ASML: An R Package for Algorithm Selection with Machine Learning

by Ignacio Gómez-Casares, Beatriz Pateiro-López, Brais González-Rodríguez, and Julio González-Díaz

Abstract For extensively studied computational problems, it is commonly acknowledged that different instances may require different algorithms for optimal performance. The R package ASML focuses on the task of efficiently selecting from a given portfolio of algorithms, the most suitable one for each specific problem instance, based on significant instance features. The package allows for the use of the machine learning tools available in the R package caret and additionally offers visualization tools and summaries of results that make it easier to interpret how algorithm selection techniques perform, helping users better understand and assess their behavior and performance improvements.

1 Introduction

Selecting from a set of algorithms the most appropriate one for solving a given problem instance (understood as an individual problem case with its own specific characteristics) is a common issue that comes up in many different situations, such as in combinatorial search problems (Kotthoff, 2016; Drake et al., 2020), planning and scheduling problems (Speck et al., 2021; Messelis and De Causmaecker, 2014), or in machine learning (ML), where the multitude of available techniques often makes it challenging to determine the best approach for a particular dataset (Vanschoren, 2019). For an extensive survey on automated algorithm selection and application areas, we refer to Kerschke et al. (2019).

Figure 1 presents a general scheme, adapted from Figure 1 in Kerschke et al. (2019), illustrating the use of ML for algorithm selection. A set of problem instances is given, each described by associated features, together with a portfolio of algorithms that have been evaluated on all instances. The instance features and performance results are then fed into a ML framework, which is trained to produce a selector capable of predicting the best-performing algorithm for an unseen instance. Note that we are restricting attention to *offline* algorithm selection, in which the selector is constructed using a training set of instances and then applied to new problem instances.

Algorithm selection tools also demonstrate significant potential in the field of optimization, enhancing performance at solving problems where multiple solving strategies are often available. For example, a key factor in the efficiency of state-of-the-art global solvers in mixed integer linear programming and also in nonlinear optimization is the design of branch-and-bound algorithms and, in particular, of their branching rules. There isn't a single branching rule that outperforms all others on every problem instance. Instead, different branching rules exhibit optimal performance on different types of problem instances. Developing methods for the automatic selection of branching rules based on instance features has proven to be an effective strategy toward solving optimization problems more efficiently (Lodi and Zarpellon, 2017; Bengio et al., 2021; Ghaddar et al., 2023).

In algorithm selection, not only do the problem domain to which it applies and the algorithms for addressing problem instances play a crucial role, but also the metrics used to assess algorithm effectiveness —referred to in this work as Key Performance Indicators (KPIs). KPIs are used in different fields to assess and measure the performance of specific objectives or goals. In a business context, these indicators are quantifiable metrics that provide valuable insights into how well an individual, team, or entire organization is progressing towards achieving its defined targets. In the context of algorithms, KPIs serve as quantifiable measures used to evaluate the effectiveness and efficiency of algorithmic processes. For instance, in the realm of computer science and data analysis, KPIs can include measures like execution time, accuracy, and scalability. Monitoring these KPIs allows for a

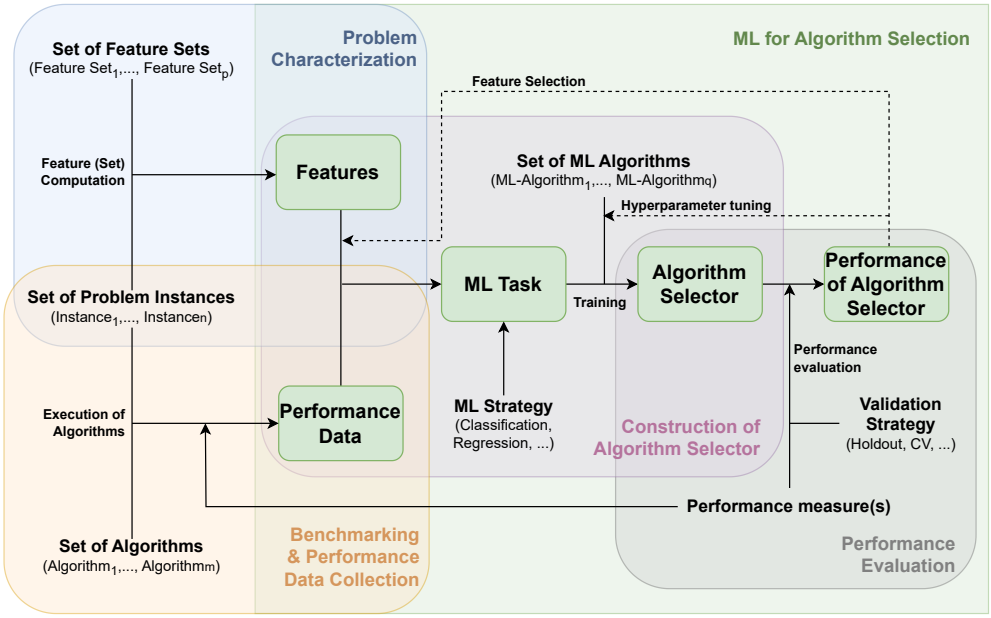

Figure 1: Schematic overview of the interplay between problem instance features (top left), algorithm performance data (bottom left), selector construction (center), and the assessment of selector performance (bottom right). Adapted from Kerschke et al. (2019).

comprehensive assessment of algorithmic performance, aiding in the selection of the most appropriate algorithm for a given instance and facilitating continuous improvement in algorithmic design and implementation.

Additionally, in many applications, normalizing the KPI to a standardized range like $[0, 1]$ provides a more meaningful basis for comparison. The KPI obtained through this process, which we will refer to as instance-normalized KPI, reflects the performance of each algorithm relative to the best-performing one for each specific instance. For example, if we have multiple algorithms and we are measuring execution time that can vary across instances, normalizing the execution time for each instance relative to the fastest algorithm within that same instance allows for a fairer evaluation. This is particularly important when the values of execution time might not directly reflect the relative performance of the algorithms due to wide variations in the scale of the measurements. Thus, normalizing puts all algorithms on an equal footing, allowing a clearer assessment of their relative efficiency.

Following the general framework illustrated in Figure 1, the R package **ASML** (González-Rodríguez et al., 2025b) provides a wrapper for ML methods to select from a portfolio of algorithms based on the value of a given KPI. It uses a set of features in a training set to learn a regression model for the instance-normalized KPI value for each algorithm. Then, the instance-normalized KPI is predicted for unseen test instances, and the algorithm with the best predicted value is chosen. As learning techniques for algorithm selection, the user can invoke any regression method from the **caret** package (Kuhn and Max, 2008) or use a custom function defined by the user. This makes our package flexible, as it automatically supports new methods when they are added to **caret**. Although initially designed for selecting branching rules in nonlinear optimization problems, its versatility allows the package to effectively address algorithm selection challenges across a wide range of domains. It can be applied to a broad spectrum of disciplines whenever there is a diverse set of instances within a specific problem domain, a suite of algorithms with varying behaviors across instances, clearly defined metrics for evaluating the performance of the available algorithms, and known features or characteristics of the instances that can be computed and are ideally correlated with algorithm performance. The visualization tools implemented in the package allow for an effective evaluation of the performance of the algorithm selection techniques. A

key distinguishing element of **ASML** is its learning-phase approach, which uses instance-normalized KPI values and trains a separate regression model for each algorithm to predict its normalized KPI on unseen instances.

2 Background

The algorithm selection problem was first outlined in the seminal work by Rice (1976) . In simple terms, for a given set of problem instances (problem space) and a set of algorithms (algorithm space), the goal is to determine a selection model that maps each problem instance to the most suitable algorithm for it. By *most suitable*, we mean the best according to a specific metric that associates each combination of instance and algorithm with its respective performance. Formally, let $\mathcal{P}$ denote the problem space or set of problem instances. The algorithm space or set of algorithms is denoted by $\mathcal{A}$. The metric $p : \mathcal{P} \times \mathcal{A} \to \mathbb{R}^n$ measures the performance $p(x, A)$ of any algorithm $A \in \mathcal{A}$ on instance $x \in \mathcal{P}$. The goal is to construct a selector $S : \mathcal{P} \to \mathcal{A}$ that maps any problem instance $x \in \mathcal{P}$ to an algorithm $S(x) = A \in \mathcal{A}$ in such a way that its performance is optimal.

As discussed in the Introduction, many algorithm selection methods in the literature use ML tools to model the relationship between problem instances and algorithm performance, using features derived from these instances. The pivotal step in this process is defining appropriate features that can be readily computed and are likely to impact algorithm performance. That is, given $x \in \mathcal{P}$, we make use of informative features $f(x) = (f_1(x), \ldots, f_k(x)) \in \mathbb{R}^k$. In this framework, the selector S maps the simpler feature space $\mathbb{R}^k$ into the algorithm space $\mathcal{A}$. A scheme of the algorithm selection problem, as described in Rice (1976), is shown in Figure 2.

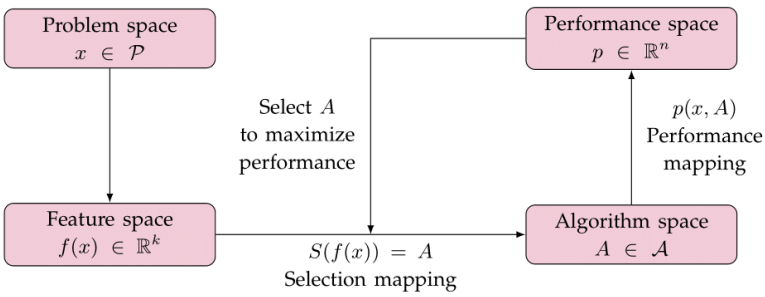

Figure 2: Scheme of the algorithm selection problem by Rice (1976).

For the practical derivation of the selection model S, we use training data consisting of features $f(x)$ and performances $p(x, A)$, where $x \in \mathcal{P}' \subset \mathcal{P}$ and $A \in \mathcal{A}$. The task is to learn the selector S based on the training data. The model allows us to forecast the performance on unobserved instance problems based on their features and subsequently select the algorithm with the highest predicted performance. A comprehensive discussion of various aspects of algorithm selection techniques can be found in Kotthoff (2016) and Pulatov et al. (2022).

3 Algorithm selection tools in R

The task of algorithm selection has seen significant advancements in recent years, with R packages facilitating this process. Here we present some of the existing tools that offer a range of functionalities, including flexible model-building frameworks, automated workflows, and standardized scenario formats, providing valuable resources for both researchers and end-users in algorithm selection.

The **llama** package (Kotthoff et al., 2021) provides a flexible implementation within R for evaluating algorithm portfolios. It simplifies the task of building predictive models to solve algorithm selection scenarios, allowing users to apply ML models effectively. In **llama**, ML algorithms are defined using the **mlr** package (Bischl et al., 2016b), offering a structured approach to model selection. On the other hand, the Algorithm Selection Library (ASlib) (Bischl et al., 2016a) proposes a standardized format for representing algorithm selection scenarios and introduces a repository that hosts an expanding collection of datasets from the literature. It serves as a benchmark for evaluating algorithm selection techniques under consistent conditions. It is accessible to R users through the **aslib** package. This integration simplifies the process for those working within the R environment. Furthermore, **aslib** interfaces with the **llama** package, facilitating the analysis of algorithm selection techniques within the benchmark scenarios it provides.

Our **ASML** package offers an approach to algorithm selection based on the powerful and flexible **caret** framework. By using **caret**'s ability to work with many different ML models, along with its model tuning and validation tools, **ASML** makes the selection process easy and effective, especially for users already familiar with **caret**. Thus, while **ASML** shares some conceptual similarities with **llama**, it distinguishes itself through its interface to the ML models in **caret** instead of **mlr**, which is currently considered retired by the mlr-org team, potentially leading to compatibility issues with certain learners, and has been succeeded by the next-generation **mlr3** (Lang et al., 2019). In addition, **ASML** automates the normalization of KPIs based on the best-performing algorithm for each instance, addressing the challenges that arise when performance metrics vary significantly across instances. **ASML** further provides new visualization tools that can be useful for understanding the results of the learning process. A comparative overview of the main features and differences between these packages can be seen in Table 1.

Table 1: Comparative overview of ASML and llama for algorithm selection.

Aspect	ASML	llama
Input data	features KPIs split by families supported	features KPIs feature costs supported
Normalized KPIs	✓	✗
ML backend	caret	mlr
Hyperparameter tuning	ASML::AStrain() supports arguments passed to caret (trainControl(), tuneGrid)	llama::cvFolds llama::tuneModel
Parallelization	✓ with snow	✓ with parallelMap
Results summary	Per algorithm Best overall and per instance ML-selected	Virtual best and single best per instance Aggregated scores (PAR, count, successes)
Visualization	Boxplots (per algorithm and ML-selected) Ranking plots Barplots (best vs ML-selected)	Scatter plots comparing two algorithm selectors
Model interpretability tools	✓ with DALEX	✗
ASlib integration	basic support	extended support
Latest release	CRAN 1.1.0 (2025)	CRAN 0.10.1 (2021)

There are also automated approaches that streamline the process of selecting and optimizing ML models within the R environment. Tools like **h2o** provide robust functionalities specifically designed for R users, facilitating an end-to-end ML workflow. These frameworks automate various tasks, including algorithm selection, hyperparameter optimization, and feature engineering, thereby simplifying the process for users of all skill levels. By integrating these automated solutions into R, users can efficiently explore a wide range of models

and tuning options without needing extensive domain knowledge or manual intervention. This automation not only accelerates the model development process but also improves the overall performance of ML projects by allowing a systematic evaluation of different approaches and configurations. However, while **h2o** excels at automating the selection of ML models and hyperparameter tuning, it does not perform algorithm selection based on instance-specific features, which is the primary focus of our approach. Instead, it evaluates multiple algorithms in parallel and selects the best-performing one based on predetermined metrics.

4 Using the ASML package

Here, we illustrate the usage of the **ASML** package with an example within the context of algorithm selection for spatial branching in polynomial optimization, aligning with the problem discussed in Ghaddar et al. (2023) and further explored in González-Rodríguez et al. (2025a). Table 2 provides an overview of the problem and a summary of the components that we will discuss in detail below.

Table 2: Summary of the branching rule selection problem.

Algorithms	`max, sum, dual, range, eig-VI, eig-CMI`
KPI	`pace`
Number of instances	407
Number of instances per library	180 (DS), 164 (MINLPLib), 63 (QPLIB)
Number of features	33

A well-known approach for finding global optima in polynomial optimization problems is based on the use of the Reformation Linearization Technique (RLT) (Sherali and Tuncbilek, 1992). Without delving into intricate details, RLT operates by creating a linear relaxation of the original polynomial problem, which is then integrated into a branch-and-bound framework. The branching process involves assigning a score to each variable, based on the violations of the RLT identities it participates in, after solving the corresponding relaxation at each node. Subsequently, the variable with the highest score is selected for branching. The computation of these scores is a critical aspect and allows for various approaches, leading to distinct branching rules that constitute our algorithm selection portfolio. Specifically, in our example, we will examine six distinct branching rules (referred to interchangeably as branching rules or algorithms), labeled as `max`, `sum`, `dual`, `range`, `eig-VI`, and `eig-CMI` rules. For the definitions and a comprehensive understanding of the rationale behind these rules, refer to Ghaddar et al. (2023).

Measuring the performance of different algorithms in the context of optimization is crucial for evaluating their effectiveness and efficiency. Two common metrics for this evaluation are running time and optimality gap, measured as a function of the lower and upper bounds for the objective function value at the end of the algorithm (a small optimality gap indicates that the algorithm is producing solutions close to the optimal). Both metrics are important and are often considered together to evaluate algorithm performance. For instance, it is meaningful to consider the time required to reduce the optimality gap by one unit as KPI. In our example, and to ensure it is well-defined, we make use of a slightly different metric, which we refer to as pace, defined as the time required to increase the lower bound by one unit. For the pace, a smaller value is preferred, as it indicates better performance.

As depicted in Figure 2, a crucial aspect of the methodology involves selecting input variables (features) that facilitate the prediction of the KPI for each branching rule. We consider 33 features representing global information of the polynomial optimization problems, such as relevant characteristics of variables, constraints, monomials, coefficients, or other attributes. A detailed description of the considered features can be found in Table 3. Although we won't delve into these aspects, determining appropriate features is often complex, and using feature-selection methods can be beneficial for choosing the most relevant ones.

Table 3: Features from the branching dataset.

Index	Description
3	Number of variables
4	Number of constraints
5	Degree
6	Number of monomials
7	Density
8	Density of VIG
9	Modularity of VIG
10	Treewidth of VIG
11	Density of CMIG
12	Modularity of CMIG
13	Treewidth of CMIG
14	Pct. of variables not present in any monomial with degree greater than one
15	Pct. of variables not present in any monomial with degree greater than two
16	Number of variables divided by number of constrains
17	Number of variables divided by degree
18	Pct. of equality constraints
19	Pct. of linear constraints
20	Pct. of quadratic constraints
21	Number of monomials divided by number of constrains
22	Number of RLT variables divided by number of constrains
23	Pct. of linear monomials
24	Pct. of quadratic monomials
25	Pct. of linear RLT variables
26	Pct. of quadratic RLT variables
27	Variance of the ranges of the variables
28	Variance of the coefficients
29	Variance of the density of the variables
30	Variance of the no. of appearances of each variable
31	Average of the ranges of the variables
32	Average of the coefficients
33	Average pct. of monomials in each constraint and in the objective function
34	Average of the no. of appearances of each variable
35	Median of the ranges of the variables

Note:
Index refers to columns of branching$x.

To assess the performance of the algorithm selection methods in this context, we have a diverse set of 407 instances from different optimization problems, taken from three well-known benchmarks (Bussieck et al., 2003; Dalkiran and Sherali, 2016; Furini et al., 2018), corresponding respectively to the MINLPLib, DS, and QPLIB libraries. Details are given in Table 2. The data for this analysis is contained within the `branching` dataset included in the package. We begin by defining two data frames. The `features` data frame includes two initial columns that provide the instance names and the corresponding family (library in our example) for each instance. The remaining columns consist of the features listed in Table 3.

We also define the `KPI` data frame, which is derived from `branching$y`. This data frame contains the pace values for each of the six branching rules considered in this study (specified by the labels in the `lab_rules` vector). These data frames will serve as the input for our subsequent analyses.

```
set.seed(1234)
library(ASML)
data(branching)
features <- branching$x
KPI <- branching$y
lab_rules <- c("max", "sum", "dual", "range", "eig-VI", "eig-CMI")
```

4.1 Pre-Processing the data

As with any analysis, the first step involves preprocessing the data. This includes using the function `partition_and_normalize`, which not only divides the dataset into training and test sets but also normalizes the KPI relative to the best result for each instance. The argument `better_smaller` specifies whether a lower KPI value is preferred (such as in our case where the KPI represents pace, with smaller values indicating better performance) or if a higher value is desired, when larger KPI values are considered more advantageous.

```
data <- partition_and_normalize(features, KPI, family_column = 1, split_by_family = TRUE,
    better_smaller = TRUE)
names(data)
```

```
#> [1] "x.train"          "y.train"          "y.train.original" "x.test"
#> [5] "y.test"           "y.test.original"  "families.train"   "families.test"
#> [9] "better_smaller"
```

When using the function `partition_and_normalize` the resulting object is of class `as_data` and contains several key components essential for our study. Specifically, the object includes `x.train` and `x.test`, representing the feature sets for the training and test datasets, respectively. Additionally, it contains `y.train` and `y.test`, with the instance-normalized KPI corresponding to each dataset, along with their original counterparts, `y.train.original` and `y.test.original`. This structure allows us to retain the original KPI values while working with the instance-normalized data. Furthermore, when the parameter `split_by_family` is set to `TRUE`, as in the example, the object also includes `families.train` and `families.test`, indicating the family affiliation for each observation within the training and test sets. Figure 3 illustrates how the split preserves the proportions of instances for each library.

As a tool for visualizing the performance of the considered algorithms, the `boxplots` function operates on objects of class `as_data` and generates boxplots for the instance-normalized KPI. This visualization facilitates the comparison of performance differences across instances. The function can be applied to both training and test observations and can also group the results by family. Additionally, it accepts common arguments typically used in R functions. Figure 4 shows the instance-normalized KPI of the instances in the train set. What becomes evident from the boxplots is that there is no branching rule that outperforms the others

Figure 3: Train/Test partition preserving the percentage of instances for each library.

across all instances, and making a wrong choice of criteria in certain problems can lead to very poor performance.

```
boxplots(data, test = FALSE, by_families = FALSE, labels = lab_rules)
```

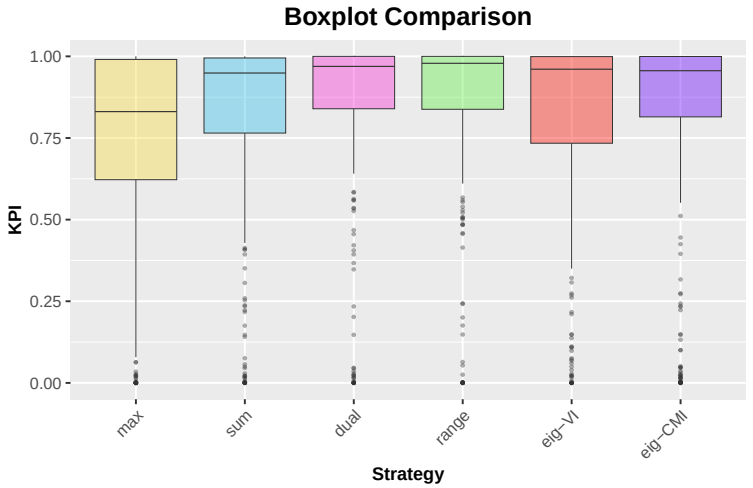

Figure 4: Boxplots of instance-normalized KPI for each algorithm across instances in the train set.

The `ranking` function, specifically designed for the **ASML** package, is also valuable for visualizing the differing behaviors of the algorithms under investigation, depending on the analyzed instances. After ranking the algorithms for each instance, based on the instance-normalized KPI, the function generates a bar chart for each algorithm, indicating the percentage of times it occupies each ranking position. The numbers displayed within the bars represent the mean value of the instance-normalized KPI for the problems associated with that specific ranking position. Again, the representation can be made both for the training and test sets, as well as by family. In Figure 5, we present the chart corresponding to the training sample and categorized by family. In particular, it is observed that certain rules, when not the best choice for a given instance, can perform quite poorly in terms of instance-normalized KPI (see, for example, the results on the MINLPLib library). This highlights the importance of not only selecting the best algorithm for each instance but also ensuring that the chosen algorithm does not perform too poorly when it isn't optimal. In some cases, even if an algorithm isn't the best-performing option, it may still provide reasonably good results, whereas a wrong choice can result in significantly worse outcomes.

```
ranking(data, test = FALSE, by_families = TRUE, labels = lab_rules)
```

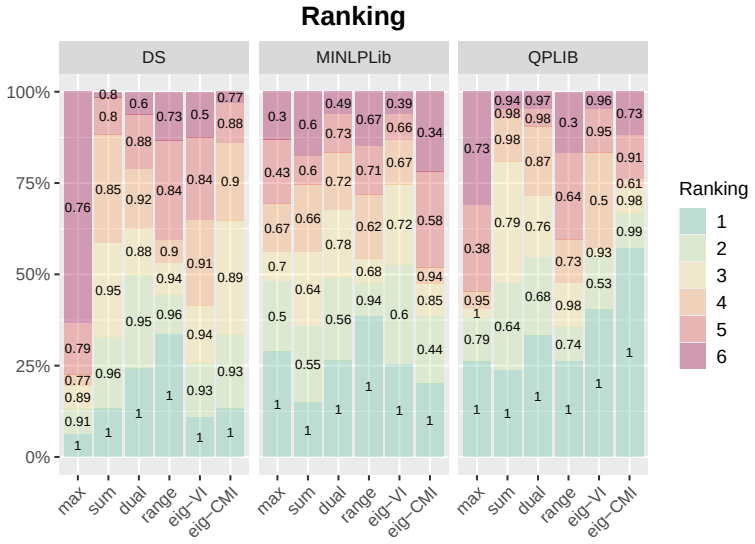

Figure 5: Ranking of algorithms based on the instance-normalized KPI for the training sample, categorized by family. The bars represent the percentage of times each algorithm appeared in different ranking positions, with the numbers indicating the mean value of the KPI.

Additionally, functions from the **caret** package can be applied if further operations on the predictors are needed. Here we show an example where the Yeo-Johnson transformation is applied to the training set, and the same transformation is subsequently applied to the test set to ensure consistency across both datasets. The flexibility of **caret** also allows for the inclusion of advanced techniques, such as feature selection and dimensionality reduction, to improve the quality of the algorithm selection process.

```
preProcValues <- caret::preProcess(data$x.train, method = "YeoJohnson")
data$x.train <- predict(preProcValues, data$x.train)
data$x.test <- predict(preProcValues, data$x.test)
```

4.2 Training models and predicting the performance of the algorithms

The approach in **ASML** to algorithm selection is based on building regression models that predict the instance-normalized KPI of each considered algorithm. To this end, users can take advantage of the wide range of ML models available in the **caret** package, which provides a unified interface for training and tuning various types of models. Models trained with **caret** can be seamlessly integrated into the **ASML** workflow using the `AStrain` function from **ASML**, as shown in the next example. Just for illustrative purposes, we use quantile random forest (Meinshausen, 2006) to model the behavior of the instance-normalized KPI based on the features. This is done with the `qrf` method in the **caret** package, which relies on the **quantregForest** package (Meinshausen, 2024).

```
library(quantregForest)
tune_grid <- expand.grid(mtry = 10)
training <- AStrain(data, method = "qrf", tuneGrid = tune_grid)
```

Additional arguments for `caret::train` can also be passed directly to `ASML::AStrain`. This allows users to take advantage of the flexibility of the **caret** package, including specifying control methods (such as cross-validation), tuning parameters, or any other relevant settings provided by `caret::train`. This integration ensures that the **ASML** workflow can fully make use of the modeling capabilities offered by **caret**. To make the execution faster (it is not our intention here to delve into the choice of the best model), we use a `tune_grid` that

sets a fixed value for `mtry`. This avoids the need for an exhaustive search for this hyperparameter, speeding up the model training process. Other modeling approaches should also be considered, as they may offer better performance depending on the specific characteristics of the data and the problem at hand. For more computationally intensive models or larger datasets, the `ASML::AStrain` function includes the argument `parallel`, which can be set to TRUE to enable parallel execution using the **snow** package (Tierney et al., 2021). This allows the training step to be distributed across multiple cores, reducing computation time. A detailed example on a larger dataset is provided in the following section, showing the scalability of the workflow and the effect of parallelization on training time.

The function `caret::train` returns a trained model along with performance metrics, predictions, and tuning parameters, providing insights into the model's effectiveness. In a similar manner, `ASML::AStrain` offers the same type of output but for each algorithm under consideration, allowing straightforward comparison within the **ASML** framework. The `ASML::ASpredict` function generates the predictions for new data by using the models created during the training phase for each algorithm under evaluation. Thus, predictions for the algorithms are obtained simultaneously, facilitating a direct comparison of their performance. By using `ASML::ASpredict` as follows, we obtain a matrix where each row corresponds to an instance from the test set, and each column represents the predicted instance-normalized KPIs for the six branching rules using the `qrf` method.

```
predict_test <- ASpredict(training, newdata = data$x.test)
```

4.3 Evaluating and visualizing the results

One of the key strengths of the **ASML** package lies in its ability to evaluate results collectively and provide intuitive visualizations. This approach not only aids in identifying the most effective algorithms but also contributes to the interpretability of the results, making it easier for users to make informed decisions based on the performance metrics and visual representations provided. For example, the function `KPI_table` returns a table showing the arithmetic and geometric mean of the KPI (both instance-normalized and not normalized) obtained on the test set for each algorithm, as well as for the algorithm selected by the learning model (the one with the largest instance-normalized predicted KPI for each instance). In Table 4, the results for our case study are shown. It is important to note that larger values are better in the columns for the arithmetic and geometric mean of the instance-normalized KPI (where values close to 1 indicate the best performance). Conversely, in the columns for non-normalized values, lower numbers reflect better outcomes. In all cases, the best results are obtained for the ML algorithm. Note also that in this case, the differences in the performance of the algorithms are likely better reflected by the geometric mean because it gives a better representation of relative differences.

```
KPI_table(data, predictions = predict_test)
```

Table 4: Arithmetic and geometric mean of the KPI (both instance-normalized and non-normalized) for each algorithm on the test set, along with the results for the algorithm selected by the learning model (first row).

	Arith. mean inst-norm KPI	Geom. mean inst-norm KPI	Arith. mean non-norm KPI	Geom. mean non-norm KPI
`ML`	0.911	0.887	88114.19	1.035
`max`	0.719	0.367	158716.13	2.574
`sum`	0.791	0.537	104402.53	1.780
`dual`	0.842	0.581	104393.92	1.634
`range`	0.879	0.644	107064.29	1.432
`eig-VI`	0.781	0.474	131194.49	2.007
`eig-CMI`	0.800	0.591	88197.74	1.616

Additionally, the function `KPI_summary_table` generates a concise comparative table displaying values for three different choices: single best, ML, and optimal, see Table 5. The single best choice refers to selecting the same algorithm for all instances based on the lowest geometric mean of the non-normalized KPI (in this case the `range` rule). This approach evaluates the performance of each algorithm across all instances and chooses the one that consistently performs best overall, rather than optimizing for individual instances. The ML choice represents the algorithm selected by the quantile random forest model. The optimal choice corresponds to solving each instance with the algorithm that performs best for that specific instance. The ML choice shows promising results, with a mean KPI close to the optimal choice, demonstrating its capability to select algorithms that yield competitive performance.

```
KPI_summary_table(data, predictions = predict_test)
```

Table 5: Arithmetic and geometric mean of the non-normalized KPI for single best choice, ML choice, and optimal choice.

	Arith. mean non-norm KPI	Geom. mean non-norm KPI
`single best`	107064.29	1.432
`ML`	88114.19	1.035
`optimal`	88085.37	0.911

The following code generates several visualizations that help us compare how well the algorithms perform according to the response variable (instance-normalized KPI) and also illustrate the behavior of the learning process. These plots give us good insights into how effective the algorithm selection process is and how it behaves in comparison to using the same branching rule for all instances. Figure 6 shows the boxplots comparing the performance of each algorithm in terms of the instance-normalized KPI, including the instance-normalized KPI of the rules selected by the ML process for the test set. In Figure 7, the performance is presented by family, allowing for a more detailed comparison across the different sets of instances. In Figure 8, we show the ranking of algorithms based on the instance-normalized KPI for the test sample, including the ML rule, categorized by family. Finally, in Figure 9, the right-side bar in the stacked bar plot (optimal) illustrates the proportion of instances in which each of the original rules is identified as the best-performing option. In contrast, the left-side bar (ML) depicts the frequency with which ML selects each rule as the top choice. Although the rule chosen by ML in each instance doesn't always match the best one for that case, ML tends to select the different rules in a similar proportion to how often those rules are the best across the test set. This means it does not consistently favor a particular rule or ignore any that are the best a significant percentage of instances.

```
boxplots(data, predictions = predict_test, labels = c(lab_rules, "ML"))
boxplots(data, predictions = predict_test, labels = c(lab_rules, "ML"), by_families = TRUE)
ranking(data, predictions = predict_test, labels = c("ML", lab_rules), by_families = TRUE)
figure_comparison(data, predictions = predict_test, by_families = FALSE, labels = lab_rules)
```

4.4 Custom user-defined methods

While **caret** provides a range of built-in methods for model training and prediction, there may be situations where researchers want to explore additional methods not directly integrated into the package. Considering alternative methods can improve the analysis and provide greater flexibility in modeling choices.

In this section, we present an example of how to modify the quantile random forest `qrf` method. The `qrf` implementation in **caret** does not allow users to specify the conditional quantile to predict, which is set to the median by default. In this case, rather than creating

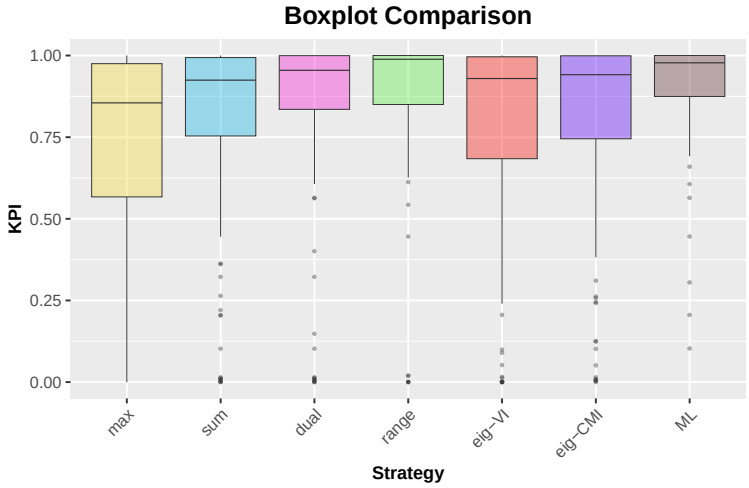

Figure 6: Boxplots of instance-normalized KPI for each algorithm, including the ML algorithm, across instances in the test set.

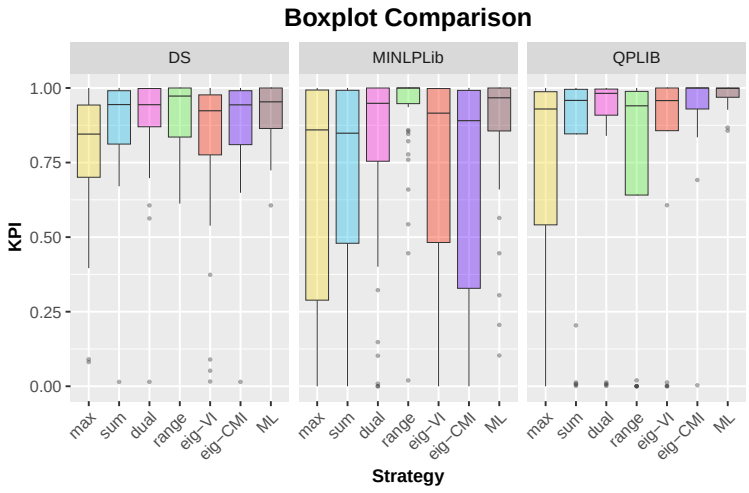

Figure 7: Boxplots of instance-normalized KPI for each algorithm, including the ML algorithm, across instances in the test set, categorized by family.

an entirely new method, we only need to adjust the prediction function to include the `what` argument, allowing us to specify the desired conditional quantile for prediction. In this execution example, we base the algorithm selection method on the predictions of the α-conditional quantile of the instance-normalized KPI for $\alpha = 0.25$.

```
qrf_q_predict <- function(modelFit, newdata, what = 0.5, submodels = NULL) {
    out <- predict(modelFit$finalModel, newdata, what = what)
    if (is.matrix(out)) {
        out <- out[, 1]
    }
    out
}

predict_test_Q1 <- ASpredict(training, newdata = data$x.test, f = "qrf_q_predict",
    what = 0.25)
KPI_summary_table(data, predictions = predict_test_Q1)
```

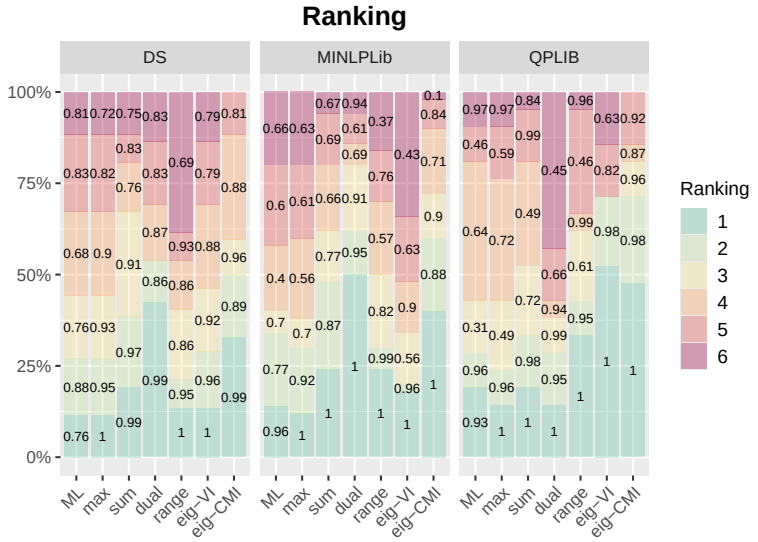

Figure 8: Ranking of algorithms, including the ML algorithm, based on the instance-normalized KPI for the test sample, categorized by family. The bars represent the percentage of times each algorithm appeared in different ranking positions, with the numbers indicating the mean value of the normalized KPI.

4.5 Model interpretability

Predictive modeling often relies on flexible but complex methods. These methods typically involve many parameters or hyperparameters, which can make the models difficult to interpret. To address this, interpretable ML techniques provide tools for exploring *black-box* models. **ASML** integrates seamlessly with the package **DALEX** (moDel Agnostic Language for Exploration and eXplanation), see Biecek (2018). With **DALEX**, users can obtain model performance metrics, evaluate feature importance, and generate partial dependence plots (PDPs), among other analyses.

To simplify the use of **DALEX** within our framework, **ASML** provides the function `ASexplainer`. This function automatically creates **DALEX** explainers for the models trained with `AStrain` (one for each algorithm in the portfolio). Once the explainers are created, users can easily apply **DALEX** functions to explore and compare the behavior of each model. The following example shows how to obtain a plot of the reversed empirical cumulative distribution function of the absolute residuals, from the performance metrics computed with `DALEX::model_performance`, see Figure 10.

```
# Create DALEX explainers for each trained model
explainers_qrf <- ASexplainer(training, data = data$x.test, y = data$y.test, labels = lab_rules)
# Compute model performance metrics for each explainer
mp_qrf <- lapply(explainers_qrf, DALEX::model_performance)
# Plot the performance metrics
do.call(plot, unname(mp_qrf)) + theme_bw(base_line_size = 0.5)
```

The code below illustrates how to obtain feature importance (via `DALEX::model_parts`) and a PDP for the predictor variable `degree` (via `DALEX::model_profile`). Plots are not displayed in this manuscript, but they can be generated by executing the code.

```
# Compute feature importance for each model in the explainers list
vi_qrf <- lapply(explainers_qrf, DALEX::model_parts)
# Plot the top 5 most important variables for each model
do.call(plot, c(unname(vi_qrf), list(max_vars = 5)))
# Compute PDP for the variable 'degree' for each model
pdp_qrf <- lapply(explainers_qrf, DALEX::model_profile, variable = "degree", type = "partial")
# Plot the PDPs generated
```

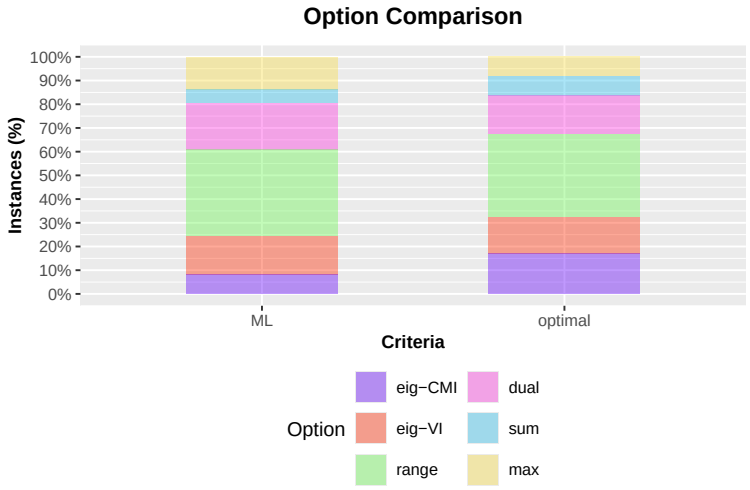

Figure 9: Comparison of the best-performing rules: The right stack shows the proportion of times each of the original rules is identified as the best-performing option, while the left stack presents the frequency of selection by ML.

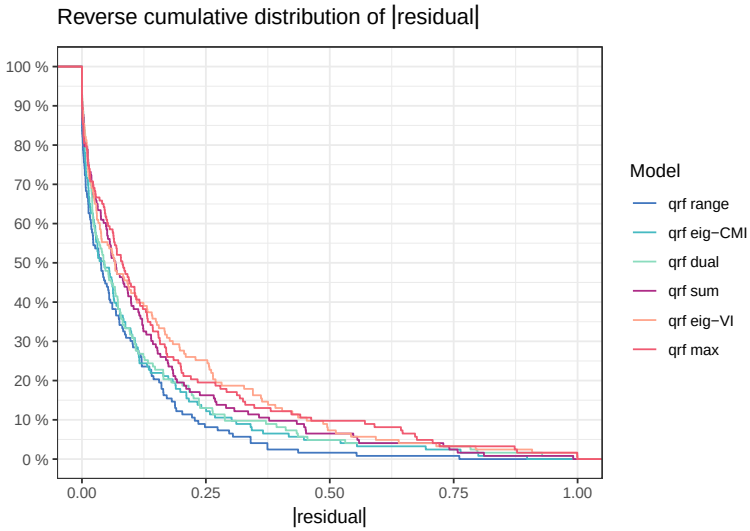

Figure 10: Reversed empirical cumulative distribution function of the absolute residuals of the trained models.

```
do.call(plot, unname(pdp_qrf)) + theme_bw(base_line_size = 0.5)
```

5 Example on a larger dataset

To analyze the scalability of **ASML**, we now consider an example of algorithm selection in the field of high-performance computing (HPC), specifically in the context of the automatic selection of the most suitable storage format for sparse matrices on GPUs. This is a well-known problem in HPC, since the storage format has a decisive impact on the performance of many scientific kernels such as the sparse matrix–vector multiplication (SpMV). For this study, we use the dataset introduced by Pichel and Pateiro-López (2018), which contains 8111 sparse matrices and is available in the **ASML** package under the name `SpMVformat`. Each matrix is described by a set of nine structural features, and the performance of the single-precision SpMV kernel was measured on a NVIDIA GeForce GTX TITAN GPU, under three storage formats: compressed row storage (CSR), ELLPACK (ELL), and hybrid (HYB). For each matrix and format, performance is expressed as the average GFLOPS (billions of floating-point operations per second), over 1000 SpMV operations. This setup allows us to study how matrix features relate to the most efficient storage format.

The workflow follows the standard **ASML** pipeline: data are partitioned and normalized, preprocessed, and models are trained using `ASML::AStrain`. We considered different learning methods available in **caret** and evaluated execution times both with and without parallel processing, which is controlled via the `parallel` argument in `ASML::AStrain`. The selected methods were run with their default configurations in **caret**, without additional hyperparameter tuning. All experiments were performed on a machine equipped with a 12th Gen Intel(R) Core(TM) i7-12700 (12 cores), 2.11 GHz processor and 32 GB of RAM. The execution times are summarized in Tables 6 and 7.

Table 6: Execution times (in seconds) on the SpMVformat dataset for the main preprocessing stages.

Stage	Execution time (seconds)
ASML::partition_and_normalize	0.03
caret::preProcess	1.55

Table 7: Training times (in seconds) on the SpMVformat dataset for different methods using ASML::AStrain. The first column shows execution without parallelization (parallel = FALSE) and the second column shows execution with parallelization (parallel = TRUE).

	Execution times (in seconds) of ASML::AStrain	
Method	parallel = FALSE	parallel = TRUE
nnet	236.58	50.75
svmRadial	881.03	263.60
rf	4753.00	1289.68

The majority of the computational cost is associated with model training, which depends on the learning method in **caret**. We observe that training times vary across methods: `nnet` (a simple feed-forward neural network), `svmRadial` (support vector machines with radial kernel), and `rf` (random forest). Parallel execution substantially reduces training times for all selected methods, demonstrating that the workflow scales efficiently to larger datasets while keeping preprocessing overhead minimal.

Apart from the execution times, we also take this opportunity to provide a brief commentary on the outcome of the algorithm selection in this application example. In particular, we illustrate the model's ability to identify the most efficient storage format by reporting the results obtained with the `nnet` method, see Figure 11. The trained model selects the best-performing format in more than 85% of the test cases, and even when it does not, the chosen format still achieves high performance, with mean value of the normalized KPI (normalized average GFLOPS) around 0.9.

```
set.seed(1234)
data(SpMVformat)
features <- SpMVformat$x
KPI <- SpMVformat$y
data <- partition_and_normalize(features, KPI, better_smaller = FALSE)
preProcValues <- caret::preProcess(data$x.train, method = "YeoJohnson")
data$x.train <- predict(preProcValues, data$x.train)
data$x.test <- predict(preProcValues, data$x.test)
training <- AStrain(data, method = "nnet", parallel = TRUE)
pred <- ASpredict(training, newdata = data$x.test)
ranking(data, predictions = pred)
```

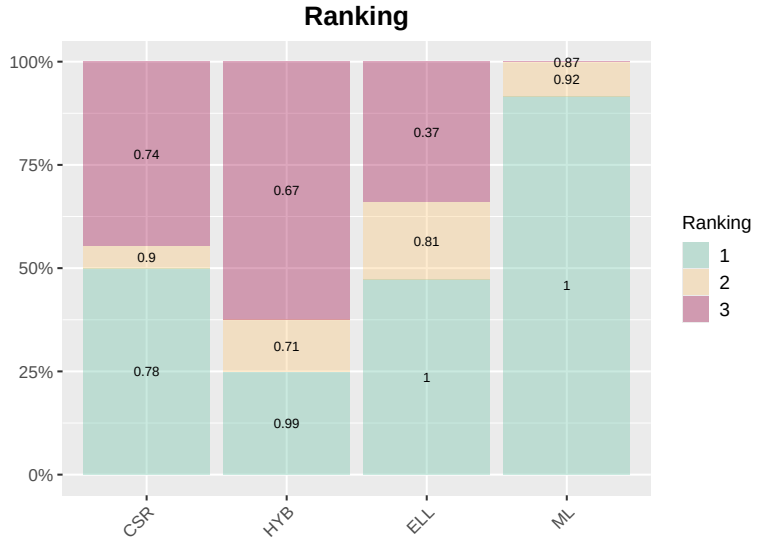

Figure 11: Ranking of storage formats, including the ML selected, based on the instance-normalized KPI for the test sample. The bars represent the percentage of times each storage format appeared in different ranking positions, with the numbers indicating the mean value of the normalized KPI.

6 Using ASML for algorithm selection on ASlib scenarios

While the primary purpose of the **ASML** package is not to systematically conduct algorithm selection studies like those found in ASlib (an area for which the **llama** toolkit is especially helpful), it does offer a complementary approach for reproducing results from the ASlib benchmark (`https://coseal.github.io/aslib-r/scenario-pages/index.html`). Our method allows for a comparative analysis using instance-normalized KPIs, which, as demonstrated in the following example, can sometimes yield improved performance results. Additionally, it can be useful for evaluating algorithm selection approaches based on methods that are not available in the **mlr** package used by **llama** but are accessible in **caret**.

6.1 Data download and preparation

First, we identify the specific scenario from ASlib we are interested in, in this case, `CPMP-2015`. Using the scenario name, we construct a URL that points to the corresponding page on the ASlib website. Then, we fetch the HTML content of the page and create a local directory to store the downloaded files[1].

```
set.seed(1234)
library(tidyverse)
library(rvest)
scen <- "CPMP-2015"
url <- paste0("https://coseal.github.io/aslib-r/scenario-pages/", scen, "/data_files")
page <- read_html(paste0(url, ".html"))
file_links <- page %>%
    html_nodes("a") %>%
    html_attr("href")

# Create directory for downloaded files
dir_data <- paste0(scen, "_data")
dir.create(dir_data, showWarnings = FALSE)
```

[1]A more direct approach would be to use the `getCosealASScenario` function from the **aslib** package; however, this function seems to be currently not working, likely due to changes in the directory structure of the scenarios.

```
# Download files
for (link in file_links) {
    full_link <- ifelse(grepl("^http", link), link, paste0(url, "/", link))
    file_name <- basename(link)
    dest_file <- file.path(dir_data, file_name)
    if (!is.na(full_link)) {
        download.file(full_link, dest_file, mode = "wb", quiet = TRUE)
    }
}
```

6.2 Data preparation with aslib

Now, we use the **aslib** package to parse the scenario data and extract the relevant features and performance metrics. The `parseASScenario` function from **aslib** creates a structured object `ASScen` that contains information regarding the algorithms and instances being evaluated. We then transform this data into cross-validation folds using the `cvFolds` function from **llama**. This conversion facilitates a fair evaluation of algorithm performance across different scenarios, allowing us to compare the results with those published [2].

```
library(aslib)
ASScen <- aslib::parseASScenario(dir_data)
llamaScen <- aslib::convertToLlama(ASScen)
folds <- llama::cvFolds(llamaScen)
```

Then we extract the key performance indicator (KPI) and features from the folds object. In this case, `KPI` refers to runtime. As described in the ASlib documentation, `KPI_pen` measures the penalized runtime. If an instance is solved within the timeout (`cutoff`) by the selected algorithm, the actual runtime is used. However, if a timeout occurs, the timeout value is multiplied by 10 to penalize the algorithm's performance. We also define `nins` as the number of instances and `ID` as unique identifiers for each instance.

```
KPI <- folds$data[, folds$performance]
features <- folds$data[, folds$features]
cutoff <- ASScen$desc$algorithm_cutoff_time
is.timeout <- ASScen$algo.runstatus[, -c(1, 2)] != "ok"
KPI_pen <- KPI * ifelse(is.timeout, 10, 1)
nins <- length(getInstanceNames(ASScen))
ID <- 1:nins
```

6.3 Quantile random forest using ASML on instance-normalized KPI

We use the **ASML** package to perform quantile random forest on instance-normalized KPI. We have already established the folds beforehand, and we want to use those partitions to maintain consistency with the original ASlib scenario design. Therefore, we provide `x.test` and `y.test` as arguments directly to the `partition_and_normalize` function.

```
data <- partition_and_normalize(x = features, y = KPI, x.test = features, y.test = KPI,
    better_smaller = TRUE)
train_control <- caret::trainControl(index = folds$train, savePredictions = "final")
training <- AStrain(data, method = "qrf", trControl = train_control)
```

In this code block, we process the predictions made by the models trained using **ASML** and calculate the same performance metrics used in ASlib, namely, the percentage of solved instances (`succ`), penalized average runtime (`par10`), and misclassification penalty (`mcp`), as detailed in the ASlib documentation (Bischl et al., 2016a).

[2] Available at: https://coseal.github.io/aslib-r/scenario-pages/CPMP-2015/llama.html (Accessed October 25, 2024).

```
pred_list <- lapply(training, function(model) {
    model$pred %>%
        arrange(rowIndex) %>%
        pull(pred)
})

pred <- do.call(cbind, pred_list)
alg_sel <- apply(pred, 1, which.max)

succ <- mean(!is.timeout[cbind(ID, alg_sel)])
par10 <- mean(KPI_pen[cbind(ID, alg_sel)])
mcp <- mean(KPI[cbind(ID, alg_sel)] - apply(KPI, 1, min))
```

In Table 8, we present the results. We observe that, in this example, using instance-normalized KPI along with the quantile random forest model offers an alternative modeling option in addition to the standard regression models employed in the original ASlib study (linear model, regression trees and regression random forest), resulting in improved performance outcomes.

Table 8: Performance results of various models on the CPMP-2015 dataset. The last row represents the performance of the quantile random forest model based on instance-normalized KPI using the **ASML** package. The preceding rows detail the results (all taken from original ASlib study) of the virtual best solver (vbs), single best solver (singleBest), and the considered regression methods (linear model, regression trees and regression random forest).

Model	succ	par10	mcp
baseline vbs	1.000	227.605	0.000
baseline singleBest	0.812	7002.907	688.774
regr.lm	0.843	5887.326	556.875
regr.rpart	0.843	5916.120	585.669
regr.randomForest	0.846	5748.065	540.574
ASML qrf	0.873	4807.633	460.863

It's important to note that this is merely an example of execution and there are other scenarios in ASlib where replication may not be feasible in the same manner, due to factors not considered in **ASML** (for more robust behavior across the ASlib benchmark, we refer to **llama**). Despite these limitations, **ASML** provides a flexible framework that allows researchers to explore various methodologies, including those not directly applicable with **llama** through **mlr**, and improve algorithm selection processes across different scenarios, ultimately contributing to improve understanding and performance in algorithm selection tasks.

7 Summary and discussion

In this work, we present **ASML**, an R package to select the best algorithm from a portfolio of candidates based on a chosen KPI. **ASML** uses instance-specific features and historical performance data to estimate, via a model selected by the user, including any regression method from the **caret** package or a custom function, how well each algorithm is likely to perform on new instances. This helps the automatic selection of the most suitable one. The use of instance-normalized KPIs for algorithm selection is a novel aspect of this package, allowing a unified comparison across different algorithms and problem instances.

While the motivation and examples presented in this work focus on optimization problems, particularly the automatic selection of branching rules and decision strategies in polynomial optimization, the **ASML** framework is inherently flexible and can be applied more broadly. In particular, in the context of ML, selecting the right algorithm is a crucial factor for the success of ML applications. Traditionally, this process involves empirically

assessing potential algorithms with the available data, which can be resource-intensive. In contrast, in the so-called Meta-Learning, the aim is to predict the performance of ML algorithms based on features of the learning problems (meta-examples). Each meta-example contains details about a previously solved learning problem, including features, and the performance achieved by the candidate algorithms on that problem. A common Meta-Learning approach involves using regression algorithms to forecast the value of a selected performance metric (such as classification error) for the candidate algorithms based on the problem features. This method is commonly referred to as Meta-Regression in the literature. Thus, **ASML** could also be used in this context, providing a flexible tool for algorithm selection across a variety of domains.

8 Acknowledgments

The authors would like to thank María Caseiro-Arias, Antonio Fariña-Elorza and Manuel Timiraos-López for their contributions to the development of the **ASML** package. This work is part of the R&D projects PID2024-158017NB-I00, PID2020-116587GB-I00 and PID2021-124030NB-C32 granted by MICIU/AEI/10.13039/501100011033. This research was also funded by Grupos de Referencia Competitiva ED431C-2021/24 and ED431C 2025/03 from the Consellería de Educación, Ciencia, Universidades e Formación Profesional, Xunta de Galicia. Brais González-Rodríguez acknowledges the support from MICIU, through grant BG23/00155.

References

Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2020.07.063. URL https://www.sciencedirect.com/science/article/pii/S0377221720306895. [p1]

P. Biecek. DALEX: Explainers for complex predictive models in R. *Journal of Machine Learning Research*, 19(84):1–5, 2018. URL http://jmlr.org/papers/v19/18-416.html. [p13]

B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren. ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016a. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2016.04.003. [p4, 17]

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016b. URL https://jmlr.org/papers/v17/15-066.html. [p4]

M. R. Bussieck, A. S. Drud, and A. Meeraus. MINLPLib-a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15:114–119, 2003. ISSN 1091-9856. doi: 10.1287/ijoc.15.1.114.15159. [p7]

E. Dalkiran and H. D. Sherali. RLT-POS: Reformulation-linearization technique-based optimization software for solving polynomial programming problems. *Mathematical Programming Computation*, 8:337–375, 2016. ISSN 1867-2949. doi: 10.1007/s12532-016-0099-5. [p7]

J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428, 2020. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2019.07.073. URL https://www.sciencedirect.com/science/article/pii/S0377221719306526. [p1]

F. Furini, E. Traversi, P. Belotti, A. Frangioni, A. Gleixner, N. Gould, L. Liberti, A. Lodi, R. Misener, H. Mittelmann, N. Sahinidis, S. Vigerske, and A. Wiegele. QPLIB: a library

of quadratic programming instances. *Mathematical Programming Computation*, 1:237–265, 2018. ISSN 1867-2957. doi: 10.1007/s12532-018-0147-4. [p7]

B. Ghaddar, I. Gómez-Casares, J. González-Díaz, B. González-Rodríguez, B. Pateiro-López, and S. Rodríguez-Ballesteros. Learning for spatial branching: An algorithm selection approach. *INFORMS Journal on Computing*, 35(5):1024–1043, 2023. doi: 10.1287/ijoc.2022.0090. URL https://doi.org/10.1287/ijoc.2022.0090. [p1, 5]

B. González-Rodríguez, I. Gómez-Casares, B. Ghaddar, J. González-Díaz, and B. Pateiro-López. Learning in spatial branching: Limitations of strong branching imitation. *INFORMS Journal on Computing*, 2025a. [p5]

B. González-Rodríguez, I. Gómez-Casares, B. Pateiro-López, and J. González-Díaz. *ASML: Algorithm Portfolio Selection with Machine Learning*, 2025b. URL https://CRAN.R-project.org/package=ASML. R package version 1.1.0. [p2]

P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation*, 27(1):3–45, 03 2019. ISSN 1063-6560. doi: 10.1162/evco_a_00242. URL https://doi.org/10.1162/evco_a_00242. [p1]

L. Kotthoff. *Algorithm Selection for Combinatorial Search Problems: A Survey*, pages 149–190. Springer International Publishing, Cham, 2016. ISBN 978-3-319-50137-6. [p1, 3]

L. Kotthoff, B. Bischl, B. Hurley, T. Rahwan, and D. Pulatov. *llama: Leveraging Learning to Automatically Manage Algorithms*, 2021. URL https://CRAN.R-project.org/package=llama. R package version 0.10.1. [p4]

Kuhn and Max. Building predictive models in R using the caret package. *Journal of Statistical Software*, 28(5):1–26, 2008. doi: 10.18637/jss.v028.i05. URL https://www.jstatsoft.org/index.php/jss/article/view/v028i05. [p2]

M. Lang, M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl. mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*, dec 2019. doi: 10.21105/joss.01903. URL https://joss.theoj.org/papers/10.21105/joss.01903. [p4]

A. Lodi and G. Zarpellon. On learning and branching: a survey. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 25(2):207–236, July 2017. doi: 10.1007/s11750-017-0451-6. URL https://ideas.repec.org/a/spr/topjnl/v25y2017i2d10.1007_s11750-017-0451-6.html. [p1]

N. Meinshausen. Quantile regression forests. *Journal of Machine Learning Research*, 7:983–999, Dec. 2006. ISSN 1532-4435. [p9]

N. Meinshausen. *quantregForest: Quantile Regression Forests*, 2024. URL https://CRAN.R-project.org/package=quantregForest. R package version 1.3-7.1. [p9]

T. Messelis and P. De Causmaecker. An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 233(3):511–528, 2014. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2013.08.021. URL https://www.sciencedirect.com/science/article/pii/S0377221713006863. [p1]

J. C. Pichel and B. Pateiro-López. A new approach for sparse matrix classification based on deep learning techniques. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 46–54, 2018. doi: 10.1109/CLUSTER.2018.00017. [p14]

D. Pulatov, M. Anastacio, L. Kotthoff, and H. Hoos. Opening the black box: Automated software analysis for algorithm selection. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pages 6/1–18. PMLR, 25–27 Jul 2022. URL https://proceedings.mlr.press/v188/pulatov22a.html. [p3]

J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976. [p3]

H. D. Sherali and C. H. Tuncbilek. A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *Journal of Global Optimization*, 2(1):101–112, 1992. doi: 10.1007/bf00121304. [p5]

D. Speck, A. Biedenkapp, F. Hutter, R. Mattmüller, and M. Lindauer. Learning heuristic selection with dynamic algorithm configuration. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS21)*, aug 2021. URL https://arxiv.org/abs/2006.08246. [p1]

L. Tierney, A. J. Rossini, N. Li, and H. Sevcikova. *snow: Simple Network of Workstations*, 2021. URL https://CRAN.R-project.org/package=snow. R package version 0.4-4. [p10]

J. Vanschoren. *Meta-Learning*, pages 35–61. Springer International Publishing, 2019. ISBN 978-3-030-05318-5. [p1]

Ignacio Gómez-Casares
Universidade de Santiago de Compostela
Department of Statistics, Mathematical Analysis and Optimization
Santiago de Compostela, Spain
ignaciogomez.casares@usc.es

Beatriz Pateiro-López
Universidade de Santiago de Compostela
Department of Statistics, Mathematical Analysis and Optimization
CITMAga (Galician Center for Mathematical Research and Technology)
Santiago de Compostela, Spain
ORCiD: 0000-0002-7714-1835
beatriz.pateiro@usc.es

Brais González-Rodríguez
Universidade de Vigo
Department of Statistics and Operational Research
SiDOR Research Group
Vigo, Spain
brais.gonzalez.rodriguez@uvigo.gal

Julio González-Díaz
Universidade de Santiago de Compostela
Department of Statistics, Mathematical Analysis and Optimization
CITMAga (Galician Center for Mathematical Research and Technology)
Santiago de Compostela, Spain
ORCiD: 0000-0002-4667-4348
julio.gonzalez@usc.es