprocessing strategies lead worsening of the models' performance. This trend did not happen every time, as for *breast-w* dataset baseline results were worse than the ones with preprocessing, and for the *credit-approval*, they were in the middle. Additionally, we can notice that aggregations by mean and median are similar to each other, thus in further plots, we will use the mean aggregation only.

The latter plot from Figure 11 suggests that for large regression tasks, the differences between various preprocessing strategies are even larger than for the smaller tasks, and tend to provide worse results than the baseline. The only task that achieved slightly better results was described by the *pol* dataset, which was a highly dimensional problem. As all three metrics provide fairly similar results, we will use only RMSE in further visualizations.

FS impact on performance

Similarly to the duration, the performance mostly depends on the fact that any feature selection method is used in the preprocessing stage. The visualization from Figure 12 can lead us to the same conclusions as the ones derived from Figure 10, and Figure 11. Moreover, we can clearly see that the improvement in the case of *breast-w*, and *pol* datasets originates in proper feature selection strategy. In other cases, however, obtained results did not improve, or were even worse than the baseline.

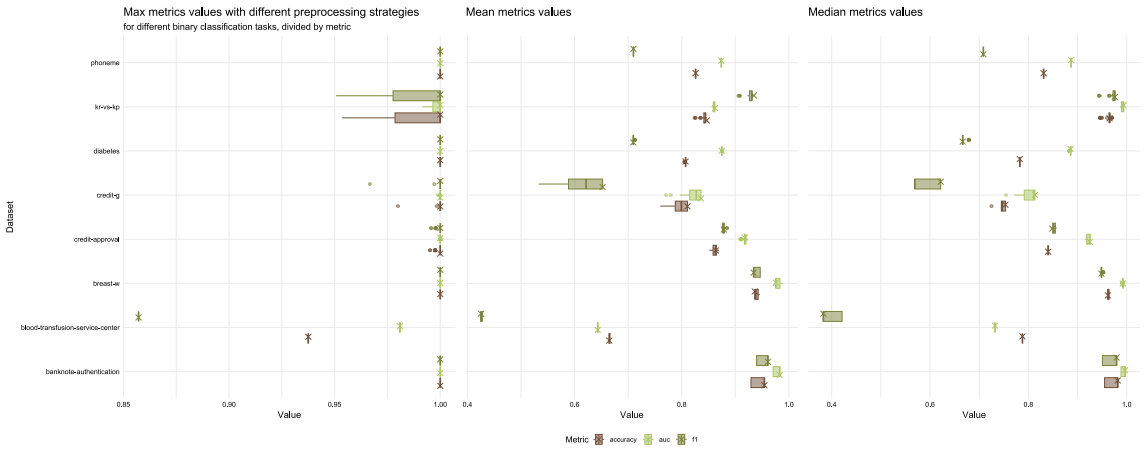


Figure 10: Binary classification results described by accuracy, auc, and f1 metrics aggregated with maximum, mean, and median values, for different datasets. The X-marks show the results obtained by the baseline model with minimal preprocessing. The results show us that in general advanced preprocessing strategies tend to worsen the results, however in some cases, like *breast-w* task, they can enhance models' performance significantly.

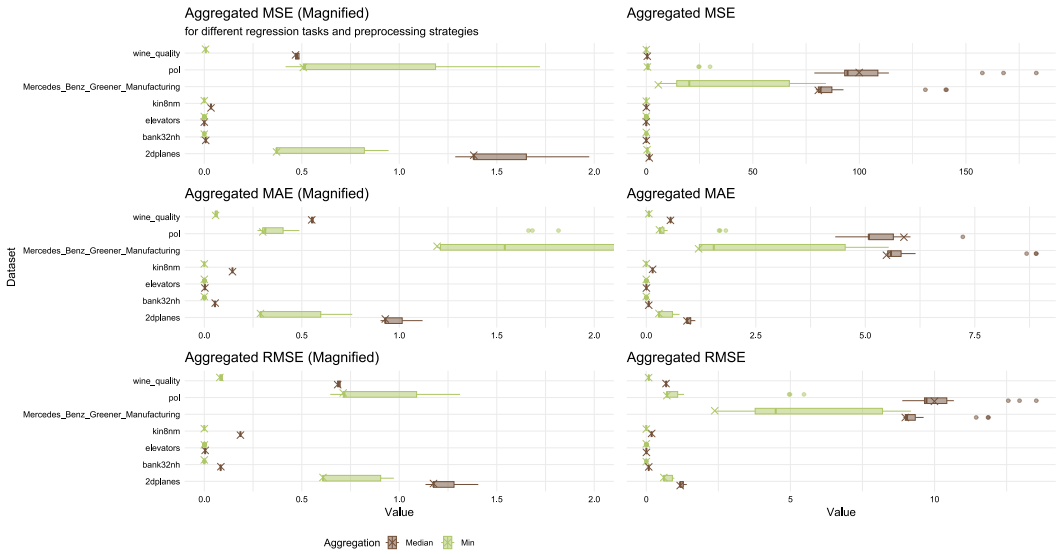


Figure 11: Regression results described by MSE, MAE, and RMSE metrics aggregated with minimum and median values, for different datasets. The X-marks show the results obtained by the baseline model with minimal preprocessing. The results show us that in general advanced preprocessing strategies tend to worsen the results, however in some cases, like *pol* task, they can enhance models' performance significantly.

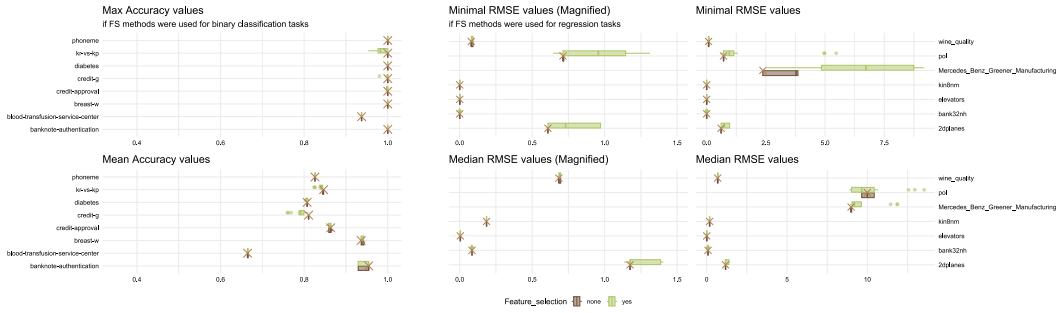


Figure 12: Binary classification and regression results whether feature selection methods were used. The X-marks show the results obtained by the baseline model with minimal preprocessing. The plots show us that in most cases the incorporation of feature selection did not improve, or even worsen the results in comparison to the baseline model.

Removal impact on performance

From Figure 13, even at first glance, we can notice that the results are more stable than when feature selection methods were used. Quite interestingly, different removal strategies also tend to obtain similar results, although there are some exceptions. For both binary classification and regression tasks, we can witness that some datasets benefit from more advanced removal settings (*credit-approval*, *breast-w*, *pol*), whereas others suffer because of it (*kr-vs-kp*, *banknote-authentication*, *phoneme*, *Mercedes_Benz_Greener_Manufacturing*). In general, however, those results do not change the outcomes drastically, but combined with the relatively low increase of computational time, they can be used to slightly enhance models' performance. The process of choosing the optimal setting for the step should be executed carefully, as there is no heuristic approach that visibly better results.

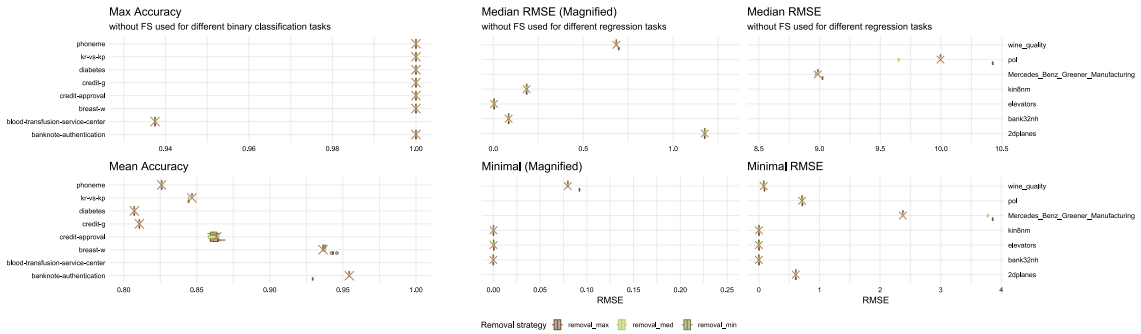


Figure 13: Binary classification and regression results when feature selection methods were not used, divided by different removal strategies. The X-marks show the results obtained by the baseline model with minimal preprocessing. The plots show us that in most cases the incorporation of more advanced removal options did not affect the outcomes significantly. It is possible however to obtain results better than the baseline if we choose the proper settings.

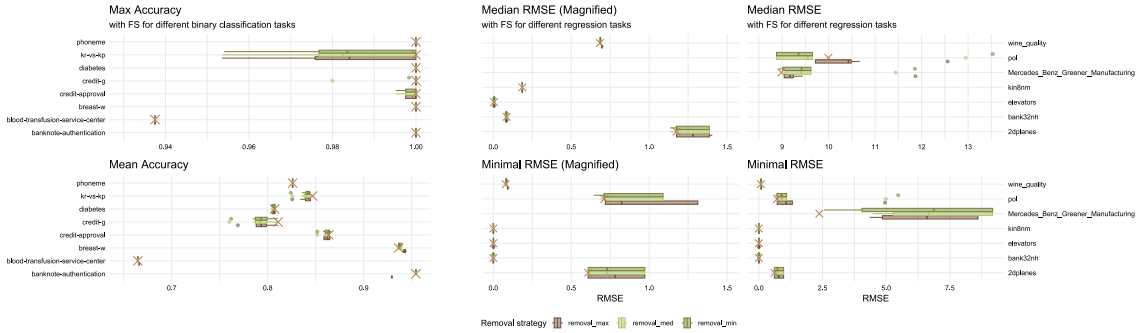


Figure 14: Binary classification and regression results when feature selection methods were used, divided by different removal strategies. The X-marks show the results obtained by the baseline model with minimal preprocessing. The plots show us that various feature selection methods impact the variability of the results for particular tasks, and the removal strategy can enhance, or worsen those results only in a minor way. Feature selection typically leads to worsening the results, but in the case of *pol* dataset it is able to improve them significantly.

Removal and feature selection

The last part of the performance analysis presented in Figure 14 combines both steps to analyze the impact that each of them has on the final results. If we compare them to the ones presented in Figure 13, we can clearly see that various selection strategies lead to higher variability of achieved outcomes, which in most cases leads to negative contributions. The best results, however, are in most cases unchanged, despite two regression tasks: *pol*, and *Mercedes_Benz_Greener_Manufacturing*. If we compare those results with the previous ones, we can clearly see that for the first one, feature selection leads to a significant enhancement of median outcomes, whereas the latter has only a negative impact. Those observations further suggest that feature selection methods should not be used with tree-based models, however in rare cases, with complex datasets, they might lead to better outcomes.

4.6 Conclusions

The provided ablation study proves that:

1. Trees, provide extremely efficient baseline models, which only use the most basic preprocessing strategies. However, in some cases of hard tasks, those results can be improved,
2. Well-suited removal strategies can improve models' performance slightly, but in most cases, they did not affect the results, thus they are safe to use without extensive study,
3. Additionally they last for a short time, and feature removal leads to dimensionality reduction, which accelerates the training time,
4. In general longer preprocessing leads to shorter training which levels off full-time,
5. Feature selection methods are the most unstable part of the pipeline in the case of both time complexity and final results,
6. They are the most time-consuming part of preprocessing, although there are fast methods like MI, and BORUTA, and slow ones: MCFS, and VI,
7. Feature selection algorithms tend to worsen the performance, but in rare cases might lead to enhancing the outcomes.

5 Limitations

Most of the limitations of this study come from the limited computational power described in Appendix A. With stronger machines, we would be able to add more datasets. Especially binary classification tasks lack the presence of bigger dataframes with worse data quality. As the datasets had a small number of missing fields, we were not able to check the imputation methods thoroughly. Moreover, the results suggest that considered tasks were unfortunately of too high quality, which would be also solved if more dataframes were tested.
Another field of improvement could be multiple-model training. In our study, the *forester* AutoML was run only once per dataset, and multiple calculations could provide more representative results. It's a minor issue, however, as the random search provided multiple models for each task, which was similar to multiple training.
The study could benefit from training the models with Bayesian optimization. Current results are representative of the general impact of preprocessing for tree-based models, and *foresters* results obtained with random search. Unfortunately, we cannot say the same for the optimization training with absolute certainty. Achieved outcomes might indicate that also for Bayesian optimization, however, it might happen that this training process would benefit from high-quality data more than random search.

6 Further works

Firstly, we could broaden the study, by adding more datasets to the analysis, analyze the impact of various feature selection methods, or check how particular models (like random forest, XGBoost, etc) are affected by each preprocessing strategy. We only have to analyze the results as raw outcomes are already calculated. Aside from adding new steps to this path, there are plenty of possibilities to follow up our research.
As a more technical aspect, we could create new module for the *forester* which conducts a single dataset, automated ablations study with the usage of random search. This could be beneficial for setting proper parameters for time consuming Bayesian optimization.
Furthermore, as tree-based models proved to cope incredibly well with high-dimensional data, we could conduct a study where we augment initial dataset with cross-products of two or more

features. Such study, would require adding another sub-module to custom preprocessing, where we augment the data.
Finally, the survival analysis task requires proper evaluation, as was done with binary classification and regression in the previous paper. We are also planning to introduce more engines and metrics for the task.

7 Conclusions

The main contribution of this paper is a wide ablation study of *foresters* custom preprocessing module and its implementation. It not only helps the user to understand the impact of various strategies but even more importantly finds out which preprocessing steps are important for the tree-based models. Our study proves that this family provides extremely efficient models when only the most basic preprocessing strategies are used. However, in some cases of hard tasks, those results can be improved. The safest option to do so is to incorporate proper removal strategies, as they rarely worsen outcomes quality, and the final changes are subtle. Additionally, they do not increase computing time a lot, and in some cases, they lead to dimensionality reduction, which accelerates the training time. Another approach is applying feature selection strategies, however, they can last for an incredibly long time and tend to worsen models' performance. Tree-based models prove to work extremely well without feature selection, however, in rare cases, it can improve the outcomes. Even though we recommend avoiding those methods when working with trees, as their incorporation requires proper investigation of various approaches and parameters in order to enhance the results.
Additionally, the paper presents other enhancements of the *forester* package. The first one is a module that supports the creation of models for survival analysis tasks. Its implementation is the beginning of further development and studies considering this topic, as there are no AutoML solutions tackling this task in the R community. Another improvement is the enhancement of the reporting module. The current implementation fixes the bugs present in previous versions, provides the results in a more approachable way, and adds thorough descriptions of how to read visualizations.

Acknowledgements. The 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 would not be possible without provided resources. I would also like to thank dr. Anna Cena, and mgr. Anna Kozak for their guidance, and insightful feedback during the project.

References

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

Bavarian, M., Jun, H., Tezak, N., Schulman, J., McLeavey, C., Tworek, J., and Chen, M. (2022). Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*.

Biecek, P. (2018). DALEX: Explainers for Complex Predictive Models in R. *Journal of Machine Learning Research*, 19(84):1–5.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

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.

Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794.

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.

Fararni, K. A., Nafis, F., Aghoutane, B., Yahyaouy, A., Riffi, J., and Sabri, A. (2021). Hybrid recommender system for tourism based on big data and AI: A conceptual framework. *Big Data Mining and Analytics*, 4(1):47–55.

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2022). Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. *Journal of Machine Learning Research*, 23(261):1–61.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, volume 28.

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*.

Hothorn, T. and Zeileis, A. (2015). partykit: A Modular Toolkit for Recursive Partytioning in R. *Journal of Machine Learning Research*, 16(118):3905–3909.

Ishwaran, H. and Kogalur, U. (2007). Random survival forests for r. *R News*, 7(2):25–31.

Ishwaran, H. and Kogalur, U. (2023). *Fast Unified Random Forests for Survival, Regression, and Classification (RF-SRC)*. R package version 3.2.2.

Ishwaran, H., Kogalur, U., Blackstone, E., and Lauer, M. (2008). Random survival forests. *Ann. Appl. Statist.*, 2(3):841–860.

Jorge, C. C., Antonio, O. A. J., Hugo, G. M. V., and Hugo, O. P. D. (2022). Machine Learning for Personal Credit Evaluation: A Systematic Review. *WSEAS TRANSACTIONS ON COMPUTER RESEARCH*, 10:62–73.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, volume 30.

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 *AutoML Conference 2023 (ABCD Track)*.

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.

Lang, M., Binder, M., Richter, J., Schratz, P., Pfisterer, F., Coors, S., Au, Q., Casalicchio, G., Kotthoff, L., and Bischl, B. (2019). mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, 4(44):1903.

LeDell, E., Gill, N., Aiello, S., Fu, A., Candel, A., Click, C., Kraljevic, T., Nykodym, T., Aboyoun, P., Kurka, M., and Malohlava, M. (2022). *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.38.0.1.

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.

Molnar, C., Casalicchio, G., and Bischl, B. (2020). Interpretable machine learning – a brief history, state-of-the-art and challenges. In *ECML PKDD 2020 Workshops*, pages 417–431.

Olson, R. S., Bartley, N., Urbanowicz, R. J., and Moore, J. H. (2016). Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 485–492.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, volume 31.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

Santos, L. and Ferreira, L. (2023). Atlantic—automated data preprocessing framework for supervised machine learning. *Software Impacts*, 17:100532.

Shimizu, H. and Nakayama, K. I. (2020). Artificial intelligence in oncology. *Cancer Science*, 111(5):1452–1460.

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.

Symeonidis, S., Effrosynidis, D., and Arampatzis, A. (2018). A comparative evaluation of pre-processing techniques and their interactions for twitter sentiment analysis. *Expert Systems with Applications*, 110:298–310.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2013). Openml: networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60.

Wirth, R. and Hipp, J. (2000). CRISP-DM: Towards a standard process model for data mining. *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*.

Wright, M. N. and 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.

A Computational resources

As mentioned in Section 4.2, our team was limited in computational power. The experiment was conducted on our 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, however as the *forester* is not yet implemented to work on the GPU, only the CPU was used.

B Datasets resources

The source code of dataset analysis that led to the creation of Table 1, and Table 2 is available in the GitHub repository `https://github.com/ModelOriented/forester/misc/ablation_study` as `ablation_study_datasets_info.Rmd` file. The notebook presents how we have used *foresters* `data_check()` function to obtain automatic descriptions of all considered tasks. The original xlsx files, describing datasets are also available `https://github.com/ModelOriented/forester/misc/ablation_study/data_info` as `datasets_classification.xlsx`, and `datasets_regression.xlsx` files.

C Preprocessing resources

Table 3 presents us the preprocessing strategies described in Subsection 4.2, in a much more compact way. The horizontal lines represent the segments described in the aforementioned section.
The code used for creation of preprocessed subsets is available in the GitHub repository `https://github.com/ModelOriented/forester/misc/ablation_study` as `ablation_study_preprocessing.Rmd` file. It is fully executable, as long as you also download `binary_CC18.RData`, `regression_bench.RData`, and `data_issues_summary.csv` files, from `https://github.com/ModelOriented/forester/misc/ablation_study` folder. This script creates a nested structure of directories and files that are singular steps of preparing the outcomes. Firstly it creates a separate directory for all the files, later it creates sub-directories for separate tasks, which inside hold RData files named 1-39 which are ids of respective preprocessing strategy. Additionally, these folders hold RData files with a list of execution times in seconds and a list of descriptive names for each checkpoint. When all 39 datasets are created, in the upper directory an aggregated file is created with the name of the task. Our GitHub repository contains those aggregated files only in the folder `https://github.com/ModelOriented/forester/misc/ablation_study/ablation_processed_results`.

Removal	Imputation	Feature Selection
min	median-other	none
med	median-other	none
max	median-other	none
min	median-other	none
min	median-frequency	none
min	knn	none
min	mice	none
min	median-other	VI
min	median-other	MCFS
min	median-other	MI
min	median-other	BORUTA
med	median-other	none
med	median-frequency	none
med	knn	none
med	mice	none
max	median-other	none
max	median-frequency	none
max	knn	none
max	mice	none
med	median-other	VI
med	median-other	MCFS
med	median-other	MI
med	median-other	BORUTA
max	median-other	VI
max	median-other	MCFS
max	median-other	MI
max	median-other	BORUTA
min	median-other	VI
min	knn	VI
min	median-other	BORUTA
min	knn	BORUTA
med	median-other	VI
med	knn	VI
med	median-other	BORUTA
med	knn	BORUTA
max	median-other	VI
max	knn	VI
max	median-other	BORUTA
max	knn	BORUTA

Table 3: Compact visualization of the preprocessing strategies described in the Subsection 4.2, used for ablation study.

D Training resources

The main notebook responsible for the models training is available in the GitHub repository `https://github.com/ModelOriented/forester/misc/ablation_study` as `ablation_study_training.Rmd`. To run it, you have to download the resulting files from `ablation_preprocessing_data` folder on google drive: `https://drive.google.com/drive/`

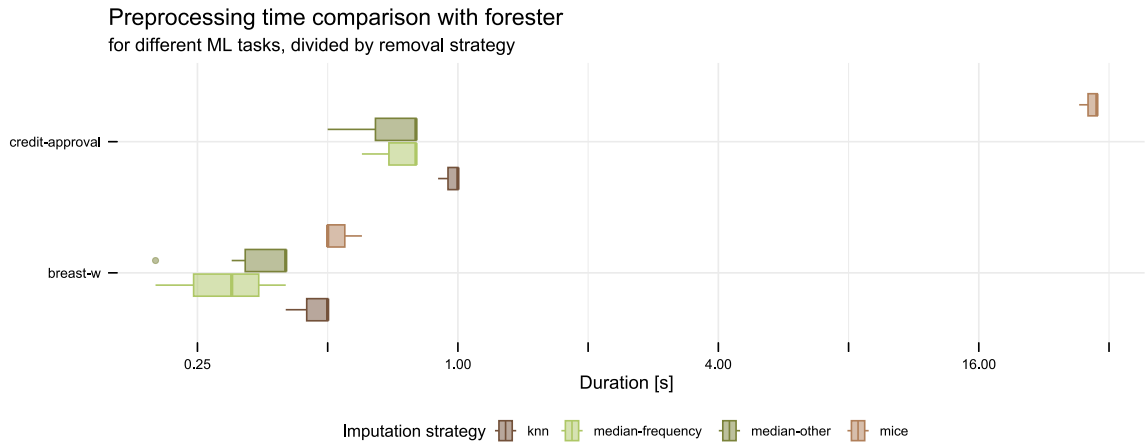


Figure 15: Duration of various imputation strategies for datasets where missing values appear.

`folders/1e8BIDJ4MjrSHoAKntYmUGsW2zifmKHxy?usp=sharing`. The code creates the same folder structure as in the case of data preprocessing, and the resulting files contain merged *forester* outcomes. These files are not available on the GitHub repository, because of their size reaching up to 3GB.
The second notebook concerning model training is `ablation_study_results_preparation.Rmd`. This aggregates the results of model training into a more approachable and compact way. The resulting files inside of `ablation_processed_results` directory are: `preprocessing_duration.RData`, `training_duration.RData`, `training_summary.RData`, and `extended_training_summary_table.RData`. The last one is of course the most important, as it combines information from all three aforementioned datasets.

E Results resources

The main notebook responsible for the ablation study results is available in the GitHub repository `https://github.com/ModelOriented/forester/misc/ablation_study` as `ablation_study_results_analysis.Rmd`, but the visualizations present in the paper were created with `ablation_study_paper_plots.Rmd`. To run them, you have to download the resulting files from `ablation_preprocessed_results` folder. The first notebook contains both initial visualizations, and conclusions, which are presented in Section 4.

F Imputation impact

The time analysis of imputation methods is extremely narrow, as we only have two datasets that contain missing fields, and their amounts are rather small (16, and 37). Even though from Figure 15 we can notice that the only method that indeed differs in terms of computational expenses is the mice algorithm, which for the credit-approval task lasted 32 times longer than other methods. According to Figure 16 we can say that as preprocessing times are fairly similar also for datasets without missing values, they don't affect whole preprocessing time a lot. Thus, we can ignore their impact in other analyses and focus only on feature selection and removal strategies.

G Used assets

In this section, we describe the packages used for both *forester*, and the experiments. The packages outside of the *forester* required for the experiments are listed in Table 4. An additional requirement for the *catboost* package is installed Java. The packages required by the *forester* as well as their versions used during the experiment are presented in Table 5.

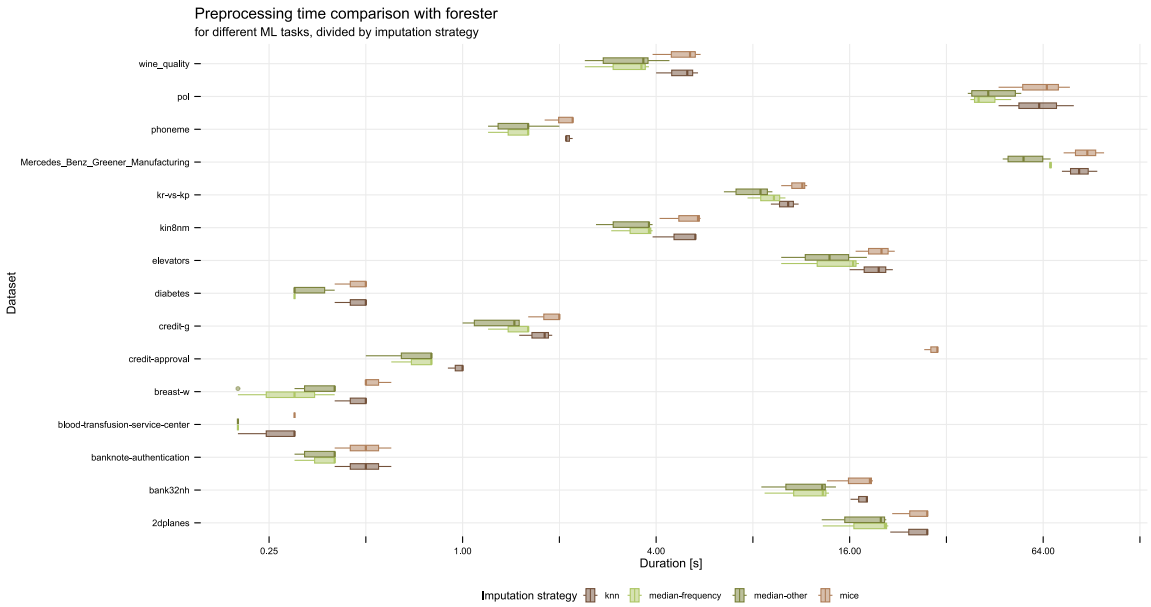


Figure 16: Duration of various imputation strategies for all datasets.

package	version	license
farff	1.1.1	BSD_2_clause
forester	1.3.0	GPL-3
ggplot2	3.4.0	MIT
patchwork	1.1.2	MIT
scales	1.2.1	MIT
OpenML	1.12	BSD_3_clause

Table 4: The packages and their versions under which the experiments were executed and supplemental materials were created.

H Package comparison

We have prepared a notebook showing the differences between the packages described in the related work section. The document includes a comparison of package installation, a description of available preprocessing, variable selection options, and model tuning. In addition, visualizations, methods of explainable machine learning, report preparation, and reference to available package documentation are described. We do not give a final assessment of the best package because it could be subjective, but we expose the reader to criticism. Notebook is available in the GitHub repository `https://github.com/ModelOriented/forester/blob/main/misc/experiments/framework_comparison.Rmd`.

package	version	license
arules	1.7-6	GPL-3
Boruta	7.0.0	GPL (>= 2)
catboost	1.1.1	Apache License (== 2.0)
crayon	1.5.2	MIT
DALEX	2.4.2	GPL
data.table	1.14.2	MPL-2.0
ggplot2	3.4.0	MIT
ggradar	0.2	GPL
ggrepel	0.9.3	GPL-3
knitr	1.40	GPL
lightgbm	3.3.2	MIT
mice	3.14.0	GPL-2 | GPL-3
mltools	0.3.5	MIT
ParBayesianOptimization	1.2.4	GPL-2
partykit	1.2-16	GPL-2 | GPL-3
parallel	4.1.2	Part of R 4.1.2
pROC	1.18.0	GPL (>= 3)
ranger	0.14.1	GPL-3
rcompanion	2.4.18	GPL-3
rmarkdown	2.16	GPL-3
rmcfs	1.3.5	GPL-3
splitTools	0.3.2	GPL (>= 2)
testthat	3.1.6	MIT
tibble	3.1.8	MIT
tinytex	0.43	MIT
varhandle	2.0.5	GPL (>= 2)
varrank	0.5	GPL-3
VIM	6.2.2	GPL (>= 2)
xgboost	1.6.0.1	Apache License (== 2.0)
stats	4.1.2	Part of R 4.1.2

Table 5: The *forester* package's dependencies and their versions used during the experiments.

I Report example

forester report

version 1.4.1

2023-09-30 16:47:07

The best models

This is the **binary_clf** task.

The best model evaluated on testing set is: **xgboost_bayes**, whereas for the validation set it is: **ranger_RS_1**.

The models inside the forester package are trained on the training set, the Bayesian Optimization is tuned according to the testing set, and the validation set is never seen during the training. The training set should not be used for evaluation as the models always perform the best for the data they have seen (overfitting). The least biased dataset is the validation set, however we can also use the testing set, ex. to check if the models overfit.

The names of the models were created by a pattern *Engine_TuningMethod_Id*, where:

- `Engine` - describes the engine used for the training (random_forest, xgboost, decision_tree, lightgbm, catboost),
- `TuningMethod` - describes how the model was tuned (`basic` for basic parameters, `RS` for random search, `bayes` for Bayesian optimization),
- `Id` - is used for separating the random search parameters sets.

More details about the dataset are present at the end of the report.

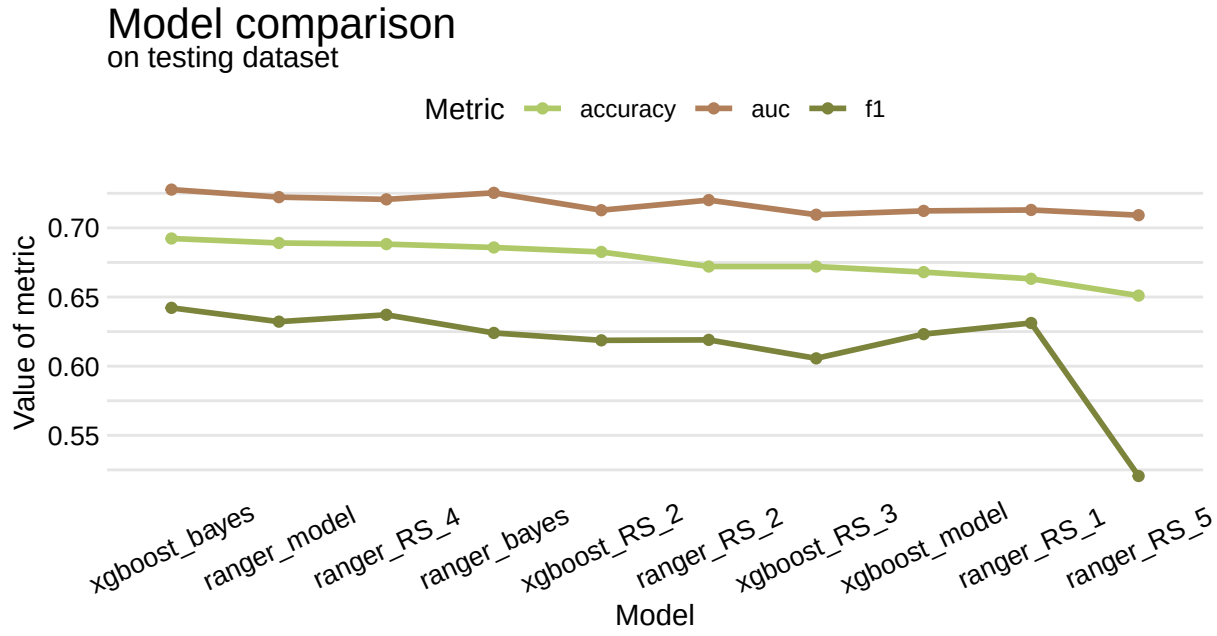
Best models for validation dataset

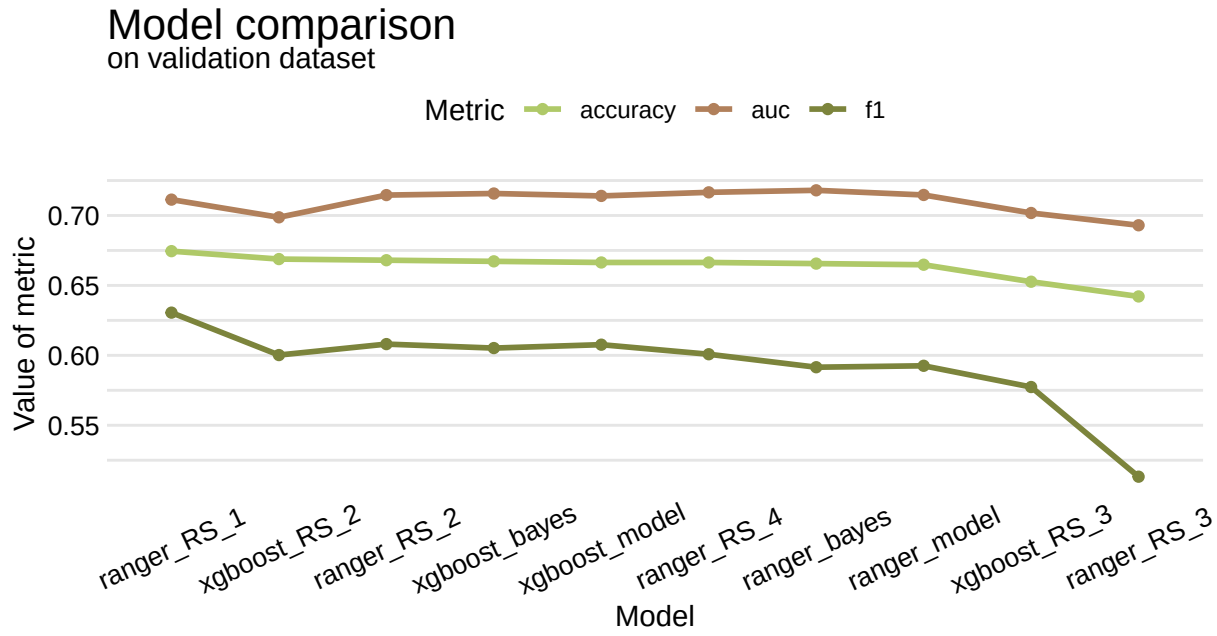
no.	name	accuracy	auc	f1
4	ranger_RS_1	0.6745	0.7113	0.6305
10	xgboost_RS_2	0.6688	0.6987	0.6002
5	ranger_RS_2	0.6680	0.7146	0.6080
20	xgboost_bayes	0.6672	0.7157	0.6052
2	xgboost_model	0.6664	0.7140	0.6076
7	ranger_RS_4	0.6664	0.7165	0.6008
19	ranger_bayes	0.6656	0.7180	0.5915
1	ranger_model	0.6648	0.7147	0.5925
11	xgboost_RS_3	0.6526	0.7018	0.5773
6	ranger_RS_3	0.6421	0.6930	0.5132

Model comparison

Metrics comparison

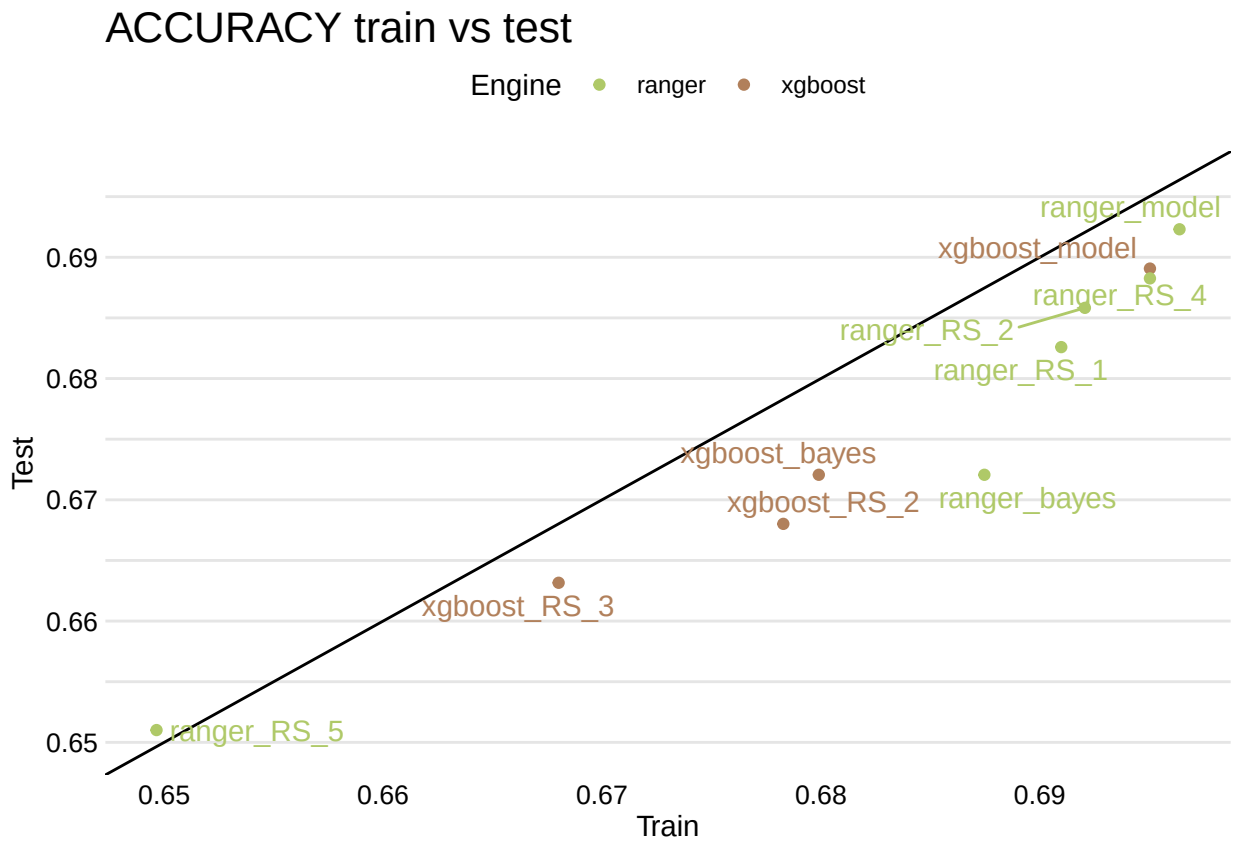
The comparison plot takes a closer look on top 10 performing models, and evaluates their performance in terms of three well-known metrics: accuracy, area under the curve (AUC), and F1 score. For each metric, the larger the value, the better the model is. As the ranked list compares the models only in terms of accuracy, we want to additionally evaluate the performance in terms of other metrics. In some cases it might happen that other model is better in terms of AUC, or F1, but slightly worse in accuracy, and we would like to choose the other model. The results are presented for both testing and validation dataset.





Train vs test plot

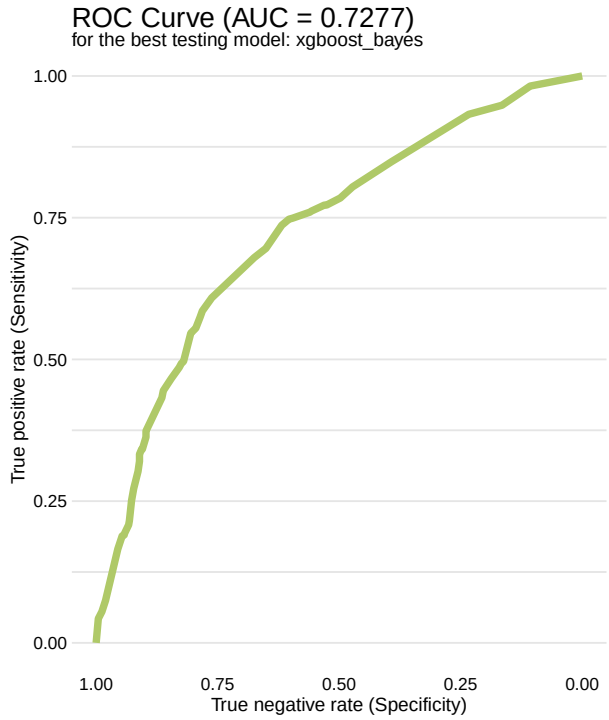
This scatter plot tackles the issue of the overfitting, and compares large amounts of models at once. On the x axis we provide the metrics value evaluated on the training dataset, whereas on the y axis we have the same for the testing dataset. Models performance is assessed in two ways. Firstly, we want the model to have as small value as possible on the testing dataset (so we want it to be lower than other models). Secondly, we want to choose the model which is close to the x = y line, because it means that the model is not overfitted, so it generalizes better. In most cases we want to chose the model that is less overfitted, even though it has worse performance.

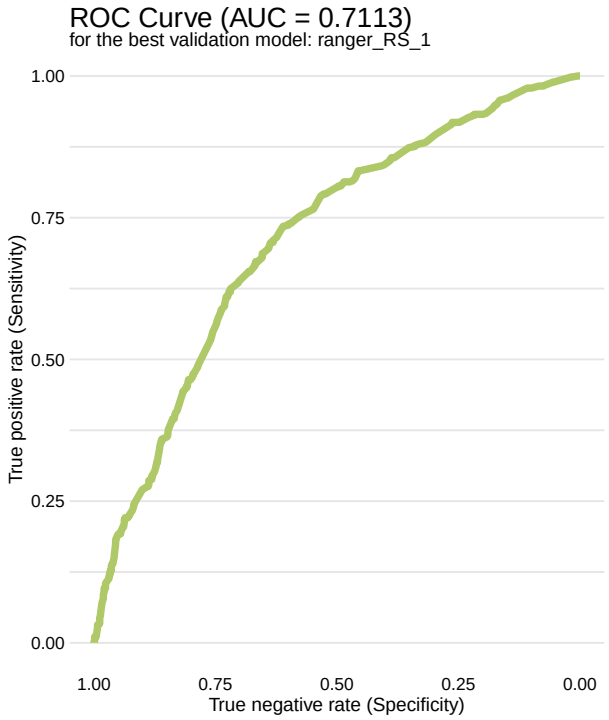


Plots for the best models

ROC

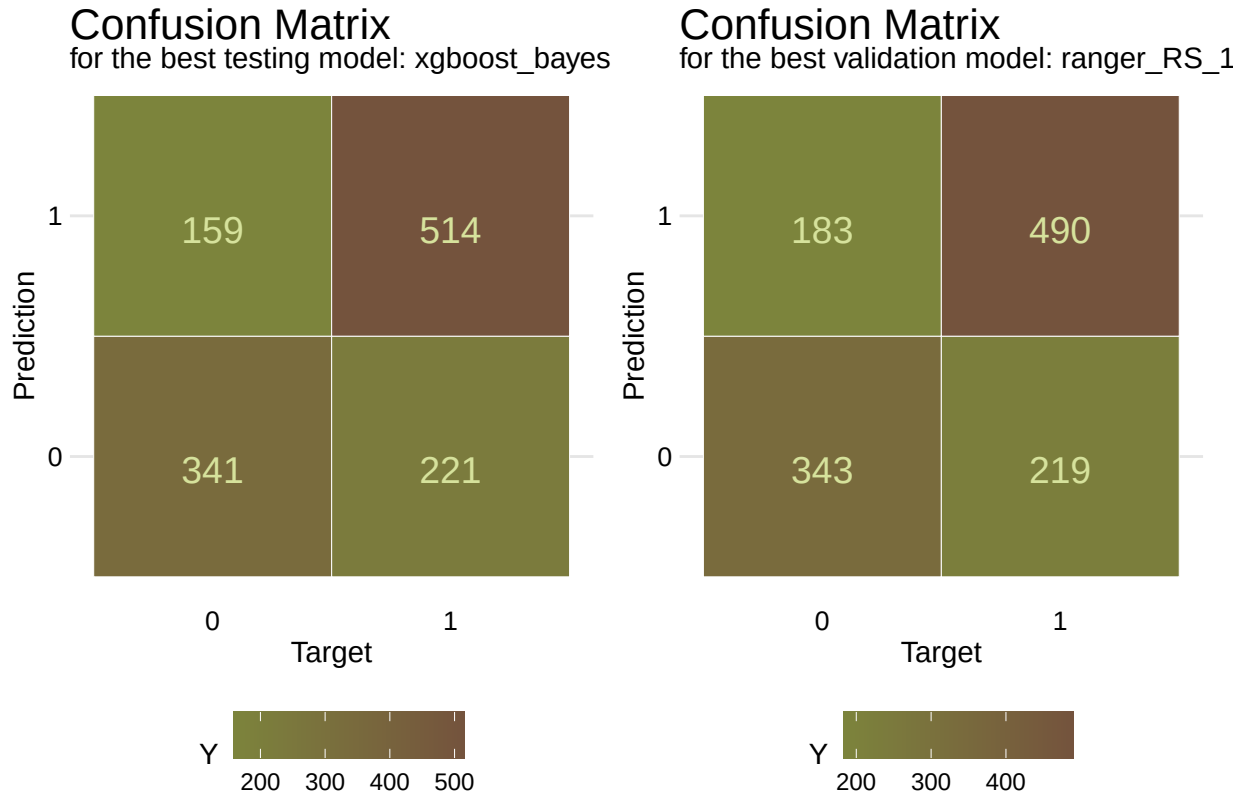
A ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: True Positive Rate (TPR/Sensitivity $= \frac{TP}{TP+FN}$), and True Negative Rate (TNR/Specificity $= \frac{TN}{TN+FP}$). A ROC curve plots TPR vs. TNR at different classification thresholds. Lowering the classification threshold classifies (probability that the prediction is classified as positive) more items as positive, thus increasing both False Positives (FP) and True Positives (TP). AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (1, 0) to (0, 1). The greater the value, the better the model distinguishes between the two classes.





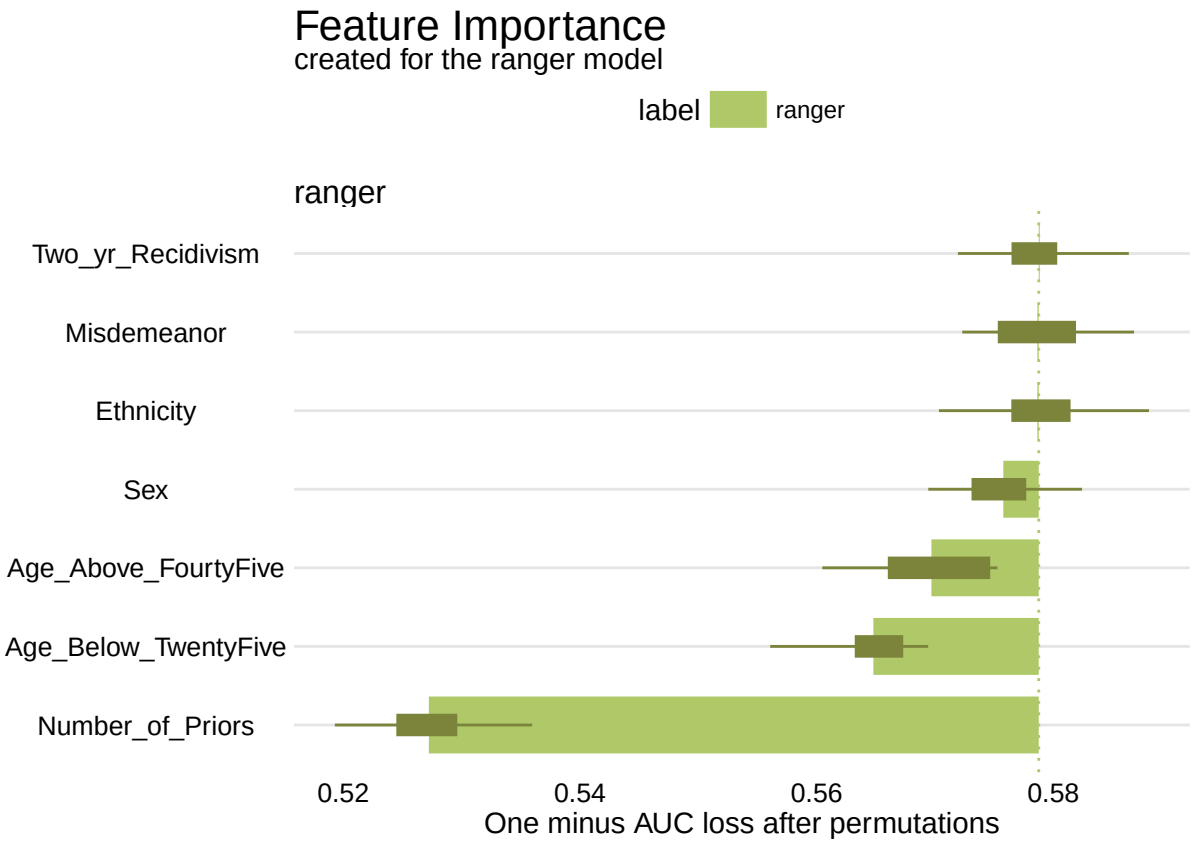
Confusion matrix

The confusion matrix is a simple way to to visualize which types of errors are made by the model. The plot below presents us the raw values of TP (x = 1, y = 1; True Positive), FP (x = 0, y = 1; False Positive), TN (x = 0, y = 0; True Negative), and FN (x = 1, y = 0; False Negative). Thanks to this visualization we can ex. see if our model has a tendency to predict mostly one class.



Feature Importance

The final visualization presents us with the feature importance plot which lets us understand what's happening inside the best model evaluated on validation set. Feature Importance (FI) shows us the most important variables for the model, and the bigger the absolute value, the more important a variable is. Large FI values for a feature indicate that if we permute the values for the column randomly, it changes the final outcomes drastically.



Details about data

CHECK DATA REPORT

The dataset has 6172 observations and 7 columns which names are:

Two_yr_Recidivism; Number_of_Priors; Age_Above_FourtyFive; Age_Below_TwentyFive; Misdemeanor; Ethnicity; Sex;

With the target described by a column Two_yr_Recidivism.

No static columns.

No duplicate columns.

No target values are missing.

No predictor values are missing.

No issues with dimensionality.

No strongly correlated, by Spearman rank, pairs of numerical values.

No strongly correlated, by Crammer's V rank, pairs of categorical values.

There are more than 50 possible outliers in the data set, so we are not printing them. They are returned in the output as a vector.

Dataset is balanced.

Columns names suggest that none of them are IDs.

Columns data suggest that none of them are IDs.

CHECK DATA REPORT END