Ludwig-Maximilians-Universität München Department of Statistics

Bachelor's Thesis

Visualizing Hyperparameter Performance Dependencies

Author

Simon Pradel

Supervisor

Dr. Janek Thomas, M.Sc. Florian Pfisterer,

M.Sc. Lennart Schneider

Munich, February 9, 2022

Abstract

The settings of hyperparameters are often decisive whether an algorithm, for instance from the field of machine learning, delivers good performance values or not. The choice of the hyperparameter configuration is often based on the result of a hyperparameter optimization algorithm such as Bayesian optimization. However, the tractability of the delivered results is poorly understood in this context, as it neither gives a deeper insight into the structure of achievable performances generated by different hyperparameter configurations nor provides information on how relevant each hyperparameter is for the selected performance measure. In the theoretical part of this paper, several methods are presented that can help to understand the relationship between hyperparameters and a performance measure. As a result of this work, a new R package is presented including four of the shown visualization methods, namely parallel coordinate plots, partial dependence plots, importance plots and heatmaps. Furthermore, an interactive use of the integrated plots is enabled in the R package by a Shiny application. At the end of this work, an application study of two datasets is used to demonstrate the insights that can be gained for optimal hyperparameter configurations through the use of the new R package.

Contents

1 Introduction

In the field of machine learning, most algorithms (e.g. random forests or deep neural networks) contain hyperparameters. A hyperparameter is a categorical or continuous parameter that is set before the data learning process and often influences decisive factors of the algorithm such as the complexity or the speed (Bischl et al., 2021). When choosing a hyperparameter value, it is important that it should be set as optimally as possible, since it can have a crucial influence on the model performance (Claesen and De Moor, 2015). One way to find approximately optimal hyperparameter configurations is to select them manually. However, since there are usually a myriad of hyperparameter combinations and settings, this approach is unlikely to result in a satisfactory algorithm performance. In addition, manual tuning is further complicated by non-linear hyperparameter interactions and time-consuming model evaluations (Yang and Shami, 2020). Another possibility to search for optimal hyperparameter configurations is through automatic algorithms. In the field of machine learning, this is often referred to as the hyperparameter optimization problem (Bischl et al., 2021). Many automatic machine learning algorithms such as the Bayesian optimization (Snoek et al., 2012) or hyperband (Li et al., 2017) exist. The superiority of algorithms compared to manual searches is often demonstrated in increased performance, reduced human effort, and reproducibility. (Feurer and Hutter, 2019).

Nevertheless, the results can be difficult to understand due to complicated algorithms that do not provide explanations for the relationship between hyperparameters and performance. Using methods that provide a visual insight into the dependency of the performance measure of the hyperparameters can help to increase the general understandability and traceability. Visually represented methods can, for example, illustrate how important individual parameters are, which configuration spaces of the hyperparameters lead to a positive or negative effect on performance, and whether there are interaction terms that have a significant impact on the results (Hutter et al., 2014).

This is a relatively under-researched area in literature, but became more and more important in recent years. Basically, two tasks are involved here. On the one hand, methods have to be found that are suitable for the investigation of hyperparameter performance dependencies, and on the other hand, software is needed that enables the investigation in a user-friendly way. An example of the use of visualization methods for our topic is provided by the work of Hutter et al. (2014). In this approach, the calculation of marginal performance using the random forest is proposed to show the effects of hyperparameters on the performance measure. Furthermore, using the results, functional ANOVA is used to calculate the importance of hyperparameters and their interactions. In other work, software have been developed to support the selection of hyperparameters and make them more comprehensible. For instance, the approach of Joo et al. (2021) introduced a visual analysis tool to support the search for optimal hyperparameter values. Another approach by Park et al. (2020) presented a software called HyperTendril, which is a system for visual analysis and control of the hyperparameter optimization process for deep neural networks.

The goal of this work is to provide visualization methods that allow a deeper understanding of various hyperparameter configurations. For this purpose, the present work makes the following contributions:

- existing visualization methods are presented, categorized and compared;
- four different visualization methods, namely partial dependence plot, importance plot, heatmap and parallel coordinate plot, are implemented in a new R package (Team, 2020);
- by using a Shiny application (Chang et al., 2021), also implemented in the R package, these plots can be used interactively;
- in a final application study, the implemented methods are used to generate knowledge about the dependencies of the performance measure on the hyperparameter configuration spaces of two datasets.

In order to provide a meaningful structure, the present thesis is organized as follows. In chapter 2 an overview of different visualization methods is provided. In subsection 2.1 some basic terms and notations are clarified. In the following subsections 2.2 to 2.5 the different methods are discussed in more detail so that a clear distinction and comparison can be made. Subsequently, an overview of the results of this work is given in section 3. In the subsection 3.1, a new R package called `VisHyp` is introduced, which makes it possible to analyze a dataset with four different visualization methods. For interactive analyzes, a Shiny application is also included in the R package. In addition, an application study is performed in subsection 3.2, showing that knowledge about good configuration spaces of hyperparameters can be generated using the plots of the `VisHyp` package. Finally, in the concluding section 4, a summary of the findings and an outlook on further research possibilities is given. A limitation of the bachelor's thesis is that only a fraction of the possible visualization methods can be presented. For more visualization methods that are transferable to this work, one can refer to the book by Molnar (2019).

2 Methods for the Visualization of Hyperparameter Dependencies

Visualization methods are helpful to understand the relationship between hyperparameters and a performance measure and are discussed in this chapter. For this purpose, the basic notation of the theory is explained first before the different methods are presented, which are divided into four categories. An overview of the different methods for each category can be found in table 1. The methods of the first category do not require a surrogate model and therefore are used for direct examination of the dataset. We refer to these methods as descriptive methods and they are mainly used to get a quick overview of the parameters and to identify some first interactions and dependencies. In the following subsection parameter effect methods are presented. These methods calculate and visualize the effects of one or two parameters on these performance measure depending on the parameter configurations applied. Based on the results, parameters can be set manually, as these methods output the estimated average performance values for each possible parameter configuration. In the following category parameter importance methods for determining the importance of hyperparameters are discussed. Importance methods are often a first approach, as they show which parameters should be examined in more detail with parameter effect plots in order to set them optimally as well as which parameters are possibly even negligible as their configurations only have a minor effect on the performance results. Finally, in the last category, methods for finding interaction effects are presented, since these can increase performance in addition to the main effects of the individual parameters. With the so-called interaction detection methods it is possible to find the strongest interaction effects in order to investigate them specifically again with methods from the two categories parameter effects and descriptive analysis.

Table 1: All visualizations methods mentioned in the bachelor's thesis.

Classification	Methods	Literature
Descriptive Analysis	Parallel coordinate plot	Gannett and Hewes (1883); Joo et al. (2021)
	Heatmap	Loua (1873); Hamid et al. (2019)
	Scatterplot	Helton and Kleijnen (1999)
	Colored scatterplot	Pianosi et al. (2016)
	Principal component analysis	Wang et al. (2019)
Parameter Effects	Partial dependence plot	Friedman (2001)
	Marginal hyperparameter performance	Hutter et al. (2014)
	ICE plot	Goldstein et al. (2015)
	ALE plot	Apley and Zhu (2020)
	Sensitivity analysis methods	Cortez and Embrechts (2013)
	Further methods	Molnar (2019)
Parameter Importance	Permutation parameter importance plot	Breiman (2001); Fisher et al. (2019)
	Functional ANOVA	Hutter et al. (2014)
	Local parameter importance	Biedenkapp et al. (2018)
	Break-down plot	Staniak and Biecek (2018)
	Sensitivity analysis methods	Iooss and Lemaître (2015)
Interaction Detection	H-statistic	Friedman and Popescu (2008)
	PDP based interaction detection	Greenwell et al. (2018)
	Variable interaction networks	Hooker (2004)

2.1 Notation

To visualize hyperparameters, one can think of hyperparameter settings and estimated performance as column entries in a dataset. The research subjects are hyperparameters, but in our setting hyperparameters are treated like features. Therefore, in the context of this work, the terms parameter, hyperparameter, and feature are used interchangeably to describe hyperparameters. The base notation to describe hyperparameters and configurations mainly follows Hutter et al. (2014) and has been adapted for this setting. A parameter is named x and can take values from its domain $\mathcal{X}$, which can be numeric or categorical. The configuration space of p

hyperparameters is $\boldsymbol{\mathcal{X}} = \mathcal{X}_1 \times ... \times \mathcal{X}_p$ and the set of all parameters $\{1,...,p\}$ can be denoted by P. A single instantiation is denoted as $\boldsymbol{x} = \langle x_1,...,x_p \rangle$ with $x_i \in \mathcal{X}_i$. The partial instantiation of a subset $U = \{u_1,...,u_q\} \subseteq P$ is $\boldsymbol{x}_U = \langle x_{u_1},...,x_{u_q} \rangle$ with $x_{u_i} \in \mathcal{X}_{u_i}$. The extension set $X(\boldsymbol{x}_U)$ of a partial instantiation $\boldsymbol{x}_U$ is defined as $\boldsymbol{x}_{P|U} = \langle x'_1,...,x'_p \rangle$ such that $\forall j(j = u_k \Rightarrow x'_j = x_{u_k})$. In case not single hyperparameter configurations are of interest, but the dataset containing all hyperparameter configurations and targets, the notation of Fisher et al. (2019) is followed. A row in a dataset is the combination of a parameter instantiation and one or more target variables. The combinations of all instantiations $\boldsymbol{x}^{(1)},...,\boldsymbol{x}^{(n)}$ in a dataset is a matrix with n observations and denoted as $\mathbf{X}$. The column of a single target variable representing the predicted performance of the hyperparameter configurations is called y and is a vector labeled $\mathbf{y}$. When a predictive model is used, it is denoted by $\hat{f} : \mathcal{X} \rightarrow \mathbb{R}$. A loss function is denoted by L and the expected loss result is denoted by the term model error e.

2.2 Descriptive Analysis

Descriptive analysis is used for visualization raw data without the support of a surrogate model. Its methods are usually easy to use, understand, and not computationally intensive. In the following sections well-known hyperparameter visualization methods are shown that can be used to analyze hyperparameter performance dependencies. For visual examples, the hyperparameters for recursive partitioning and regression trees (rpart) from the iaml_rpart dataset are used.

2.2.1 Parallel Coordinate Plot

A very frequently used tool for the visualizations of hyperparameters performance dependencies are parallel coordinate plots (Joo et al., 2021). Parallel coordinate plots (PCP) were developed at the latest by Gannett and Hewes (1883) and enable the visualization of high-dimensional data in a two-dimensional graphic for quick diagnoses. Here, each vertical line presents the range of a single hyperparameter. For each hyperparameter configuration, the values are marked on the respective axis and connected by a line, with each line given a color corresponding to the value of the performance. The order of the axes representing hyperparameters play an essential role. For example, if two categorical parameters with only a few levels are displayed side by side, the lines between the axes will overlap and a identification of the individual configurations will not be possible. This problem can be circumvented by creating a dynamic plot so that the axes can be moved. A screenshot of a dynamically PCP can be seen in figure 1.

In this graphic, four hyperparameters are shown together with the *logloss* performance measure. The red lines indicate configurations with a high performance and the blue lines configurations with a poor performance. For instance, it can be seen that configurations with low *cp* seem to produce lower *logloss* values and that the *minsplit* parameter does not have a large impact on the outcome. Further, it can be seen that this method provides a useful way to find potentially good ranges for parameter configurations in higher dimensional data. However, the

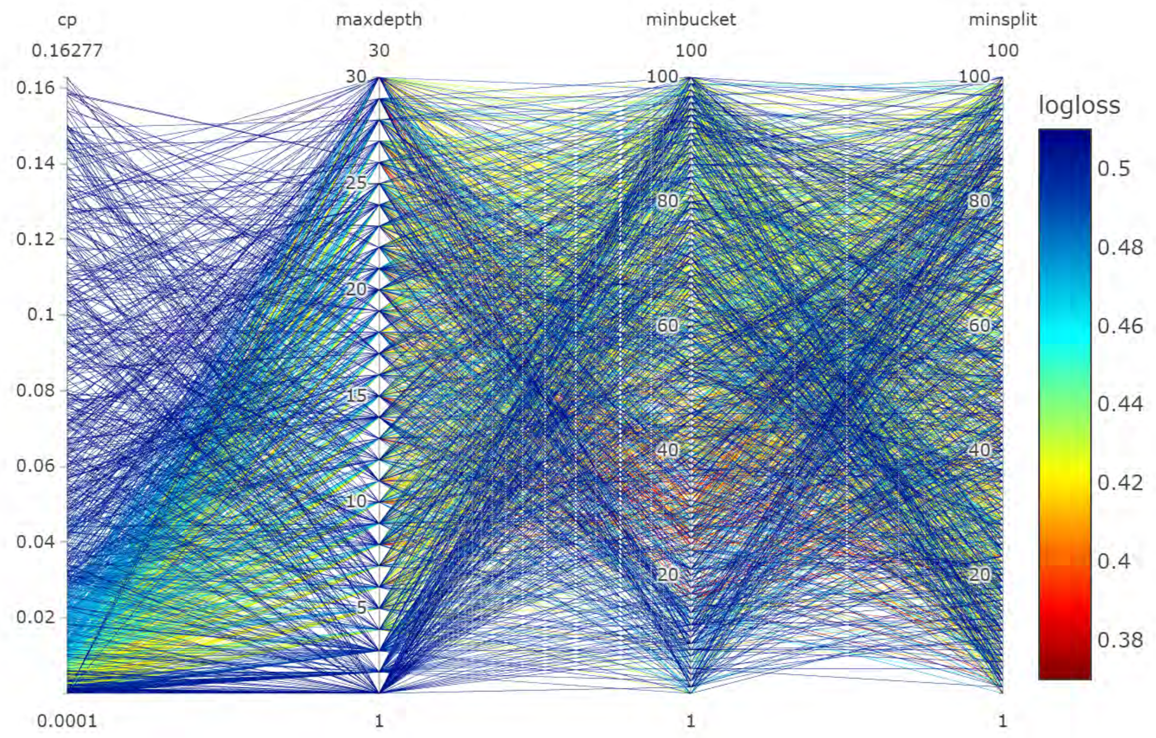


Figure 1: A parallel coordinate plot for four hyperparameters and the logloss performance measure of the iaml_rpart dataset. Red lines indicate good parameter configuration, dark blue lines indicate configurations with poor performance.

PCP quickly reaches its limits by increasing the numbers of configurations, which also increases the number of lines. This can result in a confusing display and makes it difficult to distinguish potentially good areas from seemingly worse ones. Even though there are alternatives to plotting with lines like RadViz plots or Andrew's curves (Dzemyda et al., 2013) these options share the same problems and more. For example, the RadViz plots are even more obscure due to the representation of the axes inside a circle, and Andrew's curves lose the values of the individual configurations. Although, too many lines in a PCP can lead to confusing plots and thus difficulties in finding good configuration spaces, PCPs are still a good option to get a first impression of the dependencies. In addition, they show their strengths when fewer configurations are plotted, for example, when a large dataset is meaningfully filtered or only a small dataset is considered.

2.2.2 Heatmap

Another way to visualize the relationship between parameters and performance are heatmaps. A heatmap shows performance as a function of two features (Hamid et al., 2019) and originates from shaded matrix display by Loua (1873). In case of multiple hyperparameters, one can use multiple heatmaps to analyze the respective dependence of the performance measure of two parameters and thereby find possible interactions (Park et al., 2020). To calculate a heatmap, the configuration spaces of the two parameters under consideration are divided into spaces of equal size. Then, a grid of cells is created using the Cartesian product. Finally, a summary function δ can be applied to each cell. Usually the mean is used for the function to get an overview of the average performance, but other options such as the standard deviation are also conceivable. In algorithm 1 the computation of the heatmap can be seen in more detail.

Algorithm 1 ComputeHeatmap($\mathbf{X}_i$, $\mathbf{X}_j$, $\mathbf{y}$, δ)

```
**Input**: $\mathbf{X}_i = (x_i^{(1)},...,x_i^{(n)})$, $\mathbf{X}_j = (x_j^{(1)},...,x_j^{(n)})$ $\mathbf{y} = (y^{(1)},...,y^{(n)})$, $\delta$
**Output**: Result of a summary function $\delta$
initializes an empty matrix $M$ with r rows and s columns
$\mathcal{X}_i = [c_0,c_1[,...,[c_{r-1},c_r[$                         ▷ Split the domain of $\mathcal{X}_i$
$\mathcal{X}_j = [d_0,d_1[,...,[d_{s-1},d_s[$                         ▷ Split the domain of $\mathcal{X}_j$
$[c_{k-1},c_k[\times[d_{l-1},d_l[,\ k=1,...,r,\ l=1,...,s$            ▷ Create a grid of cells
for all $k \in 1,...,r$ do
    for all $l \in 1,...,s$ do
        $z \leftarrow \{\emptyset\}$                                  ▷ Initialize an empty vector
        for all t $\in 1,...,n$ do
            if $(x_i^{(t)},x_j^{(t)}) \in [c_{k-1},c_k[\times[d_{l-1},d_l[$ then
                $z \leftarrow z \cup y^{(t)}$
            end if
        end for
        $\delta(z)$                                                   ▷ Calculation of the summary result
        $M^{(k,l)} \leftarrow \delta(z)$                              ▷ Add the result of $\delta(z)$ to the matrix $M$
    end for
end for
return $M$
```

It should be noted that other parameters are completely ignored by the calculation of a heatmap. This means that the effect of other parameters is not kept constant, so that it cannot be said with certainty whether the performance is caused only by the selected parameters. Nevertheless, a heat map is useful to get an overview of the different parameter configurations and can be displayed quickly without additional models or calculations.

For a visual representation, the parameters are plotted on the axes of a Cartesian coordinate system and the cells in the grid are colored according to the result. For illustration, figure 2 shows a heatmap for the hyperparameters *minbucket* and *maxdepth* and the performance measure *logloss* of the iaml_rpart dataset. Dark blue cells indicate configurations with a performance above average (low logloss), whereas light blue cells indicate configurations with a performance below the average (high logloss). Cells without a color indicate that there are no combinations of values for these two parameters available. It appears that poor performance values are associated with low *maxdepth* values but also the *minbuck* parameter should not be set too low.

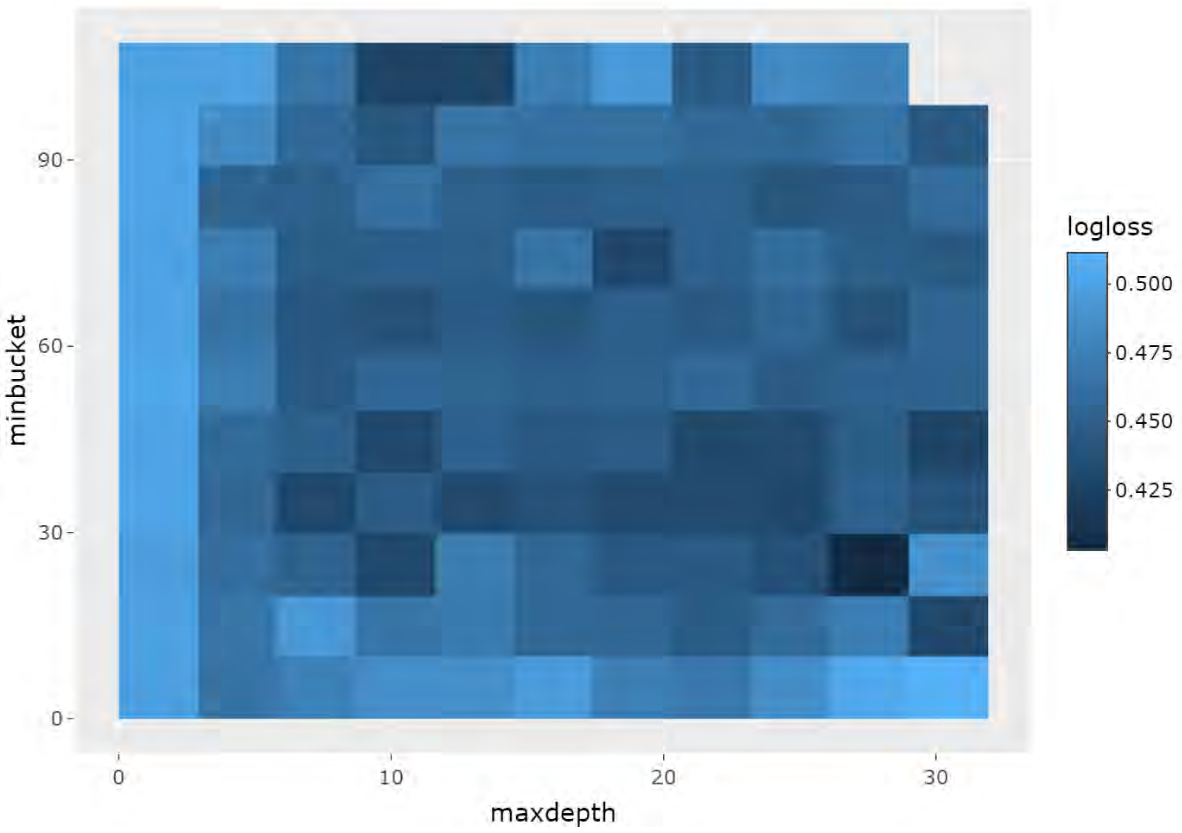


Figure 2: A heatmap for the two hyperparameters *minbucket* and *maxdepth* as well as the performance measure *logloss* of the iaml_rpart dataset. Empty cells mean that no combination of the two parameters are available. In this graphic, low values for both parameters seem to lead to worse performance values.

2.2.3 Other Methods

A very common method for visualizing the relationship between an input variable and an output variable are scatterplots (Helton and Kleijnen, 1999). Scatterplots are quite easy to use and understand and can be helpful for studying the relationship between parameters and their performance measure. However, a simple scatterplot ignores all other parameters, so possible interactions cannot be found. A possibility to find interactions is to extend the simple scatterplot to a colored one, where two parameters are plotted against each other and a color is added to each point according to the value of its performance (Pianosi et al., 2016). Unfortunately, they also have the disadvantage mentioned in subsection 2.2.1, since a huge number of configurations can lead to a confusing display. An example of a simple as well as a colored scatterplot can be found in figure 14 in the appendix.

Theoretically, dimensionality reduction methods such as principal component analysis are also possible to analyze the dependence of performance on parameters (Wang et al., 2019). However, since the results of such methods lose the details of the hyperparameter values that are important for finding good configurations, they are not recommended for our purpose.

2.3 Parameter Effects

While the descriptive analysis methods presented in the previous subsection attempt to find a good hyperparameter configuration using the raw data, the parameter effect methods presented in this subsection use a surrogate model. Surrogate models estimate the performance of parameter configurations corresponding to a trained model. Using such a model results in a loss of accuracy, since they only approximate the values, but they offer the advantage of being able to estimate a performance value for each configuration. When choosing a suitable surrogate model, random forest is often used in literature since it is accurate and flexible (Breiman, 2001) and is therefore also used for the evaluations in this work. Another difference to the methods from the subsection of description analysis is that the following methods determine the dependencies of the performance of hyperparameters only from a few parameters, as in the heatmap or the scatterplot, and not from all combinations, as in the PCP.

2.3.1 PDP

A partial dependence plot (PDP) is a visualization tool that plots average predicted values based on the marginal effect of one or two parameters and is particularly useful for interpreting black-box models with higher dimensions (Friedman, 2001). The definition of the partial dependence function is given by:

$$\overline{f}_U(\boldsymbol{x}_U) = E_{\boldsymbol{x}_C}[\widehat{f}(\boldsymbol{x}_U, \boldsymbol{x}_C)] = \int \widehat{f}(\boldsymbol{x}_U, \boldsymbol{x}_C) p_C(\boldsymbol{x}_C) d\boldsymbol{x}_C.$$

The partial dependence function $\hat{f}$ is a machine learning model that computes the average predicted outcome for a subset of fixed parameter values $\boldsymbol{x}_U$. For the computation, the complement parameter vector $\boldsymbol{x}_C$ is defined as $\boldsymbol{x}_C = \boldsymbol{x} \setminus \boldsymbol{x}_U$ and is varied over its marginal probability density $p_C(\boldsymbol{x}_C)$. Thus, the output of the function shows the marginal effect of the parameter in $\boldsymbol{x}_U$ on the predicted values. Since the marginal probability density is not known, the result must be estimated using different $\boldsymbol{x}_C$ values from the data. For this, the formula can be rewritten as a function that averages $\widehat{f}(\boldsymbol{x}_U, \boldsymbol{x}_C)$, where $\boldsymbol{x}_U$ is fixed and $\boldsymbol{x}_C$ varies over its n observations:

$$\overline{f}_U(\boldsymbol{x}_U) = \frac{1}{n} \sum_{i=1}^{n} \widehat{f}(\boldsymbol{x}_U, \boldsymbol{x}_C^{(i)}).$$

For categorical parameters, we obtain a PDP estimate by forcing all data instances into the same category. More specifically, to calculate the value for a particular category of a categorical parameter, all other categories of all data instances are replaced by the desired category and then the average of the predictions is calculated (Molnar, 2019). The goal of the PDP is to compare the estimated marginal performance of one or two parameters for different configurations. Therefore, different combinations are created for the considered parameters within their parameter space and then the performance is estimated. In the R package `iml` a gridsize is used for this, which divides the parameter space evenly (Molnar et al., 2018). With the calculation of the average performance of different configuration tuples, the PDP can then finally be visualized.

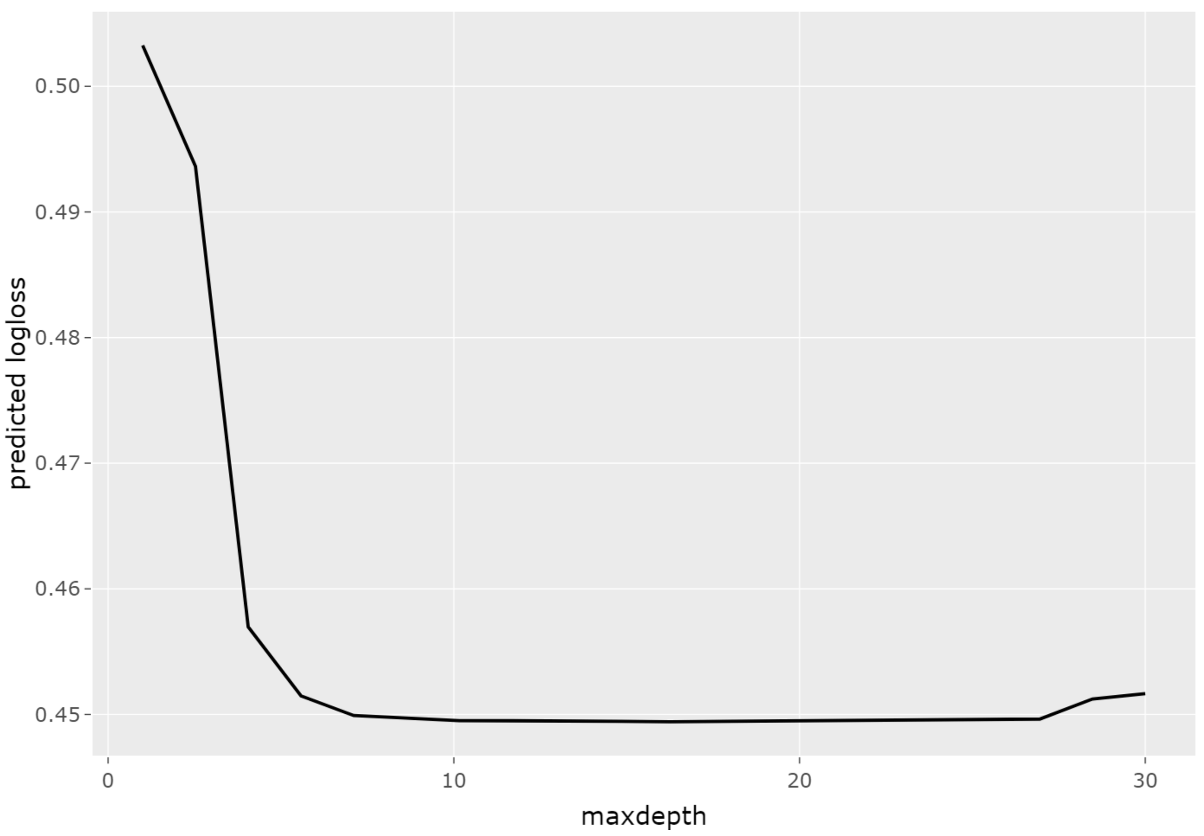


Figure 3: A partial dependence plot for the hyperparameter *maxdepth* of the iaml_rpart dataset. It can be seen that high values achieve better performance values because the predicted logloss values are lower.

The results of a single numerical parameter can be plotted in a line graph, exemplified in figure 3. For a categorical parameter, a bar chart can be used to show the estimated average marginal performance. The results of two numeric or two categorical parameters can be visualized in a heatmap. For a numerical and a categorical parameter, a line can be displayed for each category. In general, the subset $\boldsymbol{x}_U$ has no more than one or two variables because it is difficult to represent higher dimensional data in a graphical representation. In addition, it is useful to plot a rug together with the results, otherwise areas with almost no data could be over-interpreted. While PDPs have the disadvantage that the calculation of marginal effects is computationally intensive, they do provide helpful hints for good hyperparameter configurations and are well suited for the analysis of single hyperparameters or hyperparameter pairs (Moosbauer et al., 2021b). When a PDP is plotted in a heatmap, the interaction of two parameters is captured, providing additional information about the joint behavior of the parameters.

2.3.2 Marginal Hyperparameter Performance

An alternative way to calculate the marginal effect of a partial instantiation of hyperparameters is the marginal performance approach by Hutter et al. (2014). Compared to the PDP, where only the combinations of the n observations were used to estimate the marginal performance, this approach aims to estimate the marginal performance over all possible instances. This is obviously an impossible task, since, for example, continuous hyperparameters have infinite configuration values. As a possible solution Hutter et al. (2014) proposed the marginal performance to make an approximate analysis based on a model and the use of the following formula:

$$\hat{a}_U(\boldsymbol{x}_U) = E[\hat{f}(\boldsymbol{x}_{P|U})|\boldsymbol{x}_{P|U} \in X(\boldsymbol{x}_U)].$$

The function of the marginal performance $\hat{a}_U$ takes a partial instantiation $\boldsymbol{x}_U$ as an input which is the same as in the PDP. For the calculation, a model $\hat{f}$ is used which takes the vector of an extension set $X(\boldsymbol{x}_U)$ of the partial instantiation $\boldsymbol{x}_U$ as input. When analyzing hyperparameters, the values $\boldsymbol{x}_U$ in the entire hyperparameter space are usually of interest, so we assume that $X(\boldsymbol{x}_U)$ is uniformly distributed and has the probability density $\frac{1}{||X(\boldsymbol{x}_U)||}$. By definition the formula of the marginal performance can be rewritten as:

$$\hat{a}_U(\boldsymbol{x}_U) = \frac{1}{||X(\boldsymbol{x}_U)||} \int \hat{f}(\boldsymbol{x}_{P|U}) d\boldsymbol{x}_{P\backslash U}.$$

The first part refers to the probability density over $X(\boldsymbol{x}_U)$. This is similar to the probability density function or a probability mass function of a uniform distribution where the range size is in the denominator. For Hutter's formula, the range size $||S||$ for both continuous and categorical hyperparameters are defined as follows. The range size $||S||$ of an empty set S is defined as 1; for other finite S, the range size is equal to the cardinality $||S|| = |S|$. For closed intervals $S = [l,u] \subseteq \mathbb{R}$ with $l < u$, the range size is defined as $||S|| = u - l$ and for cross products the definition is $S = S_1 \times ... \times S_k$, where $||S|| = \prod_{i=1}^{k} ||S_i||$. Because the fixed parameters are considered as scalars in the calculation of cross products, the formula can be rewritten again so that $\frac{1}{||X(\boldsymbol{x}_U)||} = \frac{1}{||\boldsymbol{\mathcal{X}}_T||}$, where $T = \{t_1,...,t_k\} = P\backslash U$ and $\boldsymbol{\mathcal{X}}_T = \mathcal{X}_{t_1} \times ... \times \mathcal{X}_{t_k}$:

$$\hat{a}_U(\boldsymbol{x}_U) = \frac{1}{||\boldsymbol{\mathcal{X}}_T||} \int \hat{f}(\boldsymbol{x}_{N|U}) d\boldsymbol{x}_T.$$

Ultimately, the expected marginal performance of an instantiation is estimated from the results of a surrogate model. Hutter explains in his paper that a particularly good model $\hat{f}$ is the random forest because the marginally predicted performance can be computed in linear time and also provides high performance predictions for a wide range of highly parameterized algorithms. With his explanation, he also provides the necessary theory to apply the marginal performance formula on a random forest.

Both the PDP and marginal performance compute the average marginal performance for a given instantiation using a machine learning model, such as the random forest. While the PDP takes the surrogate model to estimate and then average the performance of existing data, this approach uses a surrogate model $\hat{f}$ to try to approximate the true marginal performance function. Just as with PDP, the result for a given partial instantiation is just a single value, but as U is varied, we can create a similar line graph or heatmap showing the marginal performance of one or two parameters.

2.3.3 Other Methods

Both of the mentioned methods are already used for visualization hyperparameter configurations. However, since the concept of hyperparameter effects is no different from feature effects in terms of visualization, there are other conceivable ways from the field of machine learning to illustrate the connection between hyperparameters and performance measures. In the following, we briefly review ICE curves and the ALE plot to present an extension and an alternative to the PDP. Many alternative methods from the field of interpretable machine learning can be found in Molnar (2019), which presents a summary of different methods for model interpretation.

As just mentioned, an important addition to the PDP is the ICE plot (Goldstein et al., 2015). While the PDP displays the partial effect between the hyperparameters and the performance on average, the ICE plot shows the functional relationship for individual observations. The creation of an ICE plot for numerical parameters is done by drawing one line per observation and can help to find heterogeneous relationships between hyperparameters and performance through interactions. For categorical features, the ICE curves can be taken to expand the barplot to a boxplot. This helps to see how the performance values are distributed. An example can be viewed in Figure 15 in the appendix. In general, an ICE plot is a useful addition to the PDP, but it can take a long time to render the lines if there are many observations. In addition, ICE curves are only applicable for one parameter, because at higher dimensions the individual curves would no longer be recognizable, since they all overlap.

A good alternative to the PDP is the ALE plot (Apley and Zhu, 2020). It also describes how a parameter affects performance on average, but can calculate it faster due to the differences in mathematical calculation. However, ALE plots cannot be drawn together with ICE curves, which add value to the PDP. Figure 4 shows the visualization of the three different methods PDP, ICE curves and ALE plot. It can be seen that both methods show that the $maxdepth$ parameter of the iaml_rpart dataset should not be set too low in order to achieve a high average performance (low logloss). Nevertheless, too high values could also have a negative impact on the result, as shown in particular by the ICE curves. Furthermore, it can be seen that the ALE plot centers the performance values and thus also differs from the PDP.

Methods from the field of data mining (DM) can also be used to visualize hyperparameter performance dependencies. Data mining aims to extract patterns from raw data using algorithms (Fayyad et al., 1996). A concrete approach by Cortez and Embrechts (2013) presents visualizations based on sensitivity analysis that open black-box models. Similar to PDP, sensitivity analysis uses methods to measure the impact on the output of a given model when inputs are varied. While in a PDP one or two parameters are varied by using a grid size and all other parameters are hold constant, the DM approach provides different methods for estimating the predictions. Moreover, the PDP and ALE plots only represent the average of the predictions, while the proposed methods also illustrate other statistics such as minimum, quantiles or other

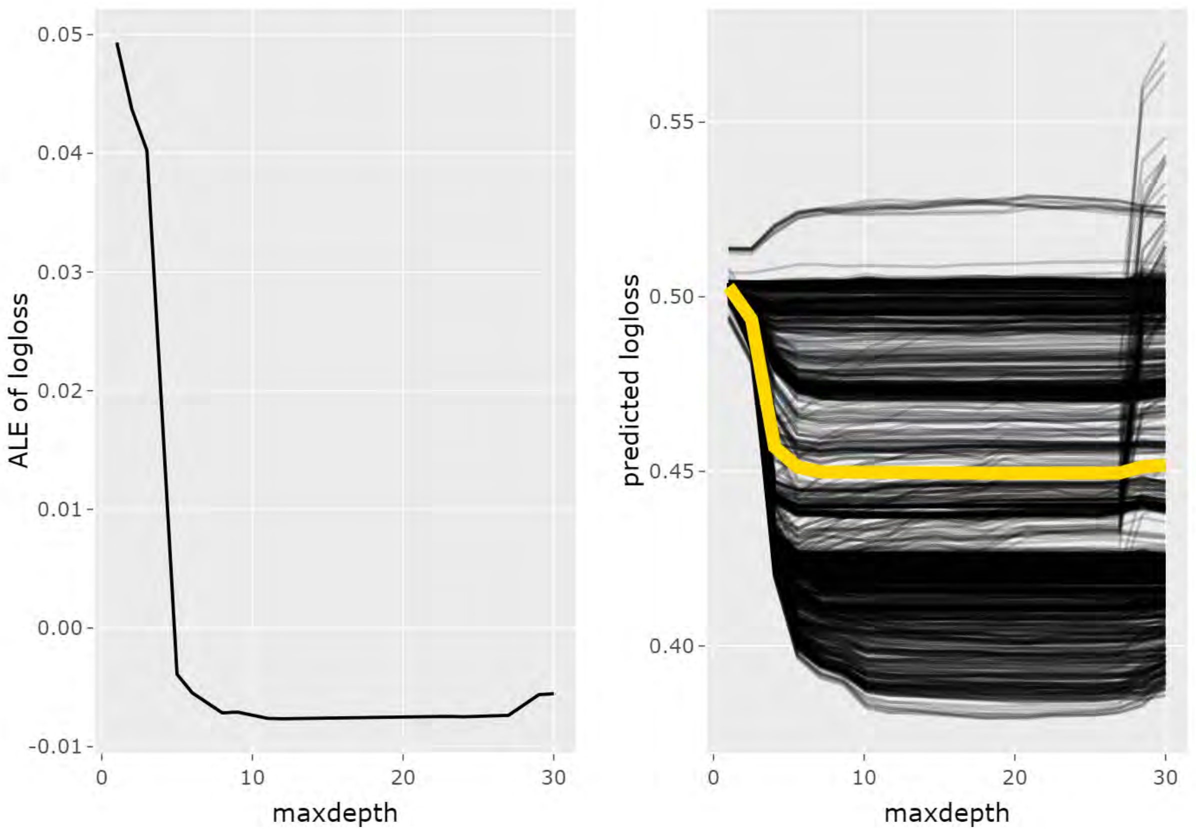


Figure 4: The left plot shows the ALE plot with centered predictions. In the right plot the PDP is displayed together with the ICE curves. Both plots show that values for the hyperparameter *maxdepth* of the iaml_rpart dataset should not be too low for a good average performance (low logloss).

sensitivity measures. Different visualization tools such as variable effect characteristic (VEC) curves, boxplots, VEC surfaces, and contour plots are proposed to represent the effect. If the parameters are normalized, multiple parameters can also be plotted in the same graph. Some of the proposed graphs for visualizations can be seen in the appendix in figure 16. The author has also implemented his approach in the R package `rminer`.

2.4 Parameter Importance

Parameter effect methods are very useful to analyze a few hyperparameters at the same time in more detail. However, often there are a lot of hyperparameters and most of the time not every hyperparameter has a significant impact on the performance. For this reason, it is often worthwhile to examine the importance of the parameters with so-called parameter importance methods, which return only one value per hyperparameter. With these methods, one can quickly get an overview of the most important parameters and obtain information about which hyperparameters should be further investigated and which can be neglected.

2.4.1 Permutation Parameter Importance Plot

Permutation provides the ability to swap values in a vector, and was used by Breiman (2001) to compute an importance measure for random forest using permuted parameters for the first time. Later, Fisher et al. (2019) introduced the importance of permutation parameters to calculate how important each parameter is for any predictive model. In the following, we also refer to the work of (Molnar, 2019). The idea behind this approach is to compare the model error $e_{orig} = \mathbb{E}L(\mathbf{y}, \hat{f}(\mathbf{X}))$ from a trained prediction model $\hat{f}$ with the model error $e_{switch} = \mathbb{E}L(\mathbf{y}, \hat{f}(\mathbf{X}_{switch}))$ from the same model but with permuted values. Swapping the values of a selected explanatory parameter cancels its effect, making the parameter completely uninformative for the target y while preserving its marginal distribution. The fraction or the difference of the two outcomes deliver a formula to measure hyperparameter importance:

$$I_i = \frac{e_{switch}}{e_{orig}} \quad or \quad I_i = e_{switch} - e_{orig}.$$

The importance I of the parameter $i \in P$ is high when the change in the model error is large and the importance of the parameter is low when the difference is close to zero. If the values of only one parameter in $\mathbf{X}_{Switch}$ are swapped, the result explains the importance of this parameter. If the values of more than one parameter are swapped, the result computes the importance of that subset of parameters. After the result is calculated for each parameter and subset, the parameters can be sorted by descending importance to allow comparison between parameters. It needs to be mentioned that for the loss function only non-negative functions, e.g. root mean square error, can be used.

It should also be noted that this method takes all interactions with other parameters into account. This is basically an advantage, since we thus know important parameters for sure, but it is also a disadvantage, since the interaction effects for both parameters are included in the performance measure (Molnar, 2019). In addition, we do not know how large the interaction effects are and where they occur. A further disadvantage of this tool is that the model error depends on the swapping of the parameter values. The result can vary greatly if the effect is calculated with a different permutation. Therefore, it makes sense to perform the permutation several times and average the results. While it increases the computation time, it also stabilizes the measure. Although, there are some limitations as already mentioned, the method can be used to estimate and interpret the importance of individual parameters very well and is therefore quite suitable for obtaining an overview of important parameters.

Often the results are presented in bar charts or just as points. Figure 5 shows a plot of the importance of parameters. In this case, it can be seen that *cp* is the most important and *minsplit* the least important parameter. The loss function used was mean absolute error (mae), and the model used was a random forest.

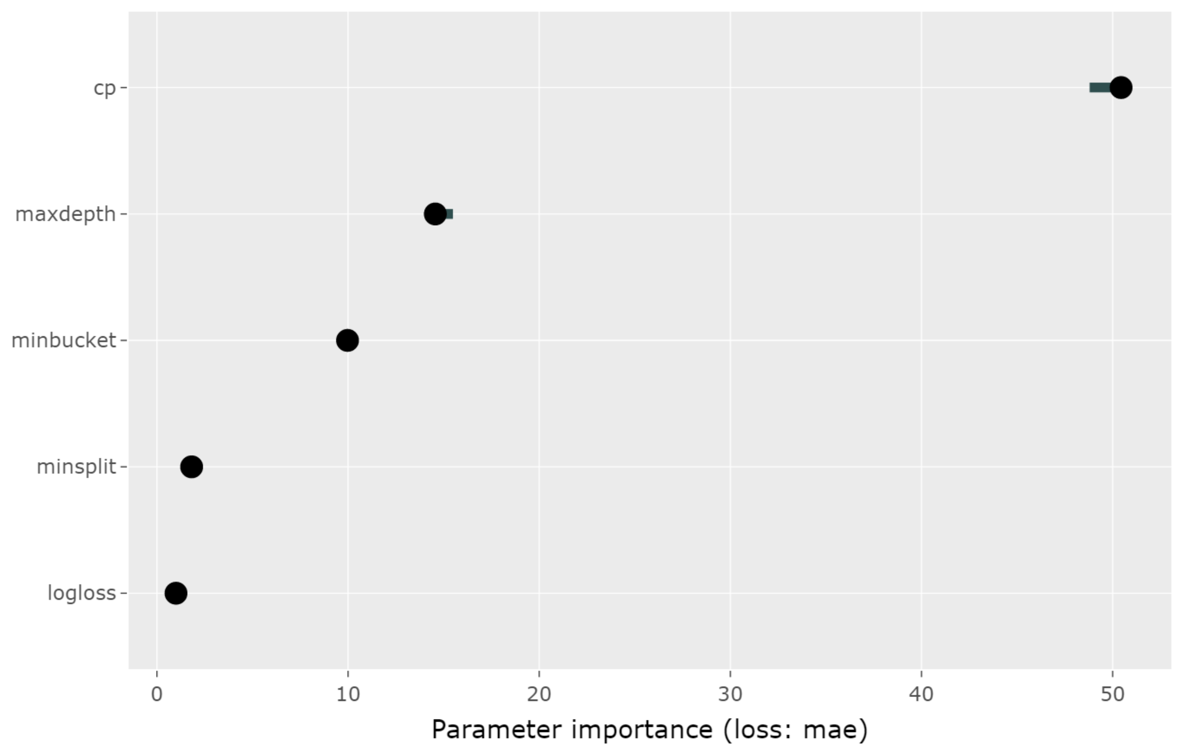


Figure 5: An importance plot for four hyperparameters. A higher value also means a higher importance of the parameter. It can be seen that *cp* is the most important and *minsplits* the least important hyperparameter for the data of the iaml_rpart dataset.

2.4.2 Functional ANOVA

The functional analysis of variance (fANOVA) is a technique that decomposes a high-dimensional function into additive components (Sobol, 1993). Based on the multi-index notation for the fANOVA proposed by (Hooker, 2004) and the marginal performance from section 2.3.2 a new tool was featured by Hutter et al. (2014). The tool allows to quantify the importance of an algorithms individual hyperparameter but also of interactions between two or more hyperparameters. By definition the model $\hat{f} : \mathcal{X}_1 \times ... \times \mathcal{X}_p \rightarrow \mathbb{R}$ can be decomposed into additive components that depend on subsets of hyperparameters P:

$$\hat{f}(\boldsymbol{x}) = \sum_{U \subseteq P} \hat{g}_U(\boldsymbol{x}_U).$$

Hutter et al. (2014) defines each component of the function $\hat{f}$ with the following formula:

$$\hat{g_U}(\boldsymbol{x}_U) = \begin{cases} \frac{1}{||\mathcal{X}||} \int \hat{f}(\boldsymbol{x}) d\boldsymbol{x} & \text{if } U = \emptyset, \\ \hat{a}_U(\boldsymbol{x}_U) - \sum_{W \subsetneq U} \hat{g}_W(\boldsymbol{x}_W) & \text{otherwise.} \end{cases}$$

If $U = \emptyset$, then the function mean is formed over the entire domain. If $|U| = 1$, the components are called main effect and capture the effect of changes of only one parameter. Specifically, it computes the importance of a hyperparameter by recording the effect of a change in this parameter, taking the average over all possible value of all other hyperparameters. If $|U| > 1$, all interactions between the hyperparameters in U are recorded, subtracted by low ordering interaction effects of $W \subsetneq S$. According to the definition of the fANOVA the variance for each subset can be calculated with

$$\mathbb{V}_U = \frac{1}{||\mathcal{X}_U||} \int \hat{g}_U(\boldsymbol{x}_U) d\boldsymbol{x}_U$$

and the total variance is the sum of the variances of all possible subsets

$$\mathbb{V} = \sum_{U \subset P} \mathbb{V}_U.$$

Finally, the importance of each subset of parameters can be calculated using the formula $\mathbb{F}_U = \mathbb{V}_U / \mathbb{V}$, which gives the fraction of the total variance for the parameters.

The biggest advantage of the fANOVA hyperparameter importance in comparison to the hyperparameter importance plot we introduced earlier is the possibility to calculate importance of interaction effects instead of only main effects. But by taking interactions into account, computational time is also greatly increased. The visual representation is just like the importance of the permutation parameters in a bar chart or simply as points in a coordinate system.

2.4.3 Local Parameter Importance

Another method to determine the importance of each parameter is the local parameter importance (LPI) which was introduced by Biedenkapp et al. (2018). With LPI the importance of the parameters is calculated based on a specific parameter configuration. This is particularly interesting for tuned parameters, as it allows the parameters responsible for the achieved performance to be clearly identified. For the LPI of a parameter $i \in P$, one needs an error metric e and an empirical performance model (e.g., random forests) that can predict the model error $\hat{e}$ for different parameter configurations. For the calculation itself, the variance of the estimated error values $\hat{e}$ of the different parameter values x_i is computed and divided by all variances. The formula for the LPI is:

$$LPI(i|\boldsymbol{x}) = \frac{Var_{v \in \mathcal{X}_i} \hat{e}(\boldsymbol{x}[x_i = v])}{\sum_{i' \in P} Var_{w \in \mathcal{X}_{i'}} \hat{e}(\boldsymbol{x}[x_{i'} = w])}$$

The outcome for the LPI of the parameter i is depending on a specific parameter configuration $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$. In the numerator, the parameter i is varied within its parameter space $\mathcal{X}_i$, while the other parameters remain constant. In the denominator, the same is done for each parameter i' in the entire parameter space $\boldsymbol{\mathcal{X}}$ to obtain the variance caused by all parameters.

The local property of this method is also the biggest difference to the fANOVA and the Permutation importance plot. While the two other methods are interested in the general importance of a hyperparameter, the LPI calculates the importance for a specific parameter configuration. They still share similarities with the fANOVA, due to the decomposition of variance, and with the permuted parameter importance, due to the fact that this method also does not calculate interactions between parameters. While this method can be very interesting when trying to understand changes in the environment of a parameter configuration, its approach is less interesting for this work because we are not interested in a specific configuration of hyperparameters, but only want to understand the dependencies between configurations and performance measurements in general. In addition, the approach does not provide interaction values, so there is no advantage by using the local parameter importance instead of the permutation parameter importance plot.

2.4.4 Other Methods

A local method to explain the performance for a given hyperparameter configuration is a break-down plot (Staniak and Biecek, 2018). In this approach, the calculated performance is decomposed so that contributions can be assigned to the various hyperparameters. While this method is also local like the local importance plot, it differs in computation and provides a plot for visualizing the effects of the parameters. An example for a break-down plot can be found in figure 17 in the appendix and further explanation of its interpretation can be found in the book of Staniak and Biecek (2018). On one hand it is advantageous that the importance of the parameters on the results can be seen and that the set configuration has a positive or negative effect on the result and its extent. But on the other hand the importance of the variables can be misleading if the interaction is large, since s break-down plot calculates the contributions based on the specified order of the parameters. For this problem, interaction-break-down, an alternative that can handle interactions but also takes more time, was proposed.

Most of these methods we introduced find application in the field of machine learning. Another field that analyze the dependencies between input and output for a similar purpose is the field of sensitivity analysis. Sensitivity analysis examines how uncertainty in the predicted output of a model can be attributed to various sources of uncertainty in the model inputs (Iooss and Lemaître, 2015). We have previously mentioned effect methods from the field of data mining that rely on sensitivity analysis, now we will directly address the field. The following discussion of alternative methods used in global sensitivity analysis is based on the paper by Iooss and Lemaître (2015).

In the global sensitivity analysis a variable selection is performed by the so-called screening to find unimportant parameters. Screening is performed under the assumption that many variables have only a small effect on the target variable. The target of these method is to reduce the number of input parameters to achieve less computationally intensive calculation methods as the ones mentioned in the following. A well known method in this field is the morris method (Morris, 1991). In this attempt not only input variables with negligible effects are categorized but also linear effects and non-linear effects with or without interaction effects. Even though the screening attempts to reduce the computationally intensity it is a fairly complex procedure and therefore still requires a considerable amount of computational capacity. For good results, the number of observations should therefore always be greater than the number of variables. After preprocessing, there are different importance measures based on linear methods, functional decomposition of variance (for non-linear and non-monotonic models) and other methods. For example, Sobol indices is a method which also make use of the fANOVA. Unlike the fANOVA suggested by Hutter et al. (2014), these methods do not use random forest but are applicable to the data itself via Monte Carlo sampling based methods. A disadvantage of these procedure is that these methods are often extremely costly, unstable and biased when the number of input parameter and number of instantiations increases. Therefore, some alternative methods with and without surrogate models are presented in this paper.

2.5 Interaction Detection

Functional ANOVA already provides a way to determine the importance of interaction effects between parameters. Another method to gain insight into interactions is the h-statistic (Friedman and Popescu, 2008). Similar to the interaction effects measured by functional ANOVA, the h-statistic measures how much the performance differences depend on the interaction of parameters. However, the h-statistic decomposes the partial dependence function, whereas the functional ANOVA decomposes a function that depends on the marginal prediction. Thus, the h-statistic does not measure the importance of the interaction terms, but rather the contribution of the interaction effect to the performance outcome. Unlike fANOVA, it is unknown whether the interaction effect has a high impact on the outcome. One way to combine the advantages of h-statistics and importance plots is proposed by Inglis et al. (2021). Accordingly, interactions and importance should be plotted together in a heatmap or a network diagram. However, Inglis et al. (2021) mentioned the computation of h-statistics is computationally intensive, as it requires the underlying calculations of the PDP.

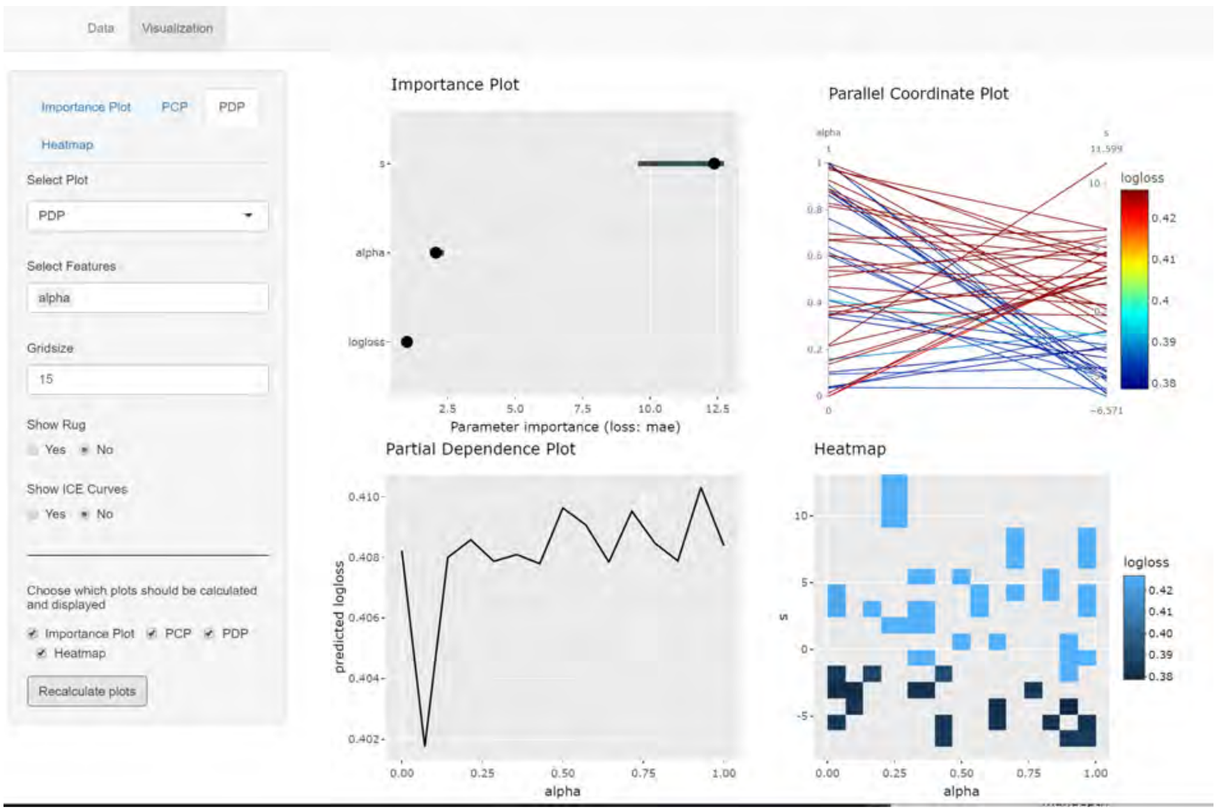
In another approach, interaction effects are searched via the variance, since weak variances of the individual parameters also indicate weak interactions. For this purpose, the importance of one parameter is calculated as a function of different points of the other parameters and the standard deviation is formed over the results (Greenwell et al., 2018). Alternatively, interactions can be computed using variable interaction networks from Hooker (2004), where the prediction function is split into main and interaction effects supported by the functional ANOVA.

3 Results

The goal of this work is to understand the relationship between hyperparameters and a selected performance measure using visualization methods. To this end, several visualization tools have been implemented in a R package and can be used through various callable functions or a Shiny application. In addition, an application study was conducted on two similar datasets to show what knowledge can be gained through the visualization tools. The results of the implementation and the application study are now presented.

3.1 Implementations in R and Shiny

There are several ways to visualize hyperparameters. For this work, the four visualization methods heatmap, parallel coordinate plot, partial dependence plot and permuted importance plot were implemented in a new R package called `VisHyp`. The R package is available on Github via the link https://github.com/Pizzaknoedel/visualize-hyperparameter. Furthermore, a Shiny application was implemented for easier use of the plots, which is also included in the R package. An example of the Shiny application including the different visualization methods can be seen in figure 6 below.



Select Plot: PDP

Select Features: alpha

Gridsize: 15

Show Rug: Yes [✓ No]

Show ICE Curves: Yes [✓ No]

Choose which plots should be calculated and displayed: [✓ Importance Plot] [✓ PCP] [✓ PDP] [✓ Heatmap]

Figure 6: An example of the Shiny application implemented in the `VisHyp` package which is able to visualize the four methods simultaneously and can be controlled by different settings.

3.2 Application Study of Two Similar Datasets

In the following the implemented methods are explained by a use case in which the two datasets smashy_lcbench and smashy_super were examined. Both datasets contain the same eleven hyperparameters, which are settings for a hyperparameter optimizer. The performance measure of the configurations is *yval*, which must be maximized to achieve a high performance. An overview of the parameters can be seen from table 2, for further details the work of Moosbauer et al. (2021a) is recommended. The application study of the two datasets aimed to understand the relationship between the hyperparameters and the performance measure to the extent that the parameter configuration spaces can be constrained while retaining the best configurations with the highest performance values.

Table 2: An overview of all hyperparameters including explanations, configuration ranges and scale. The table was partially adopted from Moosbauer et al. (2021a).

Parameter	Meaning	Range	Scale
sample	samples algorithm	$\{$`random`, `bohb`$\}$	
surrogate_learner	surrogate learner	$\{$`KNN1`, `KKNN7`, `TPE`, `RF`$\}$	
survival_fraction	survival rate	$[1, \infty)$	$1/survival\_fraction$
budget_log_step	fidelity rate	$[2^{1/4}, 2^4]$	$\log\log budget\_log\_step$
filter_algorithm	SAMPLE method	$\{$TOURNAMENT, PROGRESSIVE$\}$	
filter_select_per_tournament	filter sample per tournament	$\{1, \ldots, 10\}$	$\log filter\_select\_per\_tournament$
filter_factor_first	filtering rate of first point in batch at $t = 0$	$[1, 1000]$	$\log filter\_factor\_first$
filter_factor_last	filtering rate of first point in batch at $t = 1$	$[1, 1000]$	$\log filter\_factor\_last$
filter_with_max_budget	surrogate prediction always with maximum r	$\{$`TRUE`, `FALSE`$\}$	
random_interleave_fraction	random interleave fraction	$[0, 1]$	
random_interleave_random	random interleave the same number in every batch	$\{$`TRUE`, `FALSE`$\}$	

The results of the evaluation were visually verified by a parallel coordinate plot with parameter spaces constrained, respectively. In addition, mathematically calculated quality criteria were used to support the visual impression. In the following, the performance values from the datasets are given in set notation. The set of all occurring performance values in the datasets is denoted by Y_{all}. If only a subset of the performance values of the configurations is considered, the set is denoted by Y_{rem}. The set of the overall best 5% of all performance values are denoted Y_{best}. The following quality criteria are used for verification:

- The average performance (mean) of the considered configurations.
- The highest (max) and lowest (min) performance of the considered configurations.
- The quality criterion shows which quantile can be reached with respect to all configurations if the optimization is performed only for the considered configurations (Y_{rem}). For our two datasets, the formula for a quantile can be written as follows:

$$quantile = \frac{|\{\mathrm{y} : \mathrm{y} \in \mathrm{Y}_{all} \wedge \mathrm{y} \leq max(\mathrm{Y}_{rem})\}|}{|\mathrm{Y}_{all}|}$$

- The proportion of the performance values of the remaining configurations (Y_{rem}) and the total number of performance values (Y_{all}) calculated with the following formula:

$$proportion = \frac{|\mathrm{Y}_{rem}|}{|\mathrm{Y}_{all}|}$$

- The number of remaining configurations (Y_{rem}) that belong to the best 5% of performance values (Y_{best}) divided by the remaining configurations to explain what fraction of the configurations considered are part of to the best:

$$top\ config = \frac{|(\mathrm{Y}_{rem} \cap \mathrm{Y}_{best})|}{|\mathrm{Y}_{rem}|}$$

Only the most important results of the analysis are discussed below. A detailed analysis of the individual hyperparameters can be found in the electronic appendix or via the Github repository. It should be noted that both the complete and filtered datasets were analyzed. For instance, the best configurations (top 20% of configurations with highest *yval* values) were examined and their range were restricted according to certain factors of parameters.

3.2.1 Dataset: smashy_lcbench

When analyzing the smashy_lcbench dataset, the first step was to select the most important parameters using the permuted importance plot. We used the mean absolute error (mae) as the distance measure. The importance plot result formed the basis for further investigation, as it indicated which parameters were primarily relevant and needed further investigation. Figure 7 shows the plot and it can be seen that the *sample* parameter is the most important parameter as it has the highest value. The most important findings of all parameters are summarized in the following.

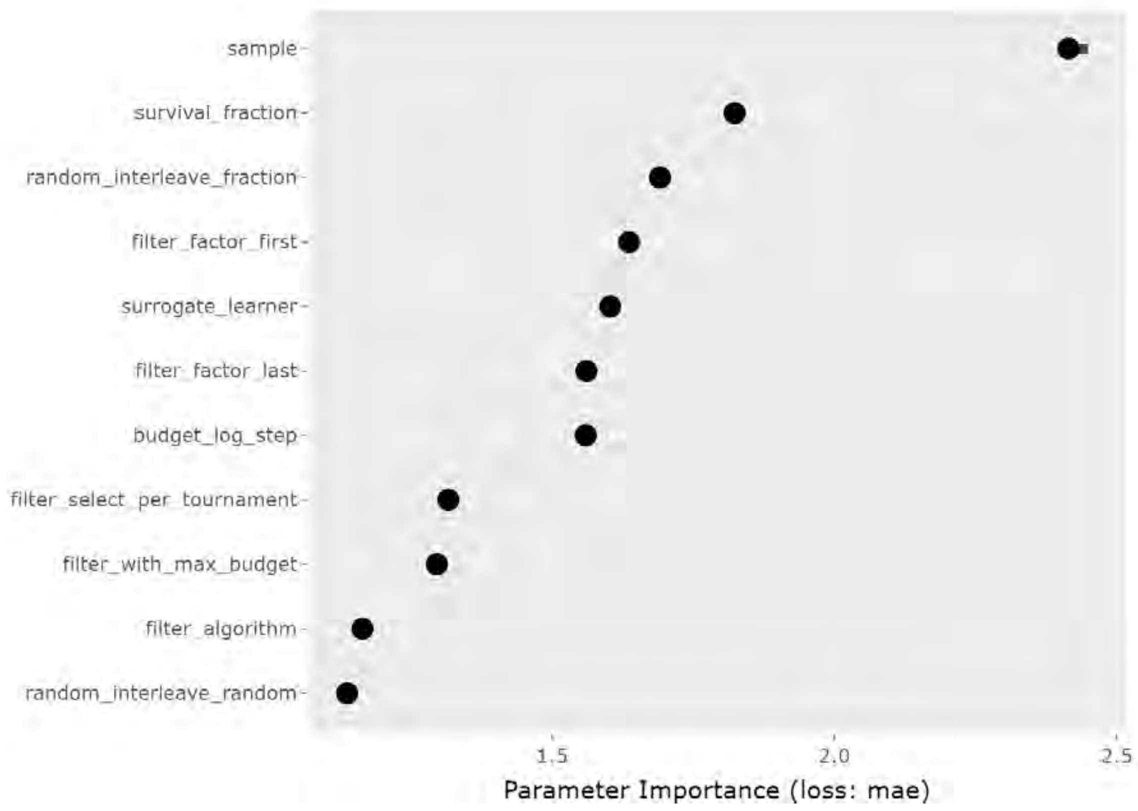


Figure 7: An importance plot for the smashy_lcbench dataset. The parameter *sample* has the highest value and therefore is the most important parameter.

sample: When examining the *sample* parameter, it was shown that the `bohb` factor should be preferred over the `random` factor, as it gave much better results on average. Moreover, by looking at the best 20% of the configurations, most of the remaining ones contained the `bohb` factor. Therefore, only the respective configurations with the factor `bohb` were used for the following investigations.

survival_fraction: The next important parameter was *survival_fraction*. It has been found that, on average, better performance was achieved with low values, but great performances could also be achieved with high values if the configuration of the other parameters were set correctly. The predicted performances can be seen in figure 8 by comparing the ICE curves with the PDP curve. While the average performance decreases with higher *survival_fraction* values, some ICE curves reach the same performances with high and low values. In addition, to set the parameter correctly, the *surrogate_learner* parameter should be restricted first, since its values cause large differences in the PDP.

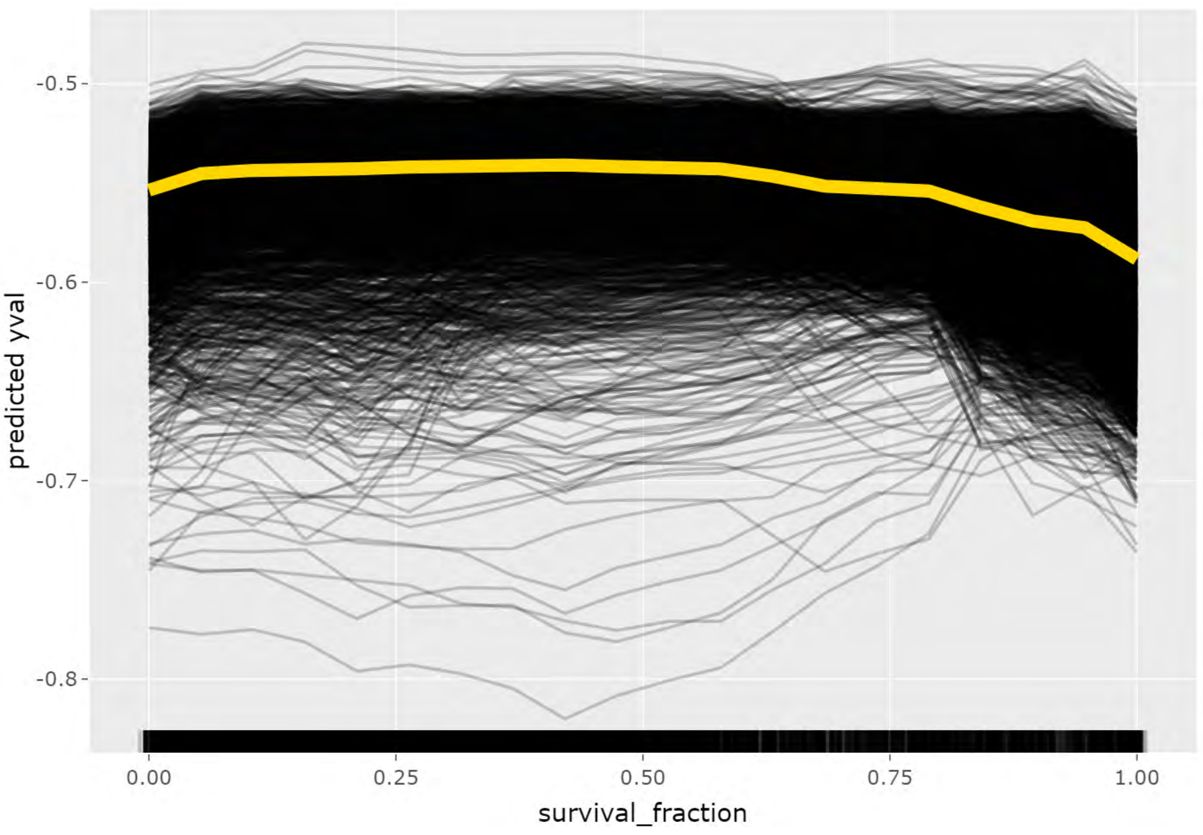


Figure 8: A PDP with ICE curves for the *survival_fraction* parameter of the dataset smashy_lcbench. The PDP shows that configurations with low values achieve the best performance on average. The ICE curves illustrate that high values can also lead to great performance values if the other parameters are set properly.

surrogate_learner: Generally, the parameter *surrogate_learner* was not as important, but it had various influences on the other parameters' performances and was therefore set first. The various influences were already mentioned for the parameter *survival_fraction*, but were also observed for other parameters like *random_interleave_fraction*. Figure 9 shows the different effects of the parameter *random_interleave_fraction* on the *yval* performance depending on the selected *surrogate_learner*.

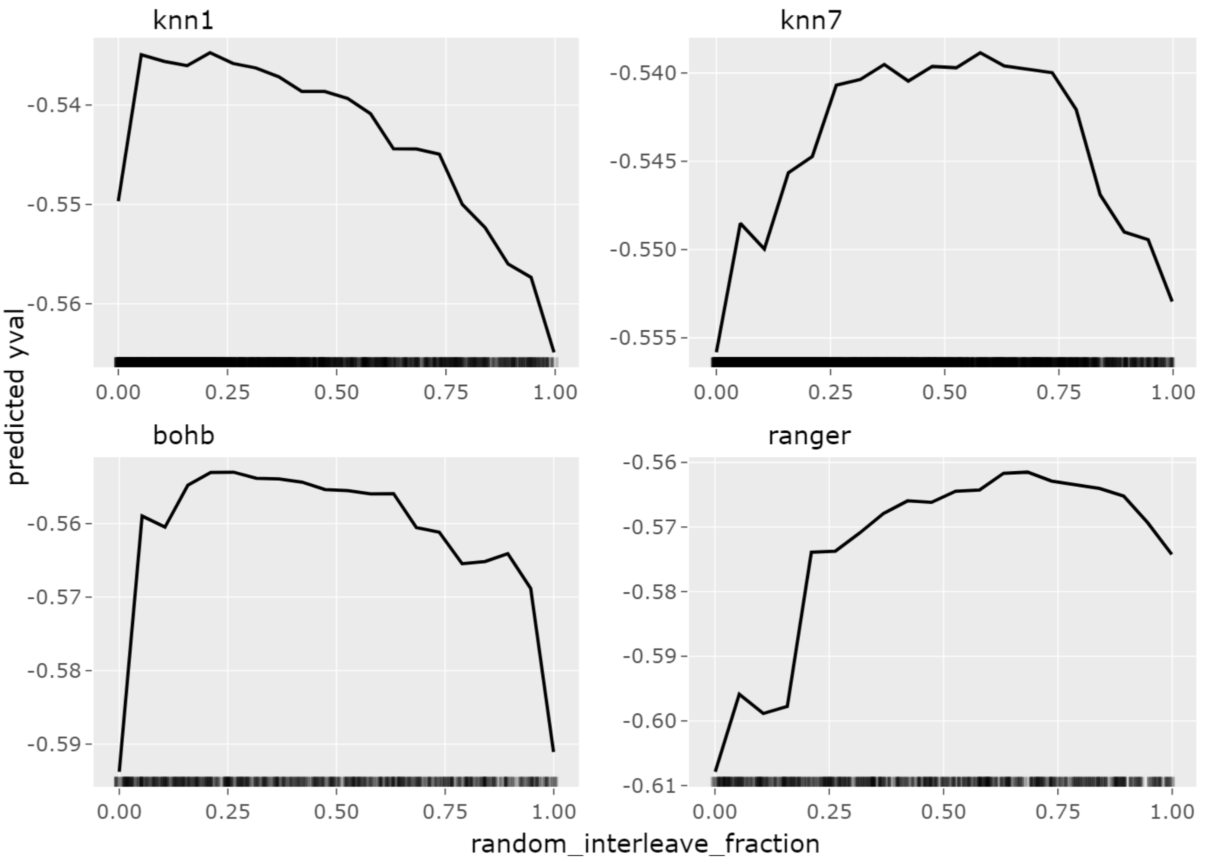


Figure 9: Different PDPs for the *random_interleave_fraction* parameter corresponding to the datasets constrained by the `knn1`, `knn7`, `bohblrn`, and `ranger` factors of the *surrogate_learner* parameter. It can be seen that the factors have different effects on the estimated performance.

budget_log_step: Another important parameter according to the importance plot was the *budget_log_step* parameter. So far, a high importance was an indication that the parameter had configuration ranges with high and low performances. However, for this parameter, we could not clearly specify a range in which the configurations gave better performance values on average. We found that the line in the PDP showed many small fluctuation, and concluded that this led to the increased metric value for the importance plot. Therefore, the parameter was not as important, so no restrictions were needed except for not setting the parameter too low, since the poorest performance values were mostly obtained at the beginning.

filter_with_max_budget: The parameter *filter_with_max_budget* was generally not important, but since it only has two factors it can easily be restricted. On average, the parameter led to a better performance for the factor `TRUE`. Nevertheless, it should be noted that combined with the parameter *surrogate_learner* and its factor `bohblrn` the effect was much more of importance than for its other factors.

Other parameters: The already mentioned hyperparameters were the most important. For other parameters like *filter_factor_first*, *filter_factor_last*, *filter_select_per_tournament* or *filter_algorithm* the impact and importance strongly depended on the settings of the other parameters and the desired goal. For example, if the best configurations for the factor `bohblrn` of the *surrogate_learner* parameter were of interest, the best combination was with `TRUE` for *filter_algorithm* and a value above5 for *filter_factor_last*, so that as many bad configurations as possible were discarded. However, these parameters did not require any constraints, since the most important restrictions have already been made with the other parameters. The remaining parameter *random_interleave_random* did not perform very well no matter what the situation of the settings was and could therefore be ignored completely. For a deeper understanding, the evaluation of the parameters can be read in the detailed analysis in the electronic appendix.

Finally, to verify the results, a PCP was created for all data and constrained according to the analysis. It should be noted that this was only one possible restriction of the parameter spaces and therefore other restrictions would have been possible as well. The restricted configuration ranges are shown as pink lines in figure 10. If no pink lines are visible, no parameter restriction is applied. It can be seen that the remaining configurations indeed achieve good performances. The exact parameter restrictions can be found in chapter 3.2.3 in the table 5 or in the more detailed analysis of the datasets.

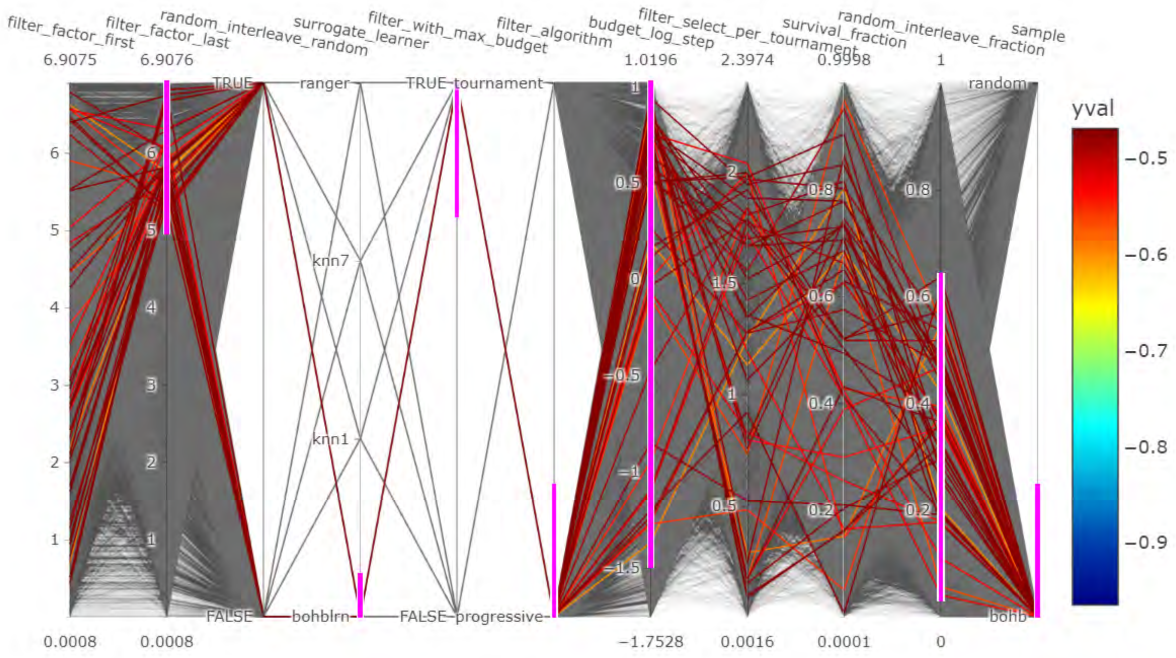


Figure 10: A PCP for the smashy_lcbench dataset. The applied constraints lead to configurations with mostly good performance values.

Additionally, the previously introduced quality criteria were used to confirm the visual impression. Here, the quality criteria were applied to the complete and the constrained dataset to allow a comparison. In table 3 it can be seen that 67% of the leftover configurations belong to the top 5% configurations of the entire dataset. Moreover, the 99.9% quantile is reached, which is the 5th best configuration. Also, the worst configuration (min) of the performance values of the remaining configurations is 0.6003, which is considerably better than the worst configuration of the whole dataset with the performance value of -0.9647.

Table 3: The evaluated quality criteria for the complete and constrained smashy_lcbench datasets. It can be seen if that the restricted range contains mostly good as well as top configurations values.

dataset	$\lvert y_{rem} \rvert$	mean	min	max	quantile	proportion	top config
smashy_lcbench	10712	-0.5646	-0.9647	-0.4690	1	1	0.05
chosen subset	49	-0.5026	-0.6003	-0.4713	0.9995	0.0046	0.67

3.2.2 Dataset: smashy_super

After examining the smashy_lcbench dataset, a further application study was performed with the smashy_super dataset to see if the results differed. First, an importance plot was plotted again to identify the most important parameters. As can be seen in figure 11 the parameter *surrogate_learner* is the most important and the two parameters *random_interleave_random* and *filter_with_max_budget* are the least important ones. In the following, the results of the parameters are again summarized individually, using the same order as in the previous chapter.

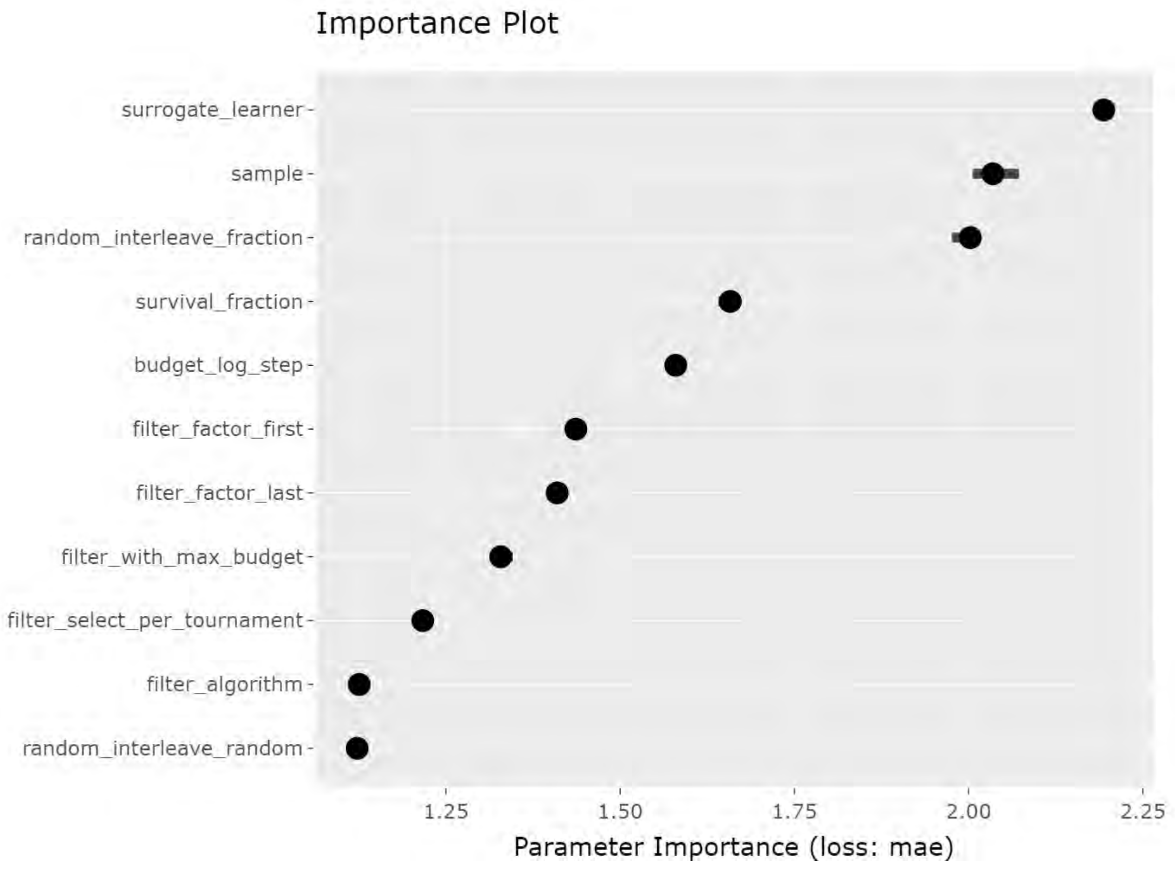


Figure 11: An importance plot for the smashy_super dataset, showing the *surrogate_learner* parameter as the most important parameter.

sample: On average, configurations of the *samples* parameter achieved a better performance with the `random` factor than with the `bohb` factor. In the best 20% of the configurations mostly `bohb` *samples* were sorted out, but those that remained had better performance on average than the `random` *samples*. Ultimately, both *samples* could result in good performance values, but since a lot of the remaining *samples* were `random`, this factor was chosen for our final configurations.

survival_fraction: In general, lower values performed better than higher values for the *survival_fraction* parameter. Regarding the top configurations, higher values did not seem to be worse, so with good configurations of the other parameters, the value of the *survival_fraction* parameter could also be high and still achieved greats performances. Although it could be shown that not all high values resulted in poor performances, lower values seemed to be the right choice, as the better configurations had mostly lower values and showed better performance on average. In figure 12 the results can be seen. For the interpretation of the PDPs, it should be noted that in the right plot, containing the best configurations, the remaining values are distributed differently as shown by the rug. Thus, while on average better performance is achieved with higher values, there are many more configurations with low values. In addition, the range of the y-axis should be taken into account, since the line in the right plot increases rapidly, but the range of the performance values only show small differences. For these reasons, low values are probably the better choice.

surrogate_learner: The *surrogate_learner* parameter was one of the most important parameters of the whole dataset. After reducing the dataset to the best 20% of the configurations, we could see that the parameter became less important, since the remaining *surrogate_learner* were mainly configurations with the factor `knn1`. Even though the best configurations of all other *Surrogate_learner* could achieve better *yval* values than the most configuration of `knn1`, `knn1` should be chosen in principle, since it gave better results on average and made up a clear majority of the best configurations.

random_interleave_fraction: Another very important parameter which was also the most important hyperparameter for the best configurations was the *random_interleave_fraction* parameter. In this case, the results were unambiguous, so higher values led to better results for both the complete dataset and the subset with the best configurations. This result could be shown with all of the effect tools used in this bachelor's thesis.

budget_log_step: A problem, similar to the problem of the *survivalfraction* parameter, occurred for the *budget_log_step* hyperparameter. Across the entire dataset, higher values were better on average, but in the best 20% of the configurations, lower values achieved better *yval* values. It should be noted, however, that unlike the *survival_fraction* parameter, the parameter values were more evenly distributed and therefore good performance values could be obtained with low and high values. In this case it was better not to limit the parameter.

filter_factor_first: If the factor `knn1` of the *surrogate_learner* parameter has been chosen,

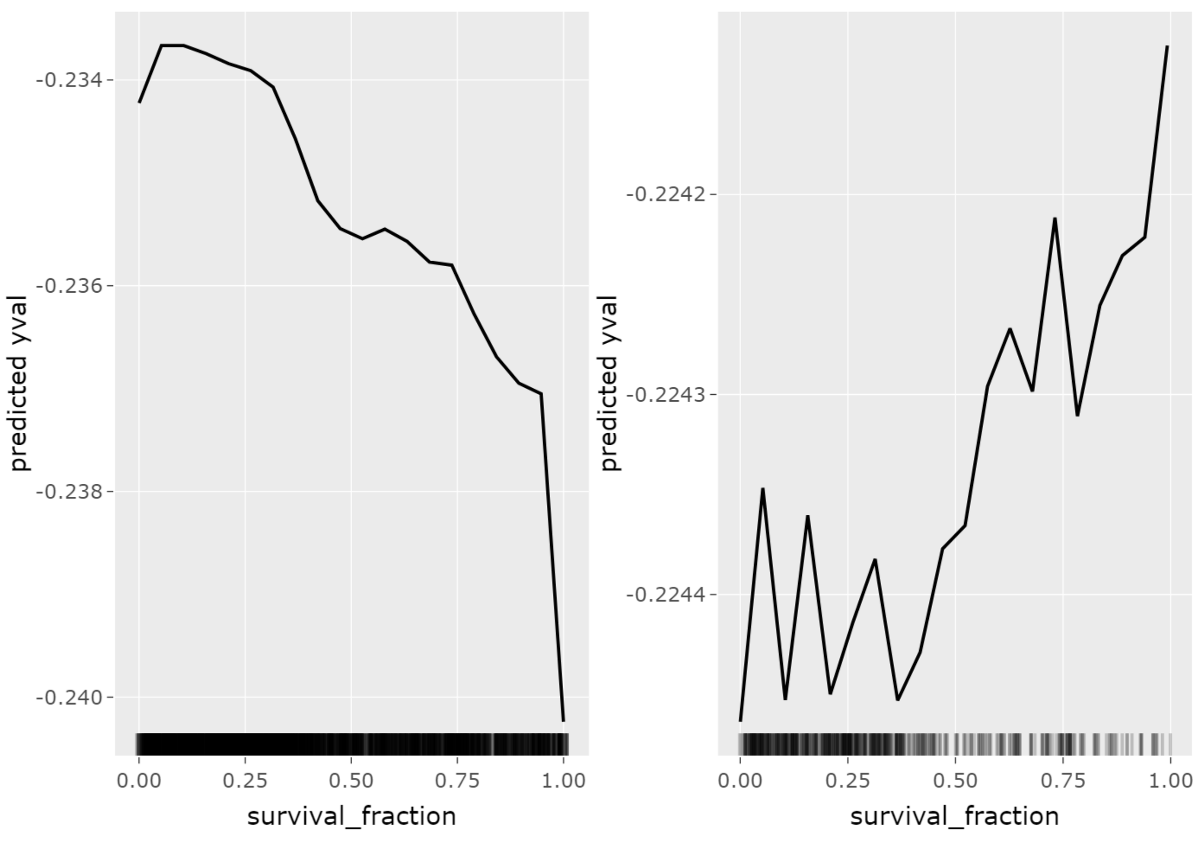


Figure 12: One PDP for the complete dataset (left) and one PDP for the best configurations (right) for the parameter *survival_fraction*. In the left plot, low values generally achieve better average performances. For the right plot low values are a better choice as well, since most of the top configurations have low values and the range of the predicted values is quite small.

the *filter_factor_first* parameter was the most important one. However, by looking at the complete dataset, this parameter was not important. There was also a difference between the ranges of the complete and the partitioned dataset. While values above 6 did not perform well for the complete dataset, they achieved the best results for the partitioned dataset. Since the majority of good values were above 4 after splitting the configurations into the best 20%, values above 4 seemed to be a reasonable choice.

filter_factor_last: The interpretation of *filter_factor_last* was a little more complicated. The hyperparameter showed large fluctuations and ranges of different prediction quality depending on whether the complete or the partial dataset was considered. Moreover, although the importance was high due to the large fluctuations, the range of predicted performances was not very large (and actually refutes the importance). In general, it could be concluded that the parameter value for *filter_factor_last* should preferably not be between 4 and 5.

filter_with_max_budget: Compared to the parameters mentioned above, the hyperparameter *filter_with_max_budget* was quite easy to interpret, but was not really important for the complete dataset, even though, considering the best configurations, the factor `TRUE` was the first choice in combination with knn1.

Other parameters: The parameters *filter_algorithm*, *filter_select_per_tournament* and *random_interleave_random* had little effect and were therefore not constrained.

The results were reviewed again by using a PCP with restricted configuration ranges. It was found that almost exclusively performances above the average were obtained, whereas most of them were even great performances. Nevertheless, not all top configurations could be found. The result of the PCP is visualized in figure 13. The exact applied restrictions of the values for the parameters can be looked up in the following chapter in table 5 or in the electronic appendix.

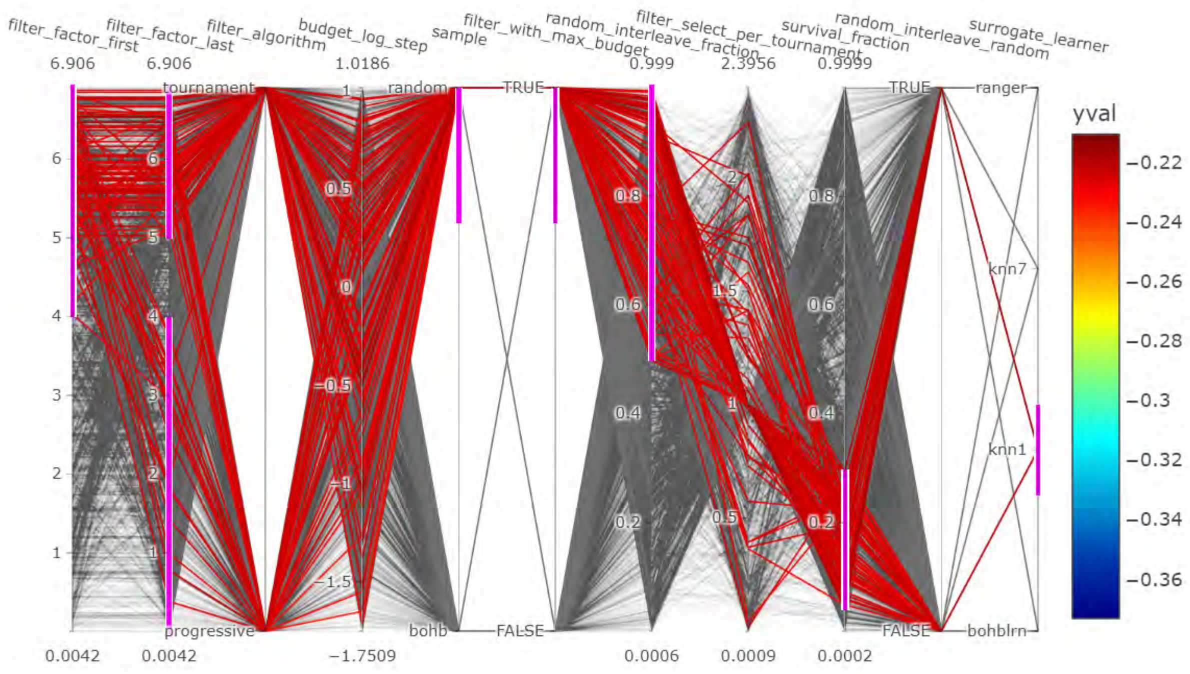


Figure 13: A PCP for the smashy_super dataset. The applied constraints lead to configurations with mostly good performance values.

Lastly, the result of the dataset with the restricted parameter ranges was expressed in various quality criteria and compared to the complete dataset. The results can be seen in table 4. It is striking that only 13% of the remaining configurations are top configurations. This is mainly due to the definition of the quality criterion, as configurations are only considered top configurations if they belong to the best 5% of all configurations. If a percentage of 20% is selected, the quality criterion would take on a value of 67%. In addition, the maximum performance value of the constrained dataset has a value of -0.2196, which corresponds to the 99% quantile of the complete dataset and shows that a pretty good configuration is included. The worst performance in the restricted dataset with -0.2335 is also much better than the worst performance value of -0.3732 concerning the entire dataset.

Table 4: The evaluated quality criteria for the complete and constrained smashy_super datasets. It can be seen that the restricted range contains many good configurations and also achieves top values.

dataset	number of y_{rem}	mean	min	max	quantile	proportion	top config
smashy_super	2845	-0.2347	-0.3732	-0.2105	1	1	0.05
chosen Subset	96	-0.2260	-0.2335	-0.2196	0.9933	0.0337	0.13

3.2.3 Comparison of Results

In the previous subsections the parameters were investigated for both datasets. The goal was to restrict the configuration areas to achieve a smaller subset with including mostly good performance values. The chosen restrictions can be seen in table 5. The results are displayed under retransformation in order to compare them optimally with the original scaled values.

Table 5: The table contains all hyperparameters of the dataset including suggested configuration restrictions. If no changes were made, a minus sign is used.

Parameter	Range in Dataset	smashy_lcbench: Restriction	smashy_super: Restriction
sample	{random, bohb}	{bohb}	{random}
surrogate_learner	{KNN1, KKNN7, BOHBLRN, RF}	{BOHBLRN}	{KNN1}
survival_fraction	$[1,\infty)$	-	$[3,20]$
budget_log_step	$[2^{1/4},2^{4}]$	$[2^{1/3},2^{4}]$	-
filter_algorithm	{TOURNAMENT, PROGRESSIVE}	{PROGRESSIVE}	-
filter_select_per_tournament	$]0,10]$	-	-
filter_factor_first	$[1,1000]$	-	$]55,1000)$
filter_factor_last	$[1,1000]$	$[148,1000)$	$]0,55)$ & $]148,1000)$
filter_with_max_budget	{TRUE, FALSE}	{TRUE}	{TRUE}
random_interleave_fraction	$[0,1]$	$[0.05,0.65]$	$[0.5,1]$
random_interleave_random	{TRUE, FALSE}	-	-

While this table above only provides a suggested set of configuration constraints, a more general comparison is made below, to highlight the similarities and differences of the datasets smashy_lcbench and smashy_super.

Similarities: Fundamentally, the importance of the individual parameters in the datasets were very similar. Thus, *sample*, *survival_fraction*, *random_interleave_fraction* were among the most important parameters for both datasets according to the importance plot. The parameter *surrogate_learner* was also quite important. In the smashy_lcbench dataset the choice of the *surrogate_learner* factor was crucial for the performances of the different settings made. In the smashy_super dataset the parameter was the most important according to the importance plot. However, the importance of the less important parameters were also similar. Even though the factor `bohblrn` was chosen for the smashy_lcbench dataset, due to the fact that, on average, the best performance values were obtained and for the smashy_super dataset most good results were attained with the factor `knn1`. High performance could be achieved for both datasets with all factors of the *surrogate_learner* parameter. The result for the *survival_fraction* parameter was similar for both datasets. Generally, lower values seem to result in better performances. Nevertheless, if the right configurations are used for the other parameters, high performances can also be achieved with high values. Furthermore, other parameters showed similarities as well. For the parameter *filter_factor_first* most good results were obtained for values above 4 and the parameter *filter_with_max_budget* should always be set to `TRUE`. The parameters *filter_algorithm*, *filter_select_per_tournament*, and *random_interleave_random* had little effect on performance in either dataset and can therefore be neglected.

Differences: For the smashy_lcbench dataset, the `bohb` factor of the *sample* parameter should always be chosen, since it was involved in almost all good results. On the contrary, for the smashy_super dataset most of the high performances were obtained with the `random` factor, but good results were achieved with `bohb` factors as well. Additionally, for the smashy_lcbench dataset, the performance of the parameter depended on the *surrogate_learner*, while for the smashy_super dataset, higher values generally led to better results.

4 Discussion and Conclusion

In this bachelor's thesis, we introduced a new R package called `VisHyp` to visualize the achieved performance of an algorithm in dependency of hyperparameter configurations by using four different methods. The implemented visualization methods are partial dependence plots, importance plots, heatmaps and parallel coordinate plots and can be used via a Shiny application integrated in the `VisHyp` package. Furthermore, to provide a more general overview of visualization methods that can be used to analyze the performance dependencies of hyperparameters, several alternative methods have been discussed. Finally, the four main methods of this thesis were used to explore the hyperparameter space of two datasets and to suggest parameter configurations. Although, the analysis resulted in configurations with good performances, the methods reached their limits at certain points which are enumerated below:

- For the importance plot, the main effects of individual hyperparameters were calculated and sorted by importance, but the interaction effects of combined hyperparameters were not calculated at all. This problem can easily be solved using, for example, the functional ANOVA of Hutter et al. (2014), since it computes the interaction effects up to a desired order. Early detection of interaction terms through this method can be helpful, as searching for interaction effects through parameter effect methods can be tedious due to the myriad combinations of hyperparameters.

- During the analysis, it was found that PCP quickly reaches its limits with large datasets. For many observations, the results overlapped strongly, so that hardly any statements about good configurations could be made. With constraints of a few hyperparameter ranges, the number of observations can be limited, but it would also be helpful to be able to narrow down the performance values to highlight a subset of configurations.

- Finally, there is room for improvement in the implemented Shiny application. For example, it is not yet possible to save and comment on the results of the interactive analysis.

This would be necessary to make the results comprehensible, which was the reason to perform the analysis via R Markdown and not via the Shiny application. In addition, the methods that require a surrogate model can currently only be performed with the random forest and the application is limited to the four visualization methods mentioned. Both could be supplemented in the future. Finally, it would be conceivable to publish the Shiny application and to rely on extremely powerful servers in the background, since the calculation of these methods require a lot of computational time for large datasets on a local computer.

In this work, we have shown the relationship between hyperparameters and performances using different visualization tools without analyzing the algorithm used for the configurations. Selecting an appropriate algorithm is an important task, but can be difficult without additional knowledge, as running multiple algorithms can be very computationally intensive (Bischl et al., 2012). Exploratory landscape analysis is used to create knowledge about the optimization problem before executing algorithms, so that a suitable algorithm can be found based on the properties it contains (Mersmann et al., 2011). Moreover, the knowledge about the characteristics of the underlying optimization problem can not only be used to choose the algorithm, but could also contribute to a better understanding of the relationship between performance and hyperparameter settings. Therefore, this approach could be an alternative or a complement to the visualization methods and could be further investigated.

Nevertheless, the `VisHyp` package can be used for a deep analysis to support the understanding of the relationship between hyperparameters and performances. The effect of individual parameter values on the performance can be visually displayed and, for example, increase the confidence in the selected configuration. By calculating the importance of individual hyperparameters, attention can be paid to the important parameters when analyzing and configuring them. At the same time, individual parameters can be neglected due to the lack of importance, which can lead to time savings and a better overview during the evaluation. Descriptive analysis also provides quick insights into structures and is therefore a good complement.

References

D. W. Apley and J. Zhu. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(4):1059–1086, 2020.

A. Biedenkapp, J. Marben, M. Lindauer, and F. Hutter. Cave: Configuration assessment, visualization and evaluation. In *International Conference on Learning and Intelligent Optimization*, pages 115–130. Springer, 2018.

B. Bischl, O. Mersmann, H. Trautmann, and M. Preuß. Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 313–320, 2012.

B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, et al. Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. *arXiv preprint arXiv:2107.05847*, 2021.

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

W. Chang, J. Cheng, J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges. *shiny: Web Application Framework for R*, 2021. URL `https://CRAN.R-project.org/package=shiny`. R package version 1.7.1.

M. Claesen and B. De Moor. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*, 2015.

P. Cortez and M. J. Embrechts. Using sensitivity analysis and visualization techniques to open black box data mining models. *Information Sciences*, 225:1–17, 2013.

G. Dzemyda, O. Kurasova, and J. Zilinskas. Multidimensional data visualization. *Methods and applications series: Springer optimization and its applications*, 75:122, 2013.

U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37–37, 1996.

M. Feurer and F. Hutter. Hyperparameter optimization. In *Automated machine learning*, pages 3–33. Springer, Cham, 2019.

A. Fisher, C. Rudin, and F. Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *J. Mach. Learn. Res.*, 20(177):1–81, 2019.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

J. H. Friedman and B. E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916 – 954, 2008. doi: 10.1214/07-AOAS148. URL `https://doi.org/10.1214/07-AOAS148`.

H. Gannett and F. Hewes. General summary showing the rank of states by ratios 1880. *Scribner's statistical atlas of the United States showing by graphic methods their present condition and their political, social and industrial development, Hewes FW,(Ed.). United States. Census Office*, page 151, 1883.

A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *journal of Computational and Graphical Statistics*, 24(1):44–65, 2015.

B. M. Greenwell, B. C. Boehmke, and A. J. McCarthy. A simple and effective model-based variable importance measure. *arXiv preprint arXiv:1805.04755*, 2018.

S. Hamid, A. Derstroff, S. Klemm, Q. Q. Ngo, X. Jiang, and L. Linsen. Visual Ensemble Analysis to Study the Influence of Hyper-parameters on Training Deep Neural Networks. In D. Archambault, I. Nabney, and J. Peltonen, editors, *Machine Learning Methods in Visualisation for Big Data*. The Eurographics Association, 2019. ISBN 978-3-03868-089-5. doi: 10.2312/mlvis.20191160.

J. Helton and J. P. Kleijnen. Statistical analyses of scatterplots to identify important factors in large-scale simulations, 2. robustness of techniques. Technical report, Sandia National Labs., Albuquerque, NM (US); Sandia National Labs ..., 1999.

G. Hooker. Discovering additive structure in black box functions. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 575–580, 2004.

F. Hutter, H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *Proc. of ICML-14*, 2014. To appear.

A. Inglis, A. Parnell, and C. B. Hurley. Visualizing variable importance and variable interaction effects in machine learning models. *Journal of Computational and Graphical Statistics*, (just-accepted):1–26, 2021.

B. Iooss and P. Lemaître. A review on global sensitivity analysis methods. In *Uncertainty management in simulation-optimization of complex systems*, pages 101–122. Springer, 2015.

H. Joo, C. Bao, I. Sen, F. Huang, and L. Battle. Guided hyperparameter tuning through visualization and inference. *arXiv preprint arXiv:2105.11516*, 2021.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

T. Loua. *Atlas statistique de la population de Paris*. J. Dejey & cie, 1873.

O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 829–836, 2011.

C. Molnar. *Interpretable Machine Learning*. 2019.

C. Molnar, B. Bischl, and G. Casalicchio. iml: An r package for interpretable machine learning. *JOSS*, 3(26):786, 2018. doi: 10.21105/joss.00786. URL `https://joss.theoj.org/papers/10.21105/joss.00786`.

J. Moosbauer, M. Binder, L. Schneider, F. Pfisterer, M. Becker, M. Lang, L. Kotthoff, and B. Bischl. Automated benchmark-driven design and explanation of hyperparameter optimizers. *arXiv preprint arXiv:2111.14756*, 2021a.

J. Moosbauer, J. Herbinger, G. Casalicchio, M. Lindauer, and B. Bischl. Towards explaining hyperparameter optimization via partial dependence plots. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021b.

M. D. Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991.

H. Park, Y. Nam, J.-H. Kim, and J. Choo. Hypertendril: Visual analytics for user-driven hyperparameter optimization of deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1407–1416, 2020.

F. Pianosi, K. Beven, J. Freer, J. W. Hall, J. Rougier, D. B. Stephenson, and T. Wagener. Sensitivity analysis of environmental models: A systematic review with practical workflow. *Environmental Modelling & Software*, 79:214–232, 2016.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

I. M. Sobol. Sensitivity analysis for non-linear mathematical models. *Mathematical modelling and computational experiment*, 1:407–414, 1993.

M. Staniak and P. Biecek. Explanations of Model Predictions with live and breakDown Packages. *The R Journal*, 10(2):395–409, 2018. doi: 10.32614/RJ-2018-072. URL `https://doi.org/10.32614/RJ-2018-072`.

R. C. Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL `https://www.R-project.org`.

Q. Wang, Y. Ming, Z. Jin, Q. Shen, D. Liu, M. J. Smith, K. Veeramachaneni, and H. Qu. Atmseer: Increasing transparency and controllability in automated machine learning. In

Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, pages 1–12, 2019.

L. Yang and A. Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

List of Figures

List of Tables

Appendix

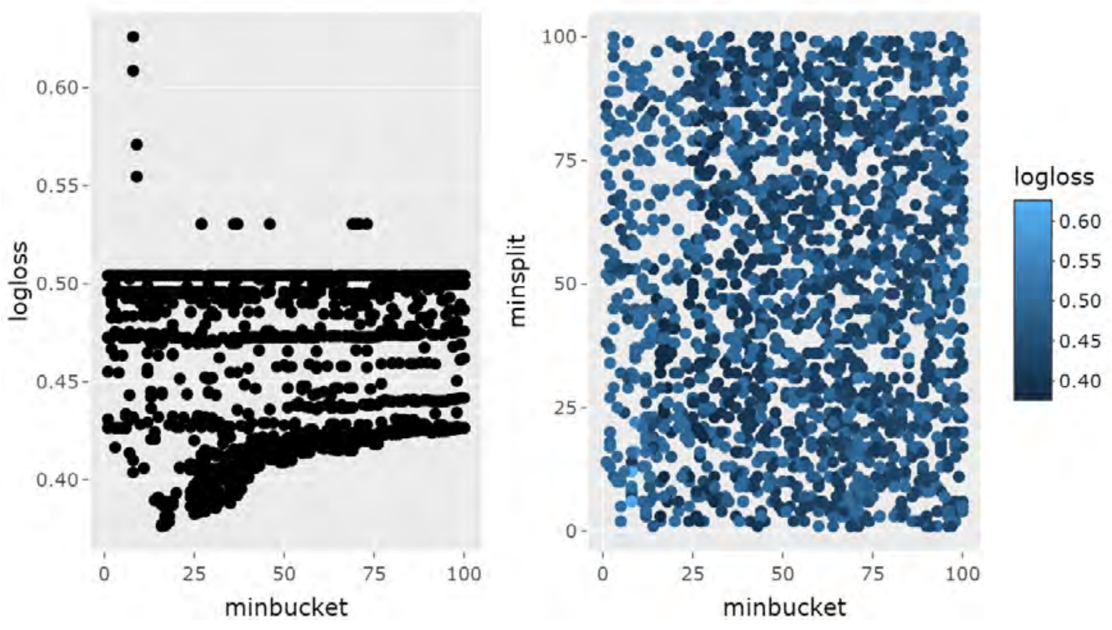


Figure 14: A scatterplot (left) and a colored scatterplot (right). In the left plot, only the parameter *minbucket* is plotted against the performance *logloss*. The value of the parameter should not be too low to achieve better performance. In the right plot, the two parameters *minbucket* and *minsplot* are displayed and colored according to the performance values. Because of the high number of dots the graphic is too cluttered. The data is available from the iaml_rpart dataset.

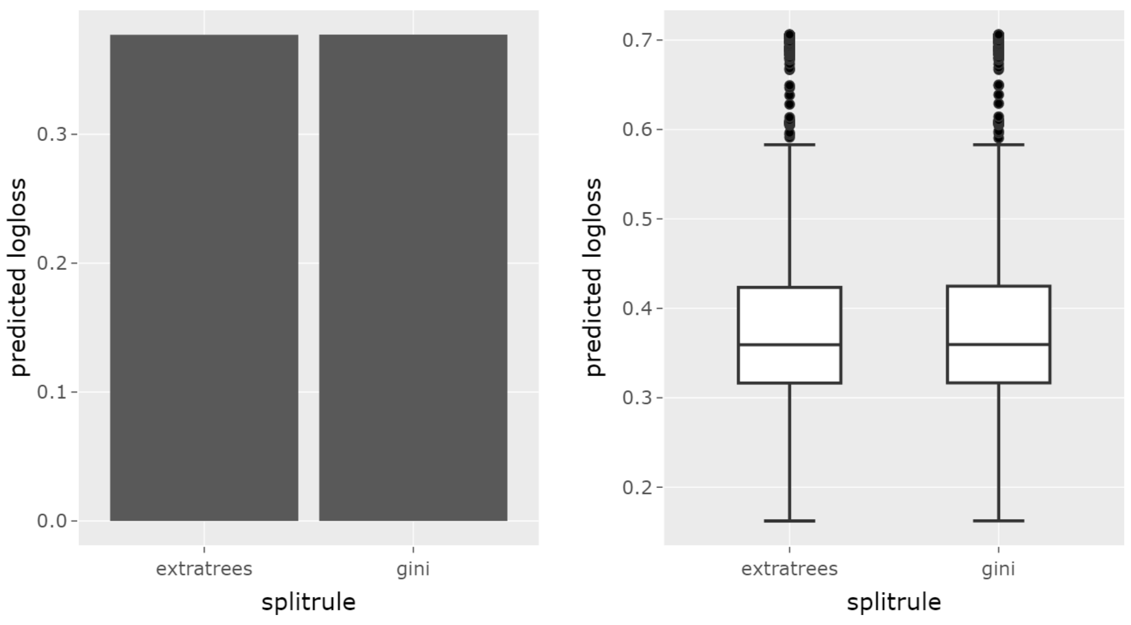


Figure 15: Two PDP for the categorical parameter "splitrule" of the iaml_ranger dataset. The left plot shows the result of the classical PDP. The right plot additionally shows the points calculated by the ICE method. It can be seen that both factors have an almost identical effect on the performance.

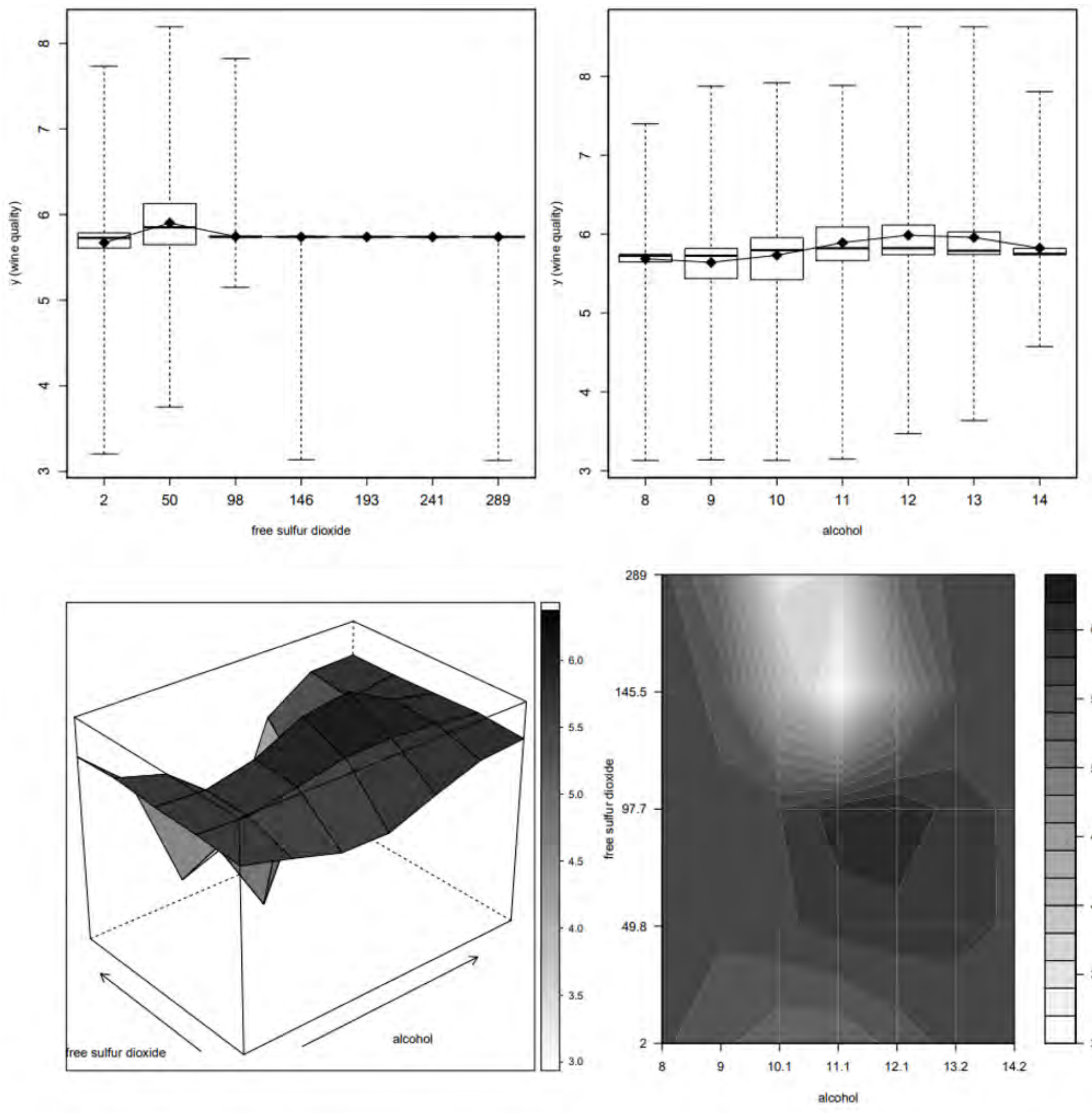


Figure 16: Various visualization techniques and their graphics proposed by Cortez and Embrechts (2013) to represent parameter effects.

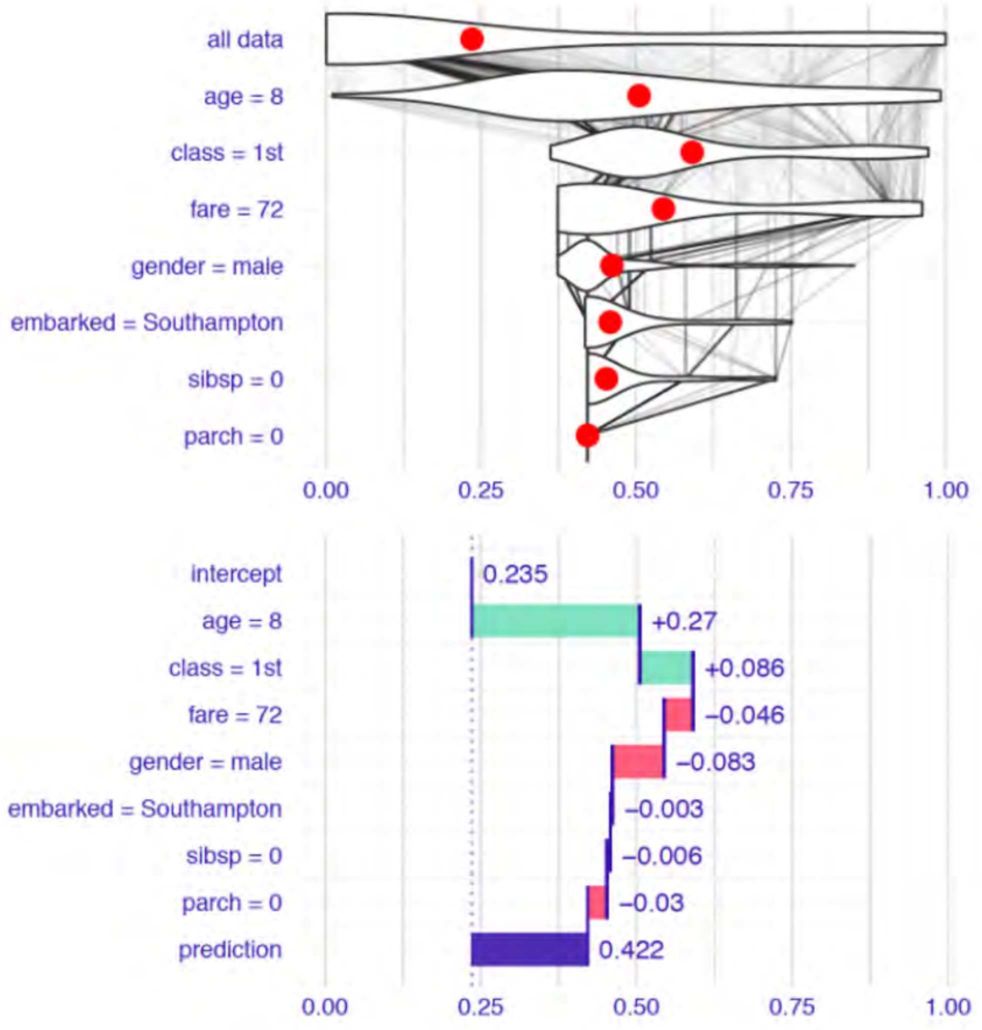


Figure 17: A break-down plot for the local importance of features. It can be seen that only *age* = 8 and *class* = 1 have a positive effect while all other feature settings have a negative effect on the result. The method and the example are from Staniak and Biecek (2018).

Declaration of Authenticity

The work contained in this thesis is original and has not been previously submitted for examination which has led to the award of degree. To the best of my knowledge and belief, this thesis contains no material previously published or written by another person excepts where due reference is made.

Simon Pradel
Munich, February 9, 2022