University of Passau

Department of Informatics and Mathematics

Bachelor Thesis

Meta-Learning for Performance Prediction on Configurable Software Systems

Author:

Bastian Fleischmann

July 24, 2018

Advisors:

Dr.-Ing. Sven Apel

Chair of Software Engineering I

Alexander Grebhahn

Chair of Software Engineering I

Fleischmann, Bastian:
Meta-Learning for Performance Prediction on Configurable Software Systems
Bachelor Thesis, University of Passau, 2018.

Abstract

Finding the performance-optimal configuration for configurable software systems is an omnipresent problem in both development and usage of software systems.
In order to identify the optimal configuration, machine-learning techniques can be used. But machine-learning techniques theirselves also provide a large set of options, each having a significant influence on the prediction accuracy and learning effort.
Here, we analyze the influence of machine-learning parameters on the prediction accuracy and learning effort.
To this end, we performed experiments on several configurable real-world case studies using four different machine-learning approaches. Using these results, performance models are created and compared in order to analyze the influence of parameters.
We could identify the distinct influences of machine-learning parameters and observe a relationship between increasing accuracy and effort. Further, we find that only a small subset of machine-learning parameters has the strongest impact on the accuracy and effort, while the remaining parameters are less relevant.

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

Many modern software systems are designed in a configurable way to provide a flexible solution that can be used in multiple domains and for a large number of application scenarios. In those systems the user or developer can decide which functionality has to be included and used in a configuration. One of the most well-known representative of those configurable software systems is the Linux Kernel with over 10000 configuration options [LSB+10]. Therefore, a vast amount of decisions can be made to meet specific functionality requirements. Besides of having an influence on the functionality of the system, these configuration options also have an influence on non-functional properties, such as memory consumption or performance. As a consequence, configuring such software systems is not only a matter of fulfilling functional requirements, both, the functionality and non-functional properties, have to be considered.

Optimizing software to maximize its performance is a challenging task for developers and users alike [XJF+15] as the default configurations often only provides an unsatisfactory performance for a large number of use cases. A naive solution for this problem could be measuring all possible configurations. However, for many systems the possible configurations, that can be chosen, in such software systems is often very large. For instance the prominent Linux Kernel theoretically provides more than 2^{10000} combinatorial possibilities. With such an enormous number of possible configurations, an exhaustive search for the best performing configuration is often not achievable due to time and resource limitations. Another possible solution for this problem would be using domain knowledge provided by the developer. However, using domain knowledge about the influences of configuration options to find the performance-optimal configuration is not always possible either, because it might not be available or the number of configuration options and complexity might be so big that there are unknown interactions affecting the performance. A third possibility would be only measuring a small subset of the configuration space and using these to predict the performance of the configurations that were not measured.

Many techniques have been developed such as Classification and Regression Trees (CART) [BFSO84], Random Forests [Bre01], Support Vector Machines [HDO+98] and Stepwise Regression [SGAK15]. However, a new problem arises when using ma-

chine learning. Many machine-learning techniques and libraries are designed to be applicable in many domains and scenarios. Therefore, they are also configurable by providing an extensive set of parameters, which can have influence on the prediction accuracy [FMS16]. This brings us back to the original problem as insight on the influence of the parameters is required in order to choose the optimal performing configuration.
Previous research indicates that using the optimal parameters can increase the prediction accuracy significantly, but is rarely done [FMS16]. As machine-learning techniques are also configurable systems, finding their optimal parameter setting might not be feasible when performing an exhaustive search. Further, while more refined search methods like Random Search [BB12], that approximate the optimal setting, are provided by these techniques, they only provide a compromise between search effort and improvement with no acquired knowledge. However, when well-tuned, the prediction accuracy of techniques can be vastly different compared to their performance with bad parameters [FMS16]. This leads to the accuracy of performance prediction also being dependent on how well the technique was tuned. Therefore, machine-learning techniques have to be treated as configurable systems and analyzed to achieve optimal results in performance prediction. To this end, we perform an empirical study to identify the influences of machine-learning parameters on prediction accuracy and learning effort.

1.1 Goal of this Thesis

The main goal of this thesis is to understand the influence and relevancy of machine-learning parameters on the prediction accuracy and learning effort to be able to make sophisticated judgments about the parameter and algorithm selection. As a result we analyze the importance of parameter-tuning. To this end, we want to answer the following research questions.

For the first research question we want to investigate how the machine-learning parameters impact the accuracy and effort, to understand how the performance of the machine-learning techniques can be increased.
`RQ1`: What is the influence of the different machine-learning parameters on prediction accuracy and effort?

Since machine-learning techniques provide extensive sets of machine-learning parameters, optimizing all parameters can be connected to high costs. To this end, we examine if subsets of parameters with higher priority for optimization exist to enable more efficient parameter-tuning and formulate the second research question as:
`RQ2`: Are some parameters more relevant for accuracy of performance predictions and learning effort than other parameters?

Since performance prediction can be performed on a multitude of configurable systems we want to explore similarities and differences in the relevancy of machine-learning parameters across multiple configurable systems. This is done to examine

if the results of our experiments can be applied to reduce the effort connected to finding the performance-optimal machine-learning parameters for performance predictions on different configurable systems.
RQ3: Does the relevance of the parameters depend on the case study?

1.2 Structure of the Thesis

In Chapter 2, we provide the basic knowledge and definitions required to understand this thesis. First, we start with introducing definitions of configurable software systems, metalearning and performance-influence models. After that, we provide an overview of four machine-learning techniques we consider in this thesis.
Our general approach for metalearning and the metrics we use are presented in Chapter 3. Additionally, *Star Plots*, the main method of visualizing our results, are explained in this chapter.
Afterwards, we describe our experimental setup in Chapter 4. Here, we also present the parameters we consider for metalearning and the case studies we use for learning.
We present the results of our experiments and evaluate them in Chapter 5. Here, we also discuss the results and answer our research questions.
In Chapter 6, we discuss threats to the validity of our results. We consider both internal threats that could stem from our experiments and external threats that could occur when applying our results.
After that, we examine work related to this thesis in the fields of performance prediction and metalearning in Chapter 7. Further, we compare them to our work and lay out differences.
We summarize this thesis and our findings and discuss the possibilities for future work that could be done on this topic with our results a basis in Chapter 8.

2. Background

In this chapter, we provide basic knowledge and describe the terms we use in this thesis.

First, we lay out a definitions and vocabulary for configurable software systems in Section 2.1. In Section 2.2, we provide an rough overview of metalearning. *Performance-influence models* are presented in Section 2.3. *Stepwise regression* - a machine-learning technique that produces *performance-influence models* - is shown in Section 2.4. The other machine-learning techniques *Classification and Regression Tress*(CARTs), *Random Forest* and *K-Nearest Neighbors Regression* are presented in Section 2.5, Section 2.6 and Section 2.7, respectively.

2.1 Configurable Software Systems

This section is based on the overview of configurable software systems provided by Apel et al. [ABKS16].
One of the most well known representative of configurable software systems is the Linux Kernel. When using configurable software systems such as the Linux Kernel, certain choices can be made either at compile or run time to customize the behavior of the software to customer needs. These choices are called *configuration options*. We differentiate between *binary options* O_{bin} and *numeric options* O_{num}. While only values of either 0 or 1 can be assigned to binary options, the values for numeric options can be chosen from a larger set of numbers. We define the set of all configuration options of a system as $\mathcal{O} = O_{bin} \cup O_{num}$.
Binary options can either be *optional* or *mandatory*. Mandatory options always have to be enabled, while optional options can also be disabled.
Besides, configurable software systems can have further constraints for the configuration process. Such constraints can be groups of required options or groups of alternative options, where only a single option can be selected. But constraints can also be any form of boolean expressions.
A valid assignment of each configuration option that fulfills all constraints, is called `configuration`. We refer to the set of all configurations as $\mathcal{C}$.

Further, a configuration $\mathcal{X}$ in this set will be denoted as the vector of values assigned to each configuration option $x_i \in \mathcal{O}$:

$$\mathcal{X} = (x_1, \ldots, x_n). \tag{2.1}$$

Such configurations can be used to run or compile the software system by setting the values of the configuration options, depending on the time of the configuration process. Those options then determine the functional behavior of the configurable software system.
Different configurations can also have different performance traits. An example for this could be enabling or disabling multithreading. Traits of configurations, that are not directly linked to the functionality of the software system and a metric for the performance or quality, are called *non-functional properties* or *nfp's*. Examples of such non-functional properties are memory consumption, execution time or energy consumption.

A set of performance measurements conducted on a configurable software system to measure the values of non-functional properties will be further referred to as $\mathcal{M}$. This set is mathematically defined as:

$$\mathcal{M} = \{(\mathcal{X}, y) | \mathcal{X} \in \mathcal{C} \ , \ y \in \mathbb{R}\}, \tag{2.2}$$

where y is the measured non-functional property value for the configuration $\mathcal{X}$.

2.2 Metalearning

Metalearning, in the context of machine learning, is a very ambiguous term, that includes multiple areas and definitions. Studying several definitions of metalearning, Lemke et al. [LBG15] defined metalearning on ground of those definitions as:
"'A metalearning system must include a learning subsystem, which adapts with experience. Experience is gained by exploiting metaknowledge extracted in a previous learning episode on single dataset, and/or from different domains or problems."'(Lemke et al. [LBG15])
So metalearning includes analyzing and learning on previous machine-learning results to improve the learning process.
Main areas of metalearning are combining or cascading multiple machine-learning techniques such as Bagging and Boosting, algorithm and parameter recommendation - often based on specific characteristics of the data set - and transferring learned knowledge [LBG15].
In the context of meta- and machine-learning, hyperparameters of the learning techniques - are free variables of the algorithm that have to be set by the user before performing the learning. These parameters can directly influence the behavior and prediction accuracy of the machine-learning technique.
Here, we focus on the aspect of parameter recommendation. In this context, metalearning is about learning and analyzing the influence and relevancy of machine-learning parameters for a machine-learning technique and different data sets by using

previous machine-learning results. As metrics to measure the quality of the previous learning results for a certain parameter setting the accuracy and learning effort can be used. This usage of metalearning aligns with how Vanschoren et al. [VBPH12] and Soares et al. [SBK04] approach metalearning.

2.3 Performance-Influence Models

We present the performance-influence models proposed by Siegmund et al. [SGAK15] as a way to represent the influence of configuration options on the values of non-functional properties in configurable software systems in this section.

Performance-influence models represent the influence of configuration options on non-functional properties in the form of a mathematical function. Such descriptive models open the possibility to infer the relevancy of each configuration option, the impact of the configuration options on the non-functional property and interactions between configuration options. With this knowledge, a developer or user can then make sophisticated decision during the configuration process with regard to the goal of optimizing a certain non-functional property or validate existing domain knowledge [GRS+17].
A performance-influence model takes a configuration $c \in \mathcal{C}$ as input. This configuration also serves as a function that maps configuration options to real numbers depending on the value, that was selected in the configuration. In case of binary options, the option $f \in O_{bin}$ results in $c(f) = 1$, if the option is enabled in this configuration, and otherwise in $c(f) = 0$. For a numeric option $n \in O_{num}$, the selected numeric value will be returned.
Mathematically, a performance-influence model Π is defined as:

$$\Pi(c) = \beta_0 + \sum_{i \in \mathcal{O}} \phi_i(c(i)) + \sum_{i \ldots j \in \mathcal{O}} \Phi_{i \ldots j}(c(i) \ldots c(j)). \tag{2.3}$$

In this performance-influence model, β_0 is the base influence that is present in every configuration. The individual influence of a single configuration option i on the non-functional property is $\phi_i(c(i))$. The ϕ_i function is a function, such as the identity function, a linear function, a logarithmic function or the square function, applied to the value of the option i. Logarithmic and quadratic influences are only useful for the set of numeric options O_{num} due to the value range of binary options.
An interaction between the options $i \ldots j$ is denoted as $\Phi_{i \ldots j}(c(i) \ldots c(j))$.
The value of an option of the configuration c is simply computed by inserting the configuration into the performance-influence model.

For example, a performance-influence model for a systems with the numeric configuration options *max_features* and *min_samples_leaf* and the alternative group of the two binary options *best* and *random* can look like this:

$$\Pi(c) = \overbrace{5}^{\beta_0} + \overbrace{3 * c(max\_features)}^{\phi_{max\_features}} + \overbrace{-10 * c(min\_samples\_leaf)}^{\phi_{min\_samples\_leaf}} + \underbrace{2 * c(best)}_{\phi_{best}} + \underbrace{-1 * c(max\_features^2)}_{\Phi_{max\_features,max\_features}} + \underbrace{-1 * c(random)}_{\phi_{random}}.$$

In this model, the base influence β_0 is 5, the influence of the option *max_features* $\phi_{max\_features}$ is 3, -10 for $\phi_{min\_samples\_leaf}$ and 2 for ϕ_{best}. Additionally, ϕ_{random} and the square of *max_features* $\Phi_{max\_features,max\_features}$ have a negative influence. The result of this performance-influence model is 8.9, when the model is evaluated for a configuration
$X = (max\_features, min\_samples\_leaf, best, random) = (1, 0.01, 1, 0)$.

2.4 Forward Feature Selection in Combination with Stepwise Regression

Stepwise regression can be used to create a mathematical model that describes the influences of independent variables on a dependent variable. In this section, we describe the stepwise regression approach by Siegmund et al. [SGAK15], which is used to create performance-influence models for configurable software systems.

The algorithm for this techniques is provided in Algorithm 2.1. The technique starts with taking a set of measurements $\mathcal{M}$, obtained by measuring a non-functional property, of a configurable software system, as input. The configurations and the respective non-functional property values of this set are used for learning, while the configuration options of the configurable software system are used as independent variables for the performance-influence model that will be derived.
Due to the number of possible interactions growing exponentially with the number of configuration options, this technique uses *forward feature selection* to derive the performance-influence model in an iterative way. With *forward feature selection*, the technique iteratively generates a set of candidates, as shown in line 7 of the algorithm, that might be added to the model, fits model(line 9) with with each candidate and selects the candidate that decreases the error rate most(line 11-13). If there is no candidate that decreases the error, no candidate will be added to the model as seen in line 16-19 of the algorithm.
Initially, candidates are generated from the set of configuration options $\mathcal{O}$ with functions applied to them. Those functions can, for example, be the linear function, the square function or a logarithmic function. If the model already contains elements, combinations of elements in the model and the initial candidates are also generated to produce interaction between configurations. This combination is done by multiplying both terms as shown in the performance-influence model in Section 2.3, where *max_features* is multiplied with itself.

The models are then fitted by using linear regression. To compute the error, the model is evaluated for every measurement in the learning set and the selected error metric is computed. The model with the best error is used as base for the next round.
This procedure continues until certain thresholds are reached. Such thresholds are a small improvement compared to the previous round or the error falling below a certain threshold. In this case the model is returned as the result. The thresholds are checked in line 20 of the algorithm.

Algorithm 2.1: Algorithm for the feature forward selection in combination with stepwise regression to create performance-influence models

Data: measurements, configuration options
Result: performance-influence **model**

```
1  modelError <- ∞;
2  model <- ∅;
3  repeat
4      bestCandidate <- NIL;
5      bestRound <- ∞;
6      previousError <- modelError;
7      candidates <- generateCandidates(model, configurationOptions);
8      forall candidate ∈ candidates do
9          fittedModel <- fitModel(model ∪ candidate, measurements);
10         currentError <- computeError(fittedModel, measurements);
11         if currentError < bestRound then
12             bestRound <- currentError;
13             bestCandidate <- candidate;
14         end
15     end
16     if bestCandidate ≠ NIL then
17         model <- model ∪ bestCandidate;
18         modelError <- bestRound;
19     end
20 until abortCondition(modelError, previousError, model);
```

2.5 CART

In this section, we provide an rough overview of Classification and Regression Trees (CARTs). This overview follows the initial definition of CARTs provided by Breiman et al. [BFSO84].

At their core, CARTs are binary (decision) trees, meaning that they are trees where every node either has two children nodes or is a leaf. Each non-terminal node splits the set of measurements at this node $M_{node} \subseteq \mathcal{M}$ by applying a split function that separates the data set into two parts by using a configuration option and one value of the value domain as splitting criterion. Such a split at a node and the corresponding configuration space is visualized in Figure 2.1. The splitting is performed recursively on each child until no more splits can be performed and a leaf node is reached. A leaf has a constant value, which is the predicted non-functional property

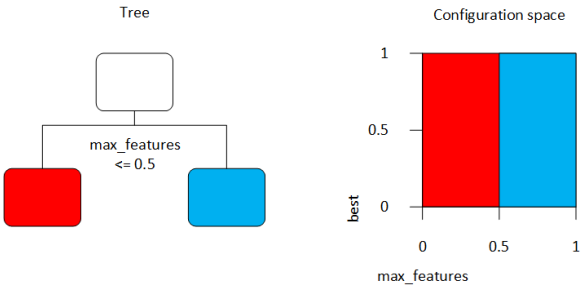


Figure 2.1: Visualization of the configuration space and the tree at a split.

value of the configuration, that fulfills the conditions to reach the leaf.

The construction of a tree starts with a measurement set $\mathcal{M}$. At the start, all elements of $\mathcal{M}$ are included in the root node. The predicted value $\hat{y}(t)$ of a leaf t with N_t measurements - in this case the root node with $|\mathcal{M}| = N_t$ measurements - is defined as the average the non-functional properties values of the measurements:

$$\hat{y}(t) = \frac{1}{N_t} \sum_{(\mathcal{X},y)\in t} y. \tag{2.4}$$

The error of a node t is the average squared difference between the measured and predicted values of all elements within this node:

$$R(t) = \frac{1}{N_t} \sum_{(\mathcal{X},y)\in t} (y - \hat{y}(t))^2.$$

The optimal split s^* at node t, that divides the elements at node t into two subsets with minimal error, is computed by

$$\Delta R(s^*, t) = \max_{s\in\mathcal{S}} \Delta R(s, t), \tag{2.5}$$

where $\mathcal{S}$ is the set of possible splits at the node t. As a split criterion, a value z for the configuration option $x_i \in \mathcal{O}$ and a less or less than condition is used. All measurements contained in the node where the value of the configuration option x_i fulfills the condition will be associated with the subtree t_L and otherwise they will be associated with the other subtree t_R. This split mechanism is then recursively done at the nodes t_L and t_R. For example, the criterion for the first split in Figure 2.2 is $max\_features < 0.88$.
This splitting of a node is performed until a stop criterion is reached. This stopping criterion is minimal improvement in learning error after a split. Additionally, the splitting stops if the number of elements $N(t)$ at the node t falls below a certain threshold. If a node fulfills one of these criteria, it will become a leaf and the average $\hat{y}(t)$ will become the predictive value of this node.
Once the tree is completely built, pruning can be performed by combining two nodes at a time to reduce overfitting. This is done until a subtree is found that is minimal respective to the both size and error of the tree.

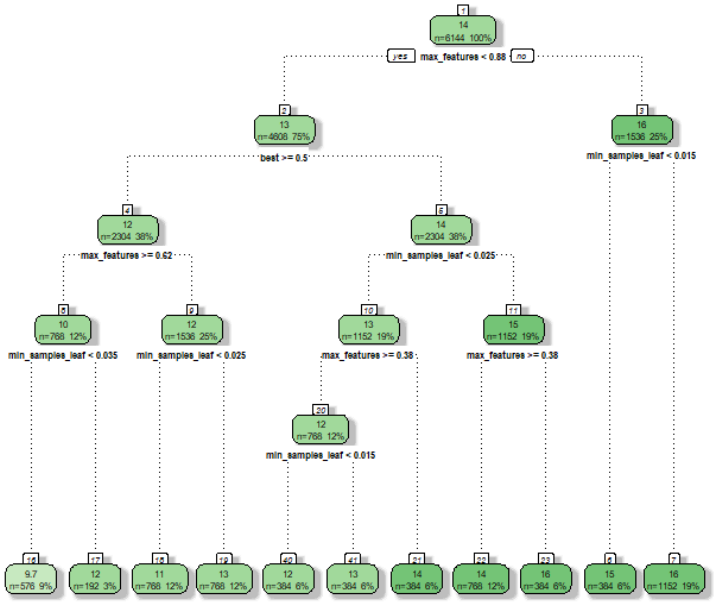


Figure 2.2: Example regression tree.

The value y_{new} of the configuration x_{new} is then predicted by traversing the tree according to the split criteria it fulfills.

For example, in Figure 2.2 a tree created by learning on the accuracy of the *CART* technique for different machine-learning parameters can be found. In this tree, a configuration with a value of 0.62 for the *max_features* numeric option, 0.025 for the *min_samples_leaf* numeric option and the *best* binary option disabled would fulfill all split criteria and reach the most left leaf. Then the predicted non-functional property would be 9.7.

2.6 Random Forest

In this section, we provide an overview of the *Random Forest* technique. As proposed by Breiman [Bre01], a Random Forest is an enhancement to Classification and Regression Trees. Instead of building one tree, a Random Forest creates multiple trees. For this a random influence on the selection of the learning set and the split criterion is included. To determine the final predicted value of the forest the predicted values of all trees are used.

When building a Random Forest with N trees from the measurement set $\mathcal{M}$, for each

tree τ_n, with $n \in [1, \ldots, N]$, a learning subset M_n is created by randomly selecting elements from $\mathcal{M}$. In this selection process, duplicate elements are not eliminated, therefore it is possible that a subset contains the same element multiple times.
An individual tree τ_n with the corresponding learning set M_n is then built in a way similar to Classification and Regression Trees. The function used to evaluate the accuracy is the mean squared error and the predicted value of a leaf in this tree is the average value of all elements in this node.
One of the differences to CART lies in which configuration options are available for a split. For each split, a subset of configuration options $O_n \subset \mathcal{O}$ is randomly selected. Only configuration options in O_n are available as split criteria for the respective split. The trees are then built until they reach the same stop criteria as Classification and Regression Trees, but there is no additional pruning of the finished tree.
To predict the non-functional property value of the configuration x_{new}, the value $\hat{y}_{t_n}(x_{new})$ has to be predicted by every tree τ_n. This is done by traversing the respective tree and returning the value of the leaf that was reached. The predicted value

$$\hat{y}_{Forest}(x_{new}) = \frac{1}{N} \sum_{n=1}^{N} \hat{y}_{t_n}(x_{new}) \tag{2.6}$$

of the Random Forest is then computed as the average of predicted values of the trees within the forest.

2.7 K-Nearest Neighbors

K-Nearest Neighbors(KNN) Regression is a technique that uses spatial information to predict new configurations and as a consequence uses similarity between configurations as a way for prediction. In contrast to the other techniques presented in this chapter, no model is created by this technique. Here, we follow the overview provided by Imandoust et al. [IB13].

Like in previous sections, the measurements set is denoted as $\mathcal{M}$. In K-Nearest Neighbors Regression, the position of the configurations, represented as points in the configurations space, relatively to each other is used to predict the performance. For this, distance metrics such as Euclidean, Manhattan or Chebychev distance can be used.
For prediction, a value for K has to be selected, which denotes the number of neighboring configurations that will be considered for the prediction process. In Figure 2.3, the $K = 3$ nearest - by Euclidean distance - neighbors of a new configuration(red) are indicated by green lines. The higher the K value is the more configurations will influence the prediction. The value y_{new} for a new configuration x_{new} is then calculated as:

$$y_{new} = \frac{1}{K} \sum_{i=1}^{K} y_i, \tag{2.7}$$

$y_1, \ldots, y_K$ are the non-functional property values of the K nearest configurations to the configuration x_{new} that has to be predicted.
This prediction does not take the distance to the nearest neighboring configuration

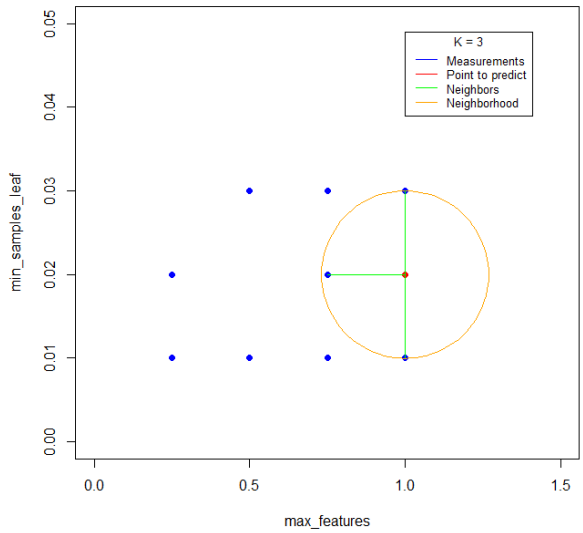


Figure 2.3: Nearest neighbors of a new configuration with the configuration options max_features and min_samples_leaf.

into consideration. Additionally, the distance to the neighboring configurations can be used as weight in order to distinguish between close and distant neighbors so that more similar configurations have more influence on the prediction of a new configuration. Such a distance metric can be any metric such as the examples given above. To this end, the prediction of y_{new} for x_{new}, when using a weight function $W(X_i, X_j)$, is

$$y_{new} = \sum_{i=1}^{K} W(x_{new}, X_i) y_i. \tag{2.8}$$

3. Methodology

In this chapter, we introduce the general approach used to answer our research questions. In Section 3.1, we provide a comprehensive description of the steps we perform to conduct our metalearning experiments. We lay out the metrics used for the actual metalearning process in Section 3.2 and introduce the visualizations in Section 3.3.

3.1 Metalearning Approach

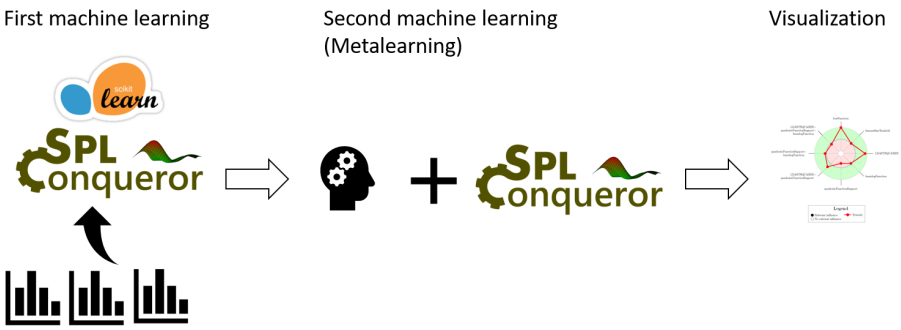


Figure 3.1: Concept of our metalearning approach.

For our metalearning approach we perform two levels of machine learning. The first level is performance prediction using performance data sets of configurable software, whereas the second level is the actual metalearning.

In the first step, we interpret the machine-learning technique as a configurable software system using a certain workload. The goal of this step is to measure the performance of a machine-learning technique for different machine-learning parameter settings. At this point, the different machine-learning parameter settings are the configurations of the machine-learning technique. For these configurations we measure the respective prediction accuracy and learning effort as the non-functional

properties of the technique. This is achieved by using data sets, retrieved from performance measurements on various configurable software systems, as workload. At this step, we measure every possible parameter setting by using the complete data set for learning and error calculation without any sampling.
We repeat this process for each combination of data set and machine-learning technique.

The second level of learning is the actual metalearning process. Here, we only apply the Stepwise Regression technique to create performance-influence models for the combinations of machine-learning techniques and workloads. The configuration options in these performance-influence models are then the machine-learning parameters of the respective technique. To achieve this, we use the measurement results produced in the previous step to generate models for the influence of parameters on the (1) accuracy and (2) learning effort. Those two models are constructed for every measurement set we obtained with the first level of learning. Just like during the previous step, the Stepwise Regression is applied without any sampling, but - unlike previously - we do not differentiate between the results of different parameter settings for the Stepwise Regression technique as we only use one fixed parameter setting.
After this stage we use the performance-influence models to deduce the influence of parameters on the on accuracy and effort and answer our research questions.

3.2 Metrics

Here, we describe the two main dependent variables we consider in the first part of our learning procedure.

As a metric for learning effort we use the clock time the machine-learning technique takes to derive a model from the learning set. This takes only time between beginning and the end of the model training process into account. To this end, the effort does not include any additional time required for prediction, initial setup of the technique and transformation of the learning set.
To measure the error as an indicator for accuracy, we compute the average relative error of the produced model. For this, we use the machine-learning technique to predict the respective non-functional property for all configurations in the data set that was used for learning. The error is then defined as:

$$Error = \frac{100}{|\mathcal{M}|} \sum_{C \in \mathcal{M}} \frac{\hat{y_C} - y_C}{y_C} \tag{3.1}$$

3.3 Visualization

In this section, we introduce the method of visualization we use to show our results and answer our research questions.

To visualize performance-influence models, we use so called *Star Plots*. As an example performance-influence model to explain the visualization, we use the performance-influence model:

$$\Pi(c) = 5 + 3 * c(max\_features) + -10 * c(min\_samples\_leaf) \\ +2 * c(best) + -1 * c(max\_features^2) + -1 * c(random).$$

A description of the domain of the configuration options and the constraints between them can be found in Figure 3.2.

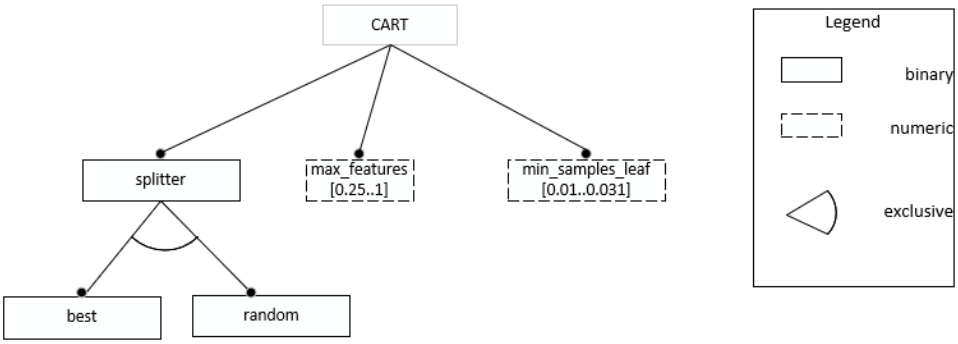


Figure 3.2: Description of the configuration options.

The base influence and influences of mandatory configuration options are neglected in this method of visualization as they can not be altered by the user.
All influences are displayed relatively to the highest absolute influence in the performance-influence model. For numeric configuration options the influence is also weighted by the value domain of the configuration option before determining the highest influence. In our example performance-influence model, *max_features* has the highest influences as the influence of *min_samples_leaf* is weighted by a factor of 0.031.
A simplified Star Plot for the performance-influence model shown in this section can

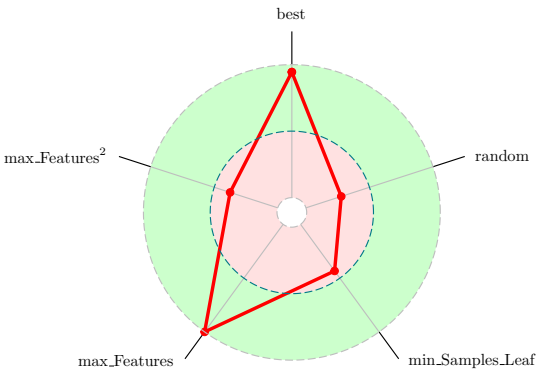


Figure 3.3: Star plot for the performance-influence model presented in this section.

be found in Figure 3.3. Star plots consist of three circles with the inner most circle denoting a relative influence on the performance metric of -1 and the outer most circle denoting a relative influence of 1. The middle circle stands for no influence.

Subsequently, the green area between the outer and the middle circle represents a relative influences between 1 and 0, while the red area represents a relative influences of 0 to −1.
For our metrics for effort and accuracy this means that influences in the green area degrade performance while influences in the red area indicate a performance improvement as error or time decreases.
All circles of the star plots are divided into several axis, where each axis denotes a influence of a configuration option or an interaction among multiple configuration options on the performance. The configuration or interaction is stated by the label of that axis, while the extent of the influence is presented by a dot in within the circles.

For visualization purpose, the axes of configuration options in alternative groups

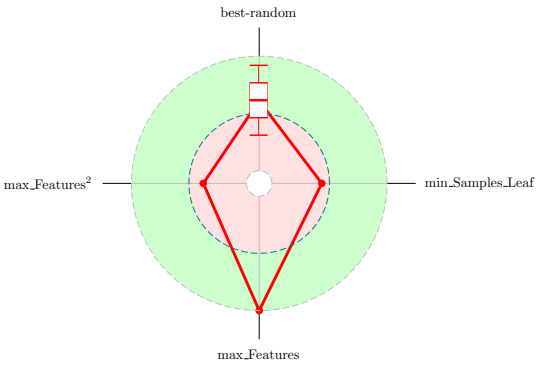


Figure 3.4: Figure 3.3 with boxplot for alternative groups.

- such as *random* and *best* in Figure 3.4- are combined to one single axis and the dot is replaced by a box plot showing the different influences within the group. The label for such a group shows the names of the configuration options ordered by the magnitude of their individual influence - starting with the highest. In case of numeric configuration options - additionally to the dot, which denotes the mean influence when taking all possible value assignments into account - a violin plot is displayed showing the values of the influence weighted by the different assignments of the numeric option. To understand how the assignment of the options changes the influence, the point where the options have their minimum value is indicated by a cross. Likewise, the point with the maximum value is indicated by a circle.

To better comprehend the influence of options, the performance models is displayed

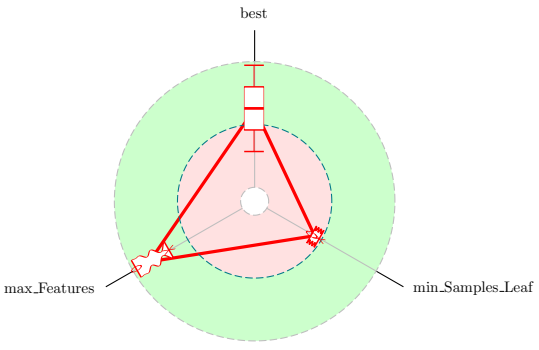


Figure 3.5: Figure 3.4 with violin plots for polynomials and numeric options.

in a more coarse granular way. Therefore, polynomials of single options and interactions are combined to a single axis if they appear multiple times in the model. All possible values that a polynomial can have are then shown in the same way as influences of numeric options. In Figure 3.5, a example for this is the combination of *max_features* and $max\_features^2$ to *max_features*.

Further, Star Plots can be used to display the average of multiple performance-influence models. The meaning for the plot stays the same, however each influence is the average across all performance-influence models that are included in that plot.

4. Experiment Setup

In this chapter, we provide the specifications of the experiments we conducted. The hardware we use to run experiments is listed in Section 4.1. Additionally, the software environment, including the machine-learning software we apply, is presented in Section 4.2. We lay out the data sets used for creating models and prediction in Section 4.3. In Section 4.4, we present the parameters we consider in the metalearning on each machine-learning technique.

4.1 Hardware Setup

We performed our experiments on two different clusters depending on the memory consumption of the machine-learning technique. The experiments on CART, Random Forest and K-Nearest Neighbors regression were performed on a cluster, which consists of nodes with Intel Core i7-4790 processors and nodes with Intel Core i5-4590 processors. For our experiments, only the nodes of this cluster with Intel Core i7-4790 processors and 16GB of available RAM were used to assure consistent results. Due to memory restrictions, the experiments for Stepwise regression were executed on a different cluster, which consists of a single node with two Intel Xeon E5-2609v2 processors and 128GB of available RAM.

To ensure consistent and replicable results, the dynamic hardware performance enhancements *Intel Turbo Boost* and *Intel Hyper-Threading* were disabled on both clusters. Additionally, all experiments on measuring the learning effort at the first level of metalearning were repeated three times.

4.2 Machine-learning software

In this section, we provide information about the machine-learning software and the software environment we used in our experiments.
For our experiments on machine-learning techniques we used the machine-learning

software SPLConqueror[1]. This software was executed on a Mono 4.2.1 environment using the Ubuntu 16.04 operating system.
SPLConqueror was used as a stand-alone machine-learning software for our experiments on the Stepwise Regression technique, while the interface to the scikit-learn[2] machine-learning API provided by SPLConqueror was used for the experiments on the Classification and Regression Trees, Random Forest and K-Nearest Neighbors techniques. For the experiments on scikit-learn a modified version SPLConqueror was used as we encountered bugs in the Mono environment. In this modified version, some default data structures that were not required for these particular experiments were removed or replaced as a underlying implementation caused segmentation faults.
Version 0.19.0 of scikit-learn was used for these experiments. Since scikit-learn is a API written for python, we utilized a python 3.5 virtual environment to execute all python code and learn with scikit-learn.
To perform metalearning on the results we obtain from using previous machine-learning, we also used the SPLConqueror system to train models and extract information about learning time and prediction accuracy.

4.3 Configurable Software Data Sets

For our experiments we use five data sets obtained from measuring non-functional properties of real world configurable software systems for various configurations. Data sets described in this section are used to fit the machine-learning software presented in Section 4.2 and for prediction and error calculation, while for both procedures all configurations in the data set are used.

The data sets of the *AJStats*[3], *Trimesh*, *Dune*[4], *VP9*[5] and *x264*[6] configurable software systems are used for learning.
AJStats is a tool for statistical analysis of aspect oriented code written with AspectJ. Configuration options configure which statistics are collected. For example, statistics about the occurrences of Classes, Interfaces, Methods and Aspects can be collected. The non-functional property, that was measured for the data set of this tool, is the time the analysis needed.
Trimesh and *Dune* are multigrid methods. Dune provides algorithms to solve Poisson equations. The exact algorithms and parameters that are used to solve the equations can be configured for this software. Similar to Dune, Trimesh solves partial differential equations using triangular grids. It can be configured in how smoothing is performed and which algorithm is used to solve the equations. For Trimesh the average time per iteration of the solving process was measured, while the time to solution was measured for Dune.
x264 is a video encoding software that encodes videos with the H264 codec. For

[1]https://github.com/se-passau/SPLConqueror - last visited on 2018-07-22
[2]http://scikit-learn.org/stable/index.html - last visited on 2018-07-22
[3]http://wwwiti.cs.uni-magdeburg.de/iti_db/forschung/ajstats/ - last visited on 2018-07-22
[4]https://dune-project.org/ - last visited on 2018-07-22
[5]https://www.webmproject.org/vp9/ - last visited on 2018-07-22
[6]http://www.videolan.org/developers/x264.html - last visited on 2018-07-22

this software the CPU utilization, the output quality and size of the encoded video and the methods used for encoding can be configured. As performance metric, the energy consumption of the encoding process was measured.
VP9 is also video encoding software. It encodes video with the VP9 encoding format. Similar to *x264*, quality constraints such as bitrate, overall quality, target size of the encoded video and CPU utilization can be configured. The non-functional property measured for *VP9* was the CPU time required to encode a video.
Furthermore, the *AJstats* and *x264* data sets only contain binary configuration options. The other data sets consist of both binary and numeric configuration options. Detailed characteristics of the data sets - such as the number of configurations measured - are shown in Table 4.1.

Table 4.1: Configurable software system data sets used for machine-learning.

Name	**Non-functional property**	**#Binary**	**#Numeric**	**#Configurations**
Dune	Performance	19	3	2304
x264	Benchmark-energy	32	0	4608
Ajstats	AnalysisTime	40	0	65536
Trimesh	TimePerIteration	22	4	239360
VP9	UserTime	37	5	216000

4.4 Machine-learning Parameters

In this section, we provide an overview of the machine-learning parameters considered for our metalearning experiments. Further, we state the values used for machine-learning parameters that we did not consider for metalearning, but still required an assignment. However, these are only a subset of all parameters provided by the techniques and therefore all other parameters not listed here are set to their default value. Detailed information on the parameters of each technique is provided in Table 4.2, Table 4.3, Table 4.4 and Table 4.5. Additionally, the abbreviations of the parameter names, we use later on, is underlined in those tables.

For the SPLConqueror case study at the first level of our metalearning process, we considered the error metric, support for logarithmic and quadratic functions, ignoring configurations that only provide marginal improvements as well as limiting the size of interactions in the performance-influence models as configuration options of the software system. A detailed list of the parameters considered for metalearning with a short description and the values used can be seen in Table 4.2.
When using SPLConqueror to generate performance-influence models at the second level of our learning process, we use the default parameter setting provided by the software to create all different models.

For metalearning on the CART technique, we consider parameters that have an influence on how the data set is split and other properties of the tree. The parameters affecting splits consist of the number of configuration options that can be used for

a split, splitting the data set using a random configuration option or the configuration option with the highest accuracy improvement, the number of configurations required to perform splits and the minimal improvement needed to continue the process. As properties of the tree we alter the maximum depth of the tree and the maximum number of leafs. Additionally the random seed was set to a fixed value of 1 to ensure consistent results. An overview of the parameters and the values we used is shown in Table 4.3.

The parameter set of the Random Forest technique is very similar to the CART technique. Compared to CART there is no longer a distinction between random and best split, because for every split the best is used. Furthermore, a parameter for the amount of trees in the forest is added. Like before, we set the random seed to a fixed value of 1. Additionally, the of parallel jobs is set to one. Detailed information on the parameters are listed in Table 4.4.

In the case study on K-Nearest Neighbors Regression we vary the algorithm used to calculated neighboring configurations, the weighting depending on the similarity between configurations and the number of neighboring configurations that influence the prediction. Due to time limitations, the number of parallel jobs for this experiment is set to the number of threads available on the executing hardware. Detailed information of the parameter set used for metalearning is compiled in Table 4.5.

Table 4.2: Considered parameters for Stepwise Regression.

Name	Domain	Step size	Description
quadraticFunctionSupport	Binary	-	Sets whether square functions can be applied to configuration options when creating performance-influence models.
learn-logFunction	Binary	-	Sets whether logarithmic functions can be used for the creation performance-influence models.
limitFeatureSize	Binary	-	Limits the number of configuration options that can partake in an interaction, when it is enabled.
ignoreBadFeatures	Binary	-	Ignore candidate for addition to performance-influence models , if they led to marginal improvements in previous rounds.
LossFunction	[leastsquares,relative]	-	Use the mean relative or squared error as a mean to calculate the error of performance-influence models.
featureSizeTreshold	[3...5]	1	Sets the maximum number of configuration options that should partake in a interaction.

Table 4.3: Considered parameters for CART.

Name	Domain	Step size	Description
splitter	[best,random]	-	Perform the best or a random split to divide a measurement set into two parts.
min_samples_split	[0.01...0.04]	0.01	The minimum percentage of the whole measurement set required at a node to perform a split.
min_samples_leaf	[0.01...0.04]	0.01	The minimum percentage of the whole measurement set required to be at a leaf node.
max_features	[0.25...1]	0.25	The maximum percentage of all configuration options considered as split criterion.
max_leaf_nodes	[50...350]	100	The maximum amount of leaf nodes in the tree.
max_depth	[10...85]	25	The maximum depth of the tree.
min_impurity_decrease	[0...0.01]	0.005	The minimum percentage decrease of learning error required to perform a split.

Table 4.4: Considered parameters for Random Forest.

Name	Domain	Step size	Description
n_estimators	[1...25]	4	The number of trees in the forest.
min_samples_split	[0.01...0.04]	0.01	See Table 4.3.
min_samples_leaf	[0.01...0.04]	0.01	See Table 4.3.
max_features	[0.25...1]	0.25	See Table 4.3.
max_leaf_nodes	[50...350]	100	See Table 4.3.
max_depth	[10...85]	25	See Table 4.3.
min_impurity_decrease	[0...0.01]	0.005	See Table 4.3.

Table 4.5: Considered parameters for K-Nearest Neighbors Regression.

Name	Domain	Step size	Description
algorithm	[brute,kd_tree,ball_tree]	-	The algorithm that computes neighboring configurations.
weights	[uniform,distance]	-	Weight assigned to neighboring configurations based on a distance metric.
n_neighbors	[3...15]	2	The number of neighbors used to calculate the nfp of a configuration.
p	[1...5]	1	The p value of the Minkowski distance metric.
leaf_size	[20...60]	10	The size of the leafs when a tree based algorithm is used to compute the neighbors.

5. Evaluation

In this chapter, we present our metalearning results, discuss the influence of the machine-learning parameters and answer our research questions. First, we present the results to answer our research questions in Section 5.1. Then, we examine which parameters affect the effort and accuracy to answer RQ1 in Section 5.2. In Section 5.3, we investigate how relevant parameters are compared to each other to answer RQ2. In Section 5.4, we discuss the transferability of our results by comparing the influence of machine-learning parameters across multiple case studies to answer RQ3.

5.1 Results

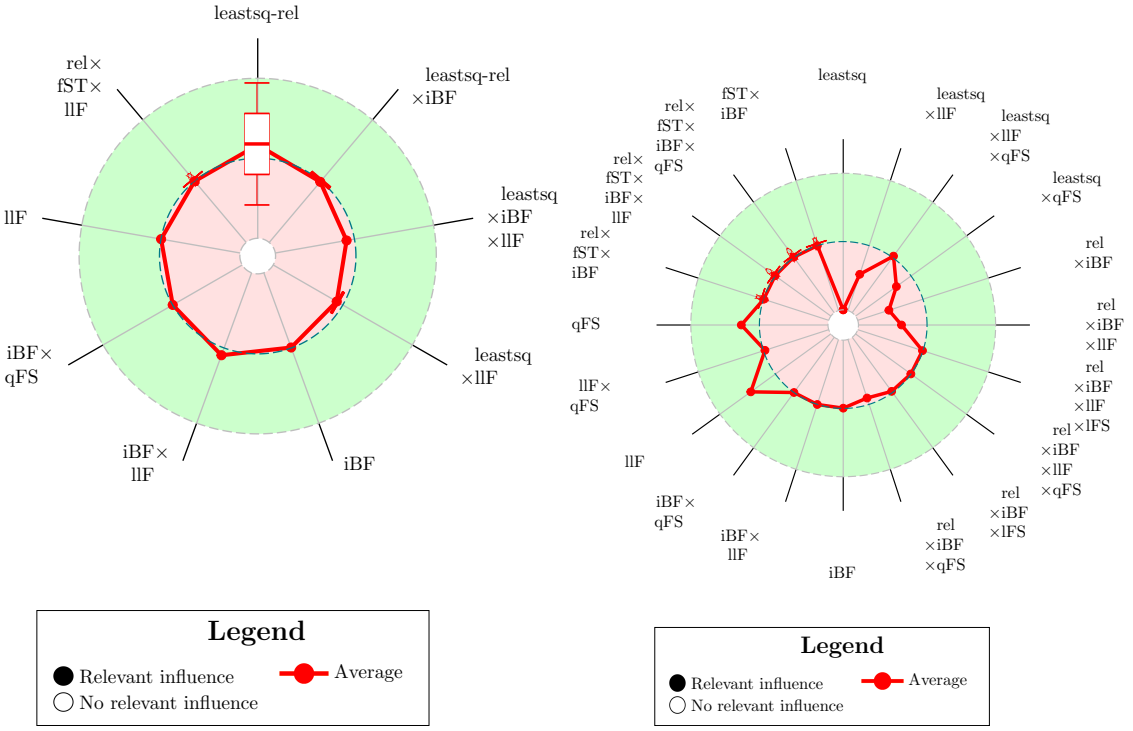
In this section, we provide the results of our experiments. The Star Plots in this section use the abbreviations we introduced in Section 4.4 to ensure better readability.
As already noted in Section 3.3, the axis labels of the Star Plots denote the machine-learning parameters that have an influence. The corresponding point or plot within the circles denotes how the parameter influences the accuracy or effort. Further, for both the effort and error rate the inner circle expresses an improvement in effort and accuracy.
To ensure comprehensibility, we present the Star Plots that display the average influences across all workloads in this section. Further, we show Star Plots that contain the influences across all workloads. All individual Star Plots that are not included in this section can be found in Section A.2.

5.1.1 Results: Forward Feature Selection in Combination with Stepwise Regression

For the Stepwise Regression technique we present the average influences on the accuracy of the performance prediction in 5.1(a). The average influences on the effort

required to build the respective model is displayed in 5.1(b). The comparisons for the influences across all data sets are shown in 5.2(a) and 5.2(b).

Influences from nearly all parameters can be observed for this machine-learning



(a) Average influences on the accuracy for Stepwise Regression

(b) Average influences on the effort for Stepwise Regression

Figure 5.1: Average plots for Stepwise Regression

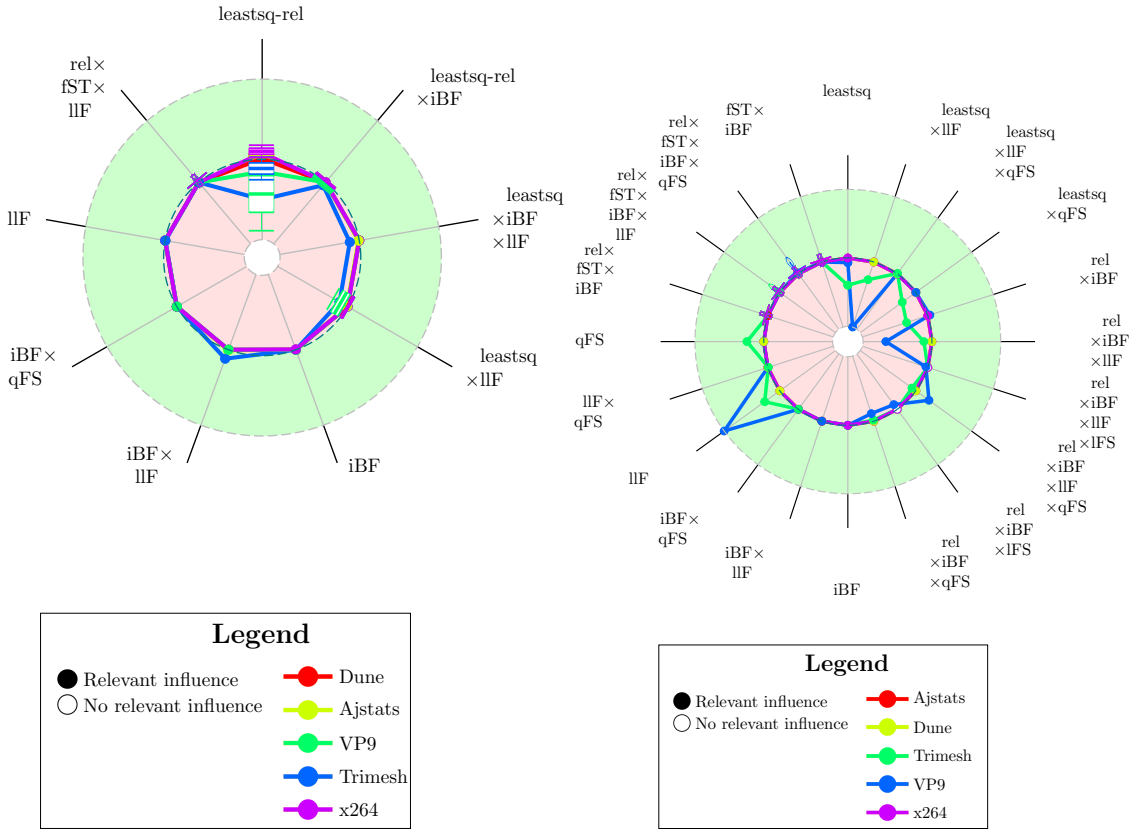
technique. The only parameter that has no influence on accuracy or effort is *limitFeatureSize*.

The *leastsquares* and *relative* alternative group has the highest influence on the accuracy. When the *leastsquares* error function is used, the learned model becomes more inaccurate, while the the *relative* error function leads to more accurate models. However, models are created faster with the *leastsquares* error function. In combination with *learn-logFunction* and *ignoreBadFeatures* the negative impact of *leastsquares* on the accuracy slightly decreases.

The support for logarithmic functions, *ignoreBadFeatures* and support for quadratic functions can also lead to a positive impact on the accuracy, even without *leastsquares*. However, their influences are less relevant. Further, ignoring bad features can also lead to a negative impact on the accuracy if other parameters such as *learn-logFunction* are enabled.

Another weak influence on the accuracy of the machine-learning technique is the interaction between *relative*, *featureSizeThreshold* and *learn-logFunction*. When *relative* and *learn-logFunction* are enabled, the accuracy slightly increases according to the value of *featureSizeThreshold*.

The strongest influence on the effort also stems from the *leastsquares* parameter. This parameter decreases the time required to build models. However, it also decreases the accuracy. Further, most parameters that increase the accuracy of the



(a) Influences on the accuracy for Stepwise Regression across all data sets

(b) Influences on the effort for Stepwise Regression across all data sets

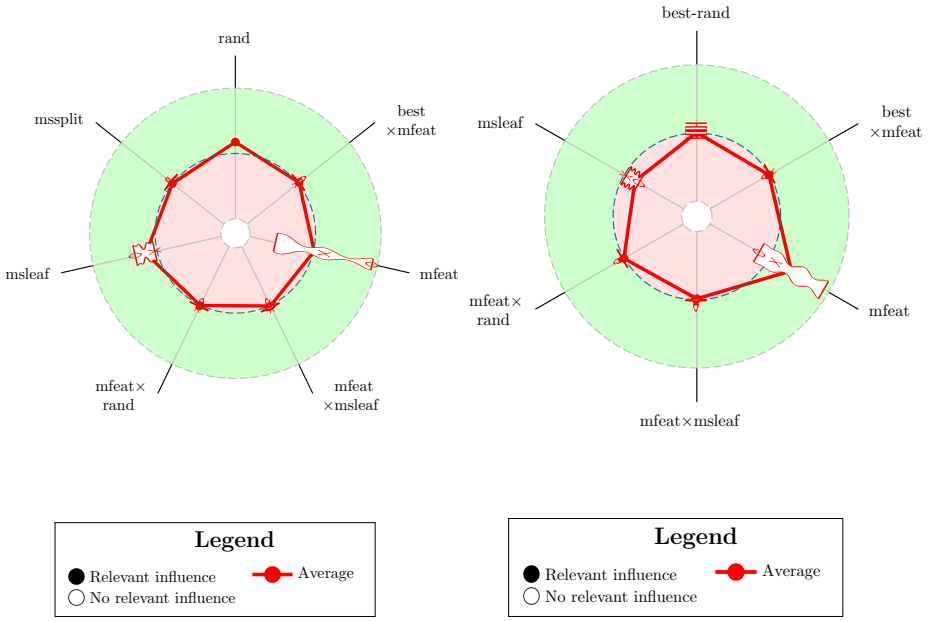
Figure 5.2: Plots with influences across all data sets for Stepwise Regression

model also increase the learning effort required to do so. Examples for such parameters are logarithmic and quadratic functions. However, their negative impact is also considerably weakened when combined with *leastsquares*. The *ignoreBadFeatures* parameter can both increase accuracy and decrease the effort depending on what other parameters are enabled.
Influences containing the *featureSizeThreshold* parameter generally lead to a small increase in the required learning effort as the threshold for interactions is increased. When considering the influences across all data sets some changes can be observed for the *quadraticFunctionSupport* and *learn-logFunction* parameters as the impact changes with the case study. This is most noticeable when examining the effort across all data sets.

5.1.2 Results: CART

In 5.3(a) and 5.3(b), we illustrate the results of the experiments on the Classification and Regression Trees technique on the basis of the the average model. To be able to compare influences over multiple learning workloads, the Star Plots that contain all workloads in 5.4(a) and 5.4(b).

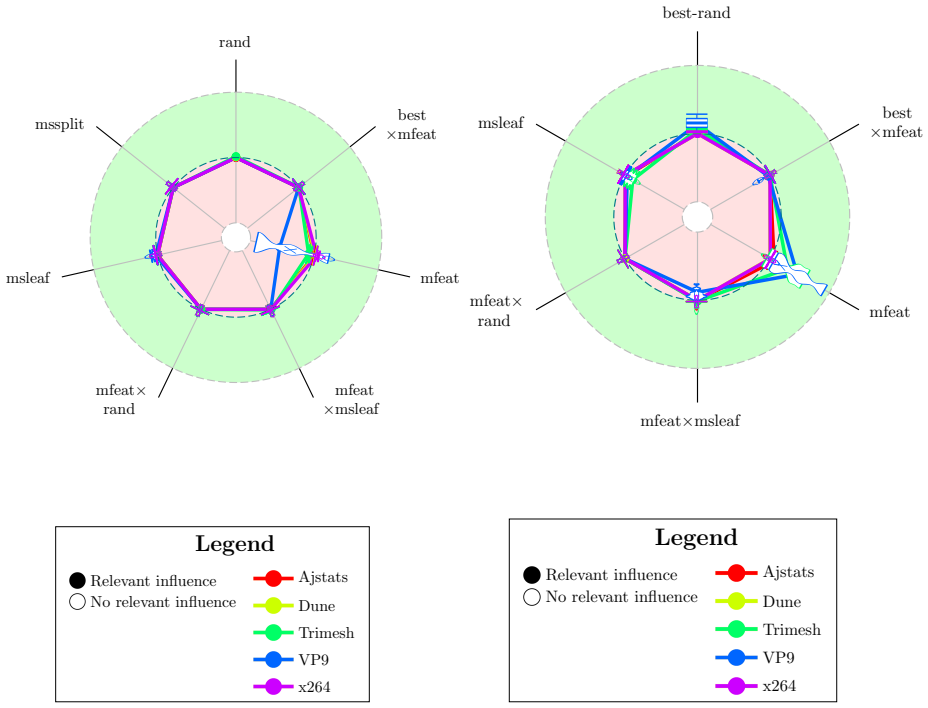
For CART, most parameters generally have an influences on the accuracy or effort. Only the stopping criteria *min_impurity_decrease*, *max_depth* and *max_leaf_nodes*



(a) Average influences on the accuracy for CART

(b) Average influences on the effort for CART

Figure 5.3: Average plots for CART



(a) Influences on the accuracy for CART across all data sets

(b) Influences on the effort for CART across all data sets

Figure 5.4: Plots with influences across all data sets for CART

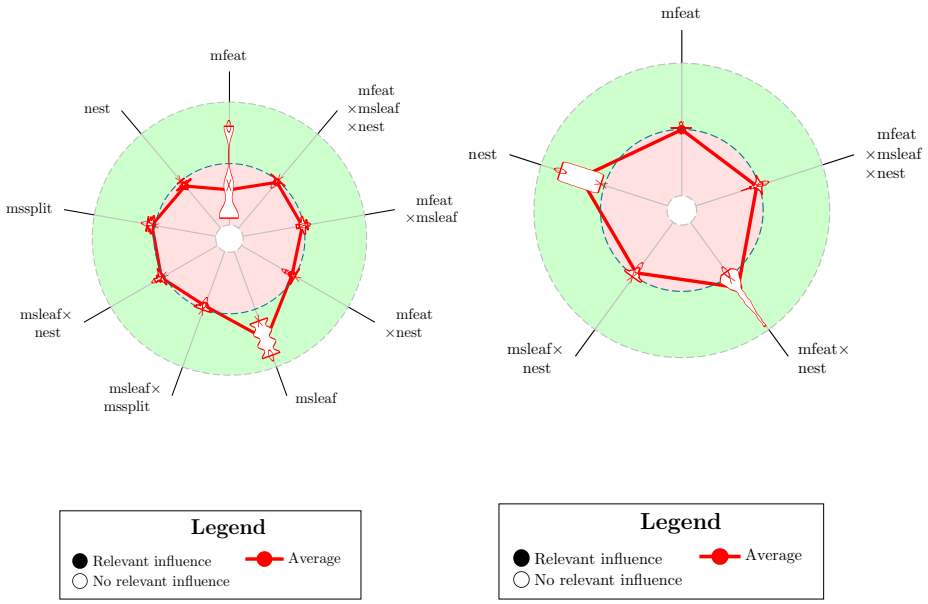
have no influence.
The *max_features* parameter has the highest influence on the accuracy of the machine-learning technique. The *min_samples_leaf* and *random* parameters have the second

highest influence on the accuracy, while *min_samples_split* has a weaker influence on the accuracy. A similar order can be observed when looking at the impact on the effort. The difference is that the *min_samples_split* stopping criterion has no influence on the effort.
The number of configuration options available for a split, *max_features*, increases the accuracy of the machine-learning technique. However the accuracy also decreases when considering all configuration options. When looking at the influence of this parameter on the effort, similar behavior can be observed as the highest value does not result in higher effort.
Accuracy also decreases when using *random* splits compared to the *best* splits. But this also results in less learning effort.
Another strong influence arises from the *min_samples_leaf* parameter. As the threshold for the abort is decreased the accuracy increases and the effort required to build the model increases.
Interactions between *max_features* and the splitting method generally only have small impact. The same applies to *min_samples_split*, with *min_samples_split* decreasing the error. The interactions increase the accuracy if *best* used and decrease it if *random* is used.
For the VP9 and x264 data sets, a decrease in the relevancy of the *random* parameter can be observed as seen in A.9(a) and A.10(a).

5.1.3 Results: Random Forest

The average influences on the accuracy of the Random Forest machine-learning technique are shown in 5.5(a). The corresponding Star Plot for the average influences on the effort is displayed in 5.5(b). Additionally, the plots showing all data sets are displayed in 5.6(a) and 5.6(b).

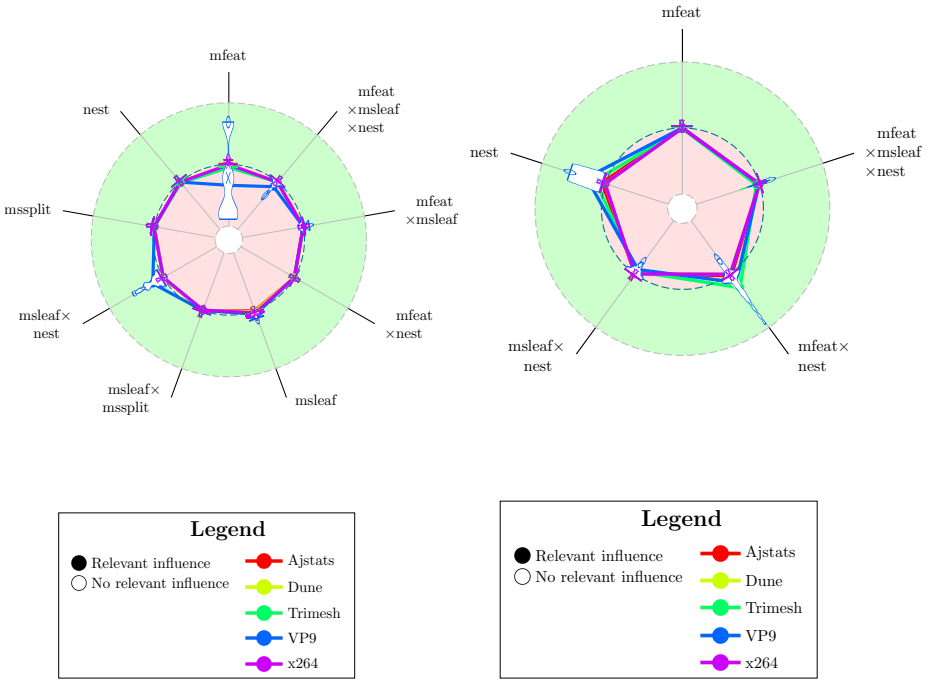
For the Random Forest technique, influences similar to the CART technique can be found. The influence of the *max_features* parameter remains strong and the impact of this parameter on the accuracy and effort stays similar. One difference is a stronger influence from the stopping criterion *min_samples_leaf*. However, the way this parameter influences the accuracy and effort stays the same. Further, the *min_impurity_decrease*, *max_depth* and *max_leaf_nodes* parameters also have no influence in this technique.
As a parameter specific to this technique, the number of trees in the forest, *n_estimators*, has both influence on the accuracy and effort. As the number of trees is increased the accuracy also increases. Furthermore, this parameter also has a strong influence on the effort. Therefore, the effort required to build a model also increases with the number of trees.
There are also interactions with *n_estimators*. A higher value for *max_features* and *n_estimators* results in more effort required to create a model. Further, this parameter also interacts with the *min_samples_leaf* stopping criterion. This interaction reduces the effort, but also decreases the accuracy.
One difference, that can be observed when comparing all data sets, is that for the Ajstats and Dune data sets the stopping criteria have no influence on the accuracy, as found in A.11(a), A.12(a).



(a) Average influences on the accuracy for Random Forest

(b) Average influences on the effort for Random Forest

Figure 5.5: Average plots for Random Forest



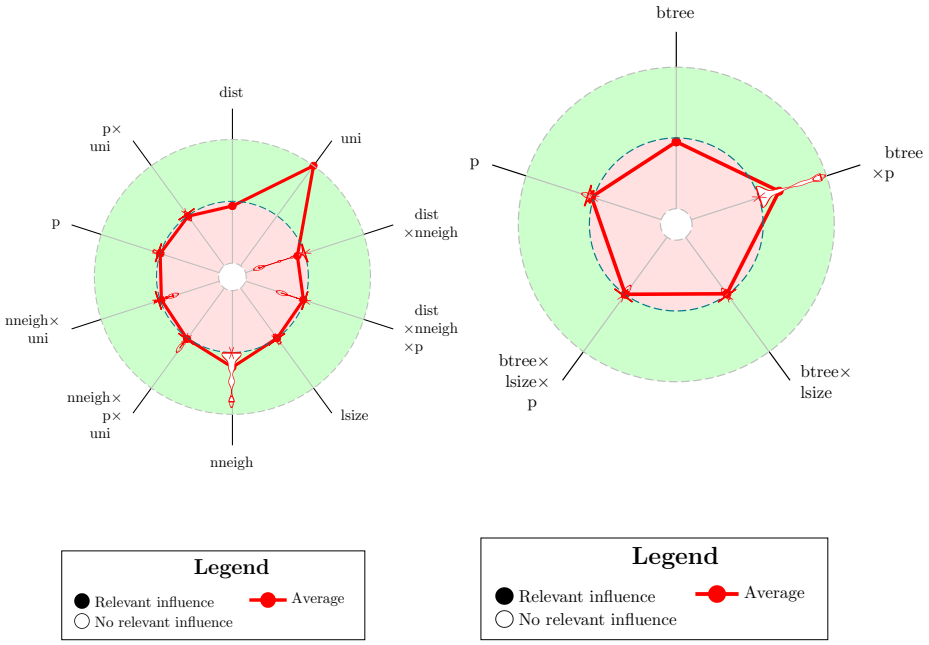
(a) Influences on the accuracy for Random Forest across all data sets

(b) Influences on the effort for Random Forest across all data sets

Figure 5.6: Plots with influences across all data sets for Random Forest

5.1.4 Results: K-Nearest Neighbor Regression

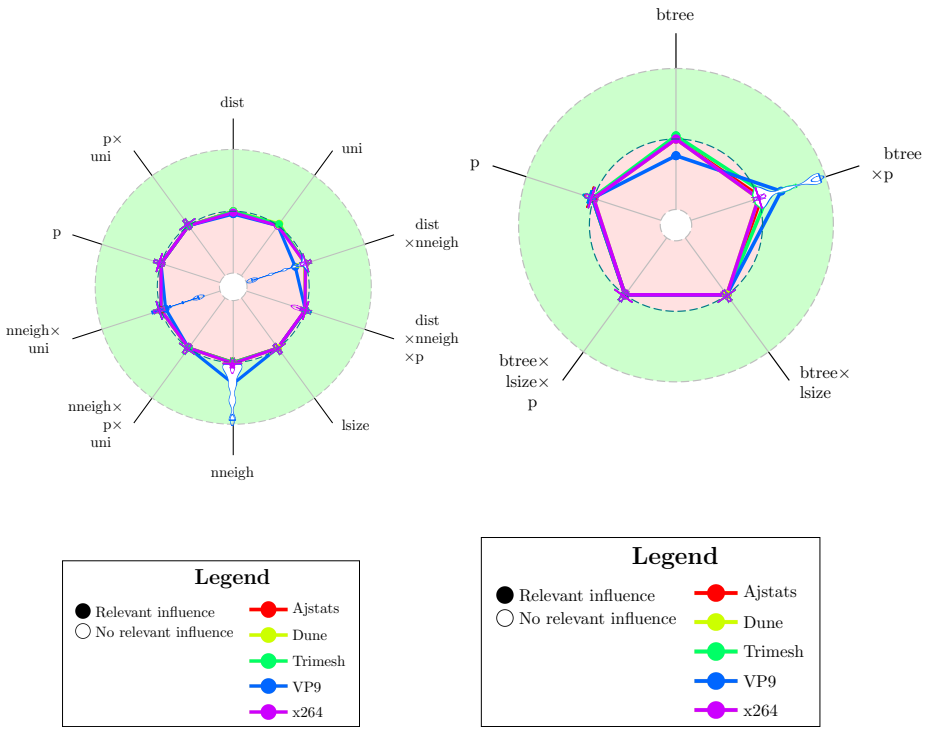
The average influences of the K-Nearest Neighbors technique are displayed in 5.7(a) and 5.7(b). In 5.8(a) and 5.8(b), the corresponding Star Plots with the influences on accuracy and effort for all workloads are shown.



(a) Average influences on the accuracy for K-Nearest Neighbors

(b) Average influences on the effort for K-Nearest Neighbors

Figure 5.7: Average plots for K-Nearest Neighbors Regression



(a) Influences on the accuracy for K-Nearest Neighbors across all data sets

(b) Influences on the effort for K-Nearest Neighbors across all data sets

Figure 5.8: Plots with influences across all data sets for K-Nearest Neighbors Regression

For the K-Nearest Neighbors technique, most parameters except for *brute* and *kd_tree* have an influence on accuracy or effort.
The highest influences on the accuracy are caused by the number of neighbors used for prediction, *n_neighbors*, and the weighting metric that is used. As the neighborhood grows, the accuracy of the technique decreases.
The *uniform* weighting metric generally results in less accurate predictions compared to the *distance* metric. Further, the *distance* metric also has lower error rate for a higher number of neighbors compared to *uniform*. The p value of the distance metric also increases the accuracy.
The parameters that influence the effort are *ball_tree*, p and *leaf_size*. The *ball_tree* algorithm leads to lower effort for low p values. This interaction accounts for the strongest influence on the effort. The *leaf_size* decreases the effort in combination the *ball_tree*. The p value of the distance metric slightly increases the effort. For this case study, differences in the impact of the *n_neighbors* can be found. However, in A.16(a) the number of neighbors has very small influence on the accuracy.

5.2 Influence of Machine-Learning Parameters

In this section we discuss how machine-learning parameter influence the learning effort and accuracy by using the results presented in Section 5.1. To answer our research question, we examine in which way the parameters influence the accuracy and effort.

5.2.1 Discussion: Stepwise Regression

The *relative* parameter decreasing the error indicates that the squared error is not as well suited for the error calculation performed in the feature subset selection.
As the *relative* and *leastsquares* parameters determine which candidates are added to the model during the feature subset selection, they interact with parameters that add or remove possible candidates. Such parameters are *learn-logFunction*, *ignoreBadFeatures* and *featureSizeThreshold*. Generally, those interactions increase the accuracy of the resulting model.
Support for logarithmic functions allows for more possibilities when building models and therefore also for more accurate models. The *quadraticFunctionSupport* parameter can also increase the accuracy for the same reason.
The increase in accuracy from *ignoreBadFeatures* might be due to the greediness of the feature subset selection, because it always adds the candidate to the performance-influence model that leads to the lowest error at the current iteration. However, this influence is not necessarily a good addition for future iterations. This parameter decreases the accuracy in combination with *learn-logFunction*, so logarithmic influences block other possibly good candidates from being added to the performance-influence model.
Because the *featureSizeThreshold* parameter can add new candidates during the feature subset selection, it leads to an increase in accuracy.
When looking at the influences on the effort required to build models, all parameters that have influences on the accuracy also have influences on the effort.
The *learn-logFunction*, *featureSizeThreshold* and *quadraticFunctionSupport* parameters increase the effort as they increase the number of candidates that have to be

evaluated in the feature subset selection process. Further, the increase in effort using these parameters is higher than the accuracy improvement, since they add all possible candidates they can generate to the feature subset selection. However, not all of these candidates lead to an improvement in accuracy.
Because the *ignoreBadFeatures* parameter removes candidates that would have to be evaluated in the feature subset selection, it decreases the effort.
As *leastsquares* produces inaccurate models, the algorithm also aborts earlier. Therefore, less effort is required to create a model. The interactions of *leastsquares* with *quadraticFunctionSupport* and *learn-logFunction* that have a positive influence on the effort only balance out the base influences of *quadraticFunctionSupport* and *learn-logFunction*. This is done, because in those interactions the algorithm aborts so early that *quadraticFunctionSupport* and *learn-logFunction* have no noticeable impact on the effort.

5.2.2 Discussion: CART

The *max_features* parameter leads to higher accuracy as the value of this parameter increases, because a bigger pool of split candidates is more likely to contain good candidates. However, the highest value does not lead to the most accurate models. This might be due to the greediness of the technique leading to an earlier satisfaction of a stopping criteria. That this parameter expresses similar impact on the effort is a further indication for that.
The *max_features* parameter also interacts with the two alternatives *best* and *random*, because it is the parameter that influences which configuration options are available for a split. Generally, the accuracy increases as both split methods benefit from having more possibilities. However, the combination with random splits can lead to a higher effort, because splits are just randomly selected and a higher number of splits with little increase in accuracy might be performed to reach an accurate model.
In addition to this, the *random* parameter also has an individual influence on the accuracy and effort. As the configuration option used for a split is randomly selected from the set of available options, the machine-learning technique also becomes less accurate when it is enabled. On the other hand, it also decreases the effort since the search for the best split is omitted.
From the set of parameters that determine the stopping criteria only influences from *min_samples_leaf*, *min_samples_split* and *min_samples_split* could be identified. They decrease the accuracy as more stricter stopping criteria generally result in an earlier abort. Therefore, the technique becomes more inaccurate but also requires less learning effort.

5.2.3 Discussion: Random Forest

The machine-learning parameters that influence the accuracy and effort of the Random Forest techniques are similar to the CART technique. The *n_estimators* parameter, that is added in this technique, determines how many trees are built, increases the accuracy of the technique as more trees are created. Since this is also connected

to higher computational costs, *n_estimators* also increases the effort.
The *max_features* parameter now also interacts with *n_estimators*, but otherwise the impact stays the same compared to the CART technique. The stopping criteria *min_samples_split* and *min_samples_leaf* decrease the accuracy of the model, while the strength depends on how they interact with each other and *n_estimators*. Both parameters also lead to less effort required to build a model. Like before, the other stopping criteria have no influence on the accuracy and effort of the technique.

5.2.4 Discussion: K-Nearest Neighbors

Increasing the size of the neighborhood results in less accurate results, because less similar configurations are included in the prediction. These configurations can have more different values for their configuration options compared to the configuration that has to be predicted and therefore more different values for the non-functional property. This decrease in accuracy is lessened by using *distance* based weighting, because takes the similarity between configuration into account and assigns a higher weight to more similar configurations.
The *uniform* weighting parameter decreases the accuracy, because it does not take the similarity between configurations into account.
Compared to that, the *distance* parameter increases the accuracy, because all predictions with weighting have an relative error of 0% since the learning and validation sets are equal. The p parameter of the Minkowski distance metric decreases the accuracy, because it influences which configurations are considered the nearest neighbors and for a higher p value configurations with more changes to the configuration options can have a smaller distance than other configurations with less changes to the configuration options.
Further, the *leaf_size* has close to no influence on the accuracy, because the parameter only influences the size limitations of the algorithms that compute the neighbors. Most parameters of this machine-learning technique that influence the accuracy have no influence on the effort, because no model is created and the effort is only influenced by the computation of the neighborhood and the distance to neighbors.
Using the *ball_tree* algorithm decreases the time required to calculate neighbors compared to the brute force method *brute* as is just the brute force computation of the neighbors. However, as the p parameter increases, the performance improvement from using *ball_tree* decreases, as it also computes distances. The effort increases with the p value of the Minkowski distance metric, as this results in more calculations that have to be performed.

Generally, influences of most machine-learning parameters can be found. The accuracy of the machine-learning techniques is mostly affected by the parameters that determine possible splits, candidates or neighbors and the parameters that evaluate those. These parameters can increase the accuracy of the machine-learning techniques depending on the right setting. Further, most parameters that increase the accuracy also lead to an increase in effort.

5.3 Relevancy of Machine-learning Parameters

In the previous section we discussed how the machine-learning parameters affect the accuracy and effort of machine-learning techniques. Here, we discuss the relevancy of parameters by comparing their relative impact on the performance of the machine-learning technique.

5.3.1 Discussion: Stepwise Regression

In the Stepwise Regression case study, we can identify very distinct levels of relevancy, while only few parameters have a strong influence. The *relative-leastsquares* alternative group has the biggest impact on the accuracy of the technique, as it determines how the error of performance models is calculated. Therefore, it is responsible for the decision which influences are added to a performance-influence model and when the machine-learning process stops. As a consequence of *leastquares* leading to an early abort, the individual influence of this parameter accounts for more than two times the influence on accuracy of any other parameter.
The next set of parameters only has a slight influence on the accuracy. These parameters are *learn-logFunction* and *ignoreBadFeatures*. However, as they add and remove candidates that need to be evaluated in the feature subset selection they have an high influence on the learning effort. Logarithmic behavior can be approximated to a certain extent with linear functions or polynomials and can only be applied to numeric options. Therefore, *learn-logFunction* is not as relevant for the accuracy. For *ignoreBadFeatures*, the number of candidates that will get ignored is also only the subset that performed bad in the previous feature subset selection round. Therefore, those two parameters are less relevant than *leastsquares*.
The *quadraticFunctionSupport* parameter also has a high impact on the effort. However, it only has a weak influence on the accuracy as square functions can also be expressed with interactions in later iterations of the feature subset selection. However, it can still have an impact as it allows square functions without the requirement of adding linear influences.
The impact of *featureSizeThreshold* being low could be an indicator that high order have a weaker influence on the performance or do not occur commonly. This aligns with findings by Liebig et al. [LAL+10], who found that the most common interactions in configurable software are of low order.

5.3.2 Discussion: CART

The most influential parameter in the CART case study is *max_features*. Both the accuracy and effort are mostly affected by this parameter as it determines which splits are possible. As splitting the learning set is the core mechanic of the CART technique, this parameter also has a very high influence.
The next group of parameters only has a weak influence on the accuracy and effort. This group contains the *best* and *random* alternative group and *min_samples_leaf*. Each individual parameter in this group has less than $\frac{1}{3}$ of the influence of *max_features*. Both the best and the random split directly and indirectly depend on which configuration options are available so they are less important than *max_features*. The

importance of the stopping criterion *min_samples_leaf* is restricted to the domain we used for our experiments, as the maximum value for this parameter was 4%. All other parameters have no influence on the effort.
The *min_samples_split* parameter has close to no influence on the accuracy. So this stopping criterion was rarely reached for the values we assigned to it. Like for the previous machine-learning technique, the *max_depth* option having no influence indicates that mostly low order interactions influence the performance of configurable software systems. Further, *min_samples_split* having no or close to no influence could be an indication that configurable software systems are less prone to the problem of unbalanced splits.

5.3.3 Discussion: Random Forest

Since Random Forest is a derivation of the CART technique, the parameters also display similar relevancy. This is also displayed by the relevancy of the machine-learning parameters, as the relevancy of parameters in CART and Random Forest both depend on the same factors. Further, as creating trees is the most computational intensive task for this technique, the *n_estimators* is highly relevant for the effort. Additionally, the number of estimators is relevant for the accuracy, because more trees in the forest results in less accurate trees, produced due to the randomness of this techniques, having less impact on the overall prediction.

5.3.4 Discussion: K-Nearest Neighbors

The two most important parameters on the accuracy of the K-Nearest Neighbors technique are the distance weightings *uniform* and *distance* and the size of the neighborhood *n_neighbors*. The distance weighting leading to such high accuracy is due to the prediction error being 0% when using the *distance* based metric as for both learning was performed on all configuration was performed on all configurations in the data set. However, these parameters have no influence on the effort.
All other parameters have marginal or no influence on the effort. The p parameter only has a small impact on the accuracy, because the effect of configurations having lower distance despite displaying more changes in their configuration options than other configurations is only limited to few neighbors that still display similarities. Further, the other parameters only affect the computation of the neighbors.
As these parameters influence the computation they have high impact on the effort. The *ball_tree* parameter in combination with the p parameter has the highest influence on the effort. While not displayed in our results, the *kd_tree* and *brute* parameter have high influence on the accuracy as they are the alternatives to *ball_tree*. Additionally, the *leaf_size* parameter can have a weak impact on the effort.

Only a few parameters like *max_features* and *leastquares* are very relevant for the accuracy or effort. Further, most parameters with strong impact are very relevant for both accuracy and effort. Other parameter are considerably less influential than those parameters with some parameters even having close to no influence. Only a few parameters have no influence whatsoever.

5.4 Across Case Study Comparison

In this section, we explore whether the relevancy on machine-learning parameters depends on the configurable software system that was used for metalearning to answer RQ3. For this research question we mainly focus on the Star Plots that show results for all subject systems. Additionally, we refer to individual Star Plots displayed in Section A.2 to point out differences.
To answer the research question, we compare the impact of the machine-learning parameters for all case study.

5.4.1 Discussion: Stepwise Regression

Generally, we can observe that the overall picture stays the same for all subject systems. Evaluation of candidates is the core part in this technique. Therefore, as the *leastsquares* error function does not perform good when evaluating candidates, the *leastquares* and *relative* alternative group has the strongest influence on accuracy in all five workloads. This group also retains its strong influence when looking at the effort, because an earlier abort of the feature subset selection caused by *leastquares* leads to a big decrease of time consumption, independent of the workload used for learning. For accuracy, changes in the relevancy can only be observed for parameters with weaker influences.
The influences of logarithmic and quadratic support depend on the data set that was used for learning. As mentioned before, these parameters are only useful when the data set contains numeric configuration options and also rely on the traits of the learning set. Therefore, the relevancy of this machine-learning parameter depends on the workload. For workloads with no numeric configuration options these parameters have no influence on the accuracy and effort. Such behavior can be observed when learning with the Ajstats and x264 data sets as seen in A.1(a), A.1(b), A.5(a) and A.5(b). However, all other influences retain a similar level of relevancy.

5.4.2 Discussion: CART

When looking at the results for the CART technique, the order of the parameters mostly stays the same. The *max_features* parameter is the most influential parameter in all data sets as with a higher number of configuration options available for a split more influences can be modeled. The splitting criteria and *min_samples_leaf* sustain moderate influence. However, there can be small changes in their relative impact on the effort. A example for this is found when comparing the Dune data set in A.7(b) to the results of Ajstats in A.6(b). Further, for the VP9 and x264 data sets a decrease in the relevancy of the *random* parameter can be found, as seen in A.10(a) and A.9(a). This could be a result of the randomness. However, both data sets share the similarity that they were obtained by measuring video encoding software. So, a possible explanation is the high number of alternative groups in these data sets as the influences of alternatives can be modeled in multiple ways using different configuration options. In this case, it is more likely that a suitable configuration will be randomly selected that compared to optional configuration options.

5.4.3 Discussion: Random Forest

The parameters with the strongest influence on the accuracy and effort of the Random Forest technique mostly also have the strongest influence across all case studies. For those parameters one change that can be observed, appears in the results of the Dune case study in A.12(b) as the impact of the *max_features* on the effort becomes very low. This is due to the specific traits of this case study, as it has less configurations options than any other case study.
When it comes to the parameters with weaker influences, the influence of the *min_samples_split* is only above zero for the Trimesh and the x264 data sets. This can be seen A.13(a) and A.15(a). For Trimesh, this is due to a higher influence from numeric configurations options, which results in unbalanced splits. Similar to this x264 is not split up evenly as it has a high number of alternative groups.
The relevancy for all other other parameters generally stays the same or similar.

5.4.4 Discussion: K-Nearest Neighbors

The only difference in the influence of significant parameters can be found in the Ajstats data set. In A.16(a), the *n_neighbors* parameter has very low influence on the accuracy. The reason for this is that in this particular data set all configurations have relatively similar non-functional property values. Therefore, the accuracy barely decreases when less similar configurations are used for the prediction. However, the *n_neighbors* has a strong influence on the accuracy in all other data sets. In these data sets a high spacial dissimilarity leads to a high difference in the values of the non-functional property.
Otherwise, no differences in the relevancy of parameters can be observed. This is due to the low number of parameters that influence the accuracy and effort, as seen in 5.7(a) and 5.7(b). Furthermore, the high influence of the *distance* parameter is independent from the data set as it always leads to an error of 0%.

The most relevant parameters are mostly consistent across all data sets. However, there exist specific data sets where certain parameters are less relevant. For example, logarithmic support has no influence on the effort when creating models for data sets that contain no numeric configuration options.

6. Validity

In this chapter, we discuss possible threats to the validity that could affect our experiments and lead to invalid results. Further, we describe how we tried to minimize those threats. The threats that arise when measuring performance such as execution time and our ways to deal with them are discussed in Section 6.1. In section Section 6.2, we examine the transferability of our results.

6.1 Internal Validity

When measuring performance, there can be multiple factors not tied to the experiments that influence the performance and lead to inaccurate results. Since we measure the execution time of the machine-learning techniques as one metric for performance, these factors have to be addressed.
One major group of factors includes dynamic properties of the hardware the experiments are executed on. Such properties include manufacturer specific techniques to dynamically scale the frequency of processor cores and increasing the number the number of parallel threads depending on the workload, but also more common techniques caching. To prevent potential error produced due to manufacturer specific performance enhancements we disabled *Turbo Boost* and *Multithreading* for our experiments. Other factors such as caching are either essential or could not be disabled and, therefore, are included in the means we use to deal with unknown factors.
The other major factor we consider is other software being executed in parallel to our experiments. To prevent other experiments from running on the same node at the same time, we blocked the node when our experiments were performed. So, not more than one single experiment could be executed on the same node at the same time. However, other background software like the operating system could not be prevented and could not be dealt with directly.
To mitigate the error produced by unknown factors or factors that we could not deal with directly, we repeated each measurement three times and used the average of those runs. Additionally, to prevent outliers from influencing our results we computed the relative standard deviation of the three runs. If the relative standard

deviation of the effort was higher than 10%, we assumed that the error in that particular measurement was too high, discarded it and repeated the measurement until we obtained stable results. For accuracy we apply a more stricter rule and only consider measurements with constant accuracy across all runs.

6.2 External Validity

Despite using five different workloads, obtained from measuring the performance of configurable software from different domains, for our experiments, they were only limited to the domain of configurable software systems. Other data sets from domains that are not linked to configurable software might display completely different behavior and therefore produce different results. Additionally, there could be configurable software systems with performance traits not present in our data sets. This could lead to our results not being representative for those systems. As we examined four specific machine-learning techniques for our experiments, our results might also not be suited for machine-learning techniques we did not consider. The only mean to deal with this problem would be using more workloads and machine-learning techniques, which was not possible due to time restrictions.
Another problem that might arise is the lack of sampling. The usage of sampling could alter the representation of the data set and change the influences of machine-learning parameters. However, we did not consider sampling for our experiments, because of time limitations and the steep increase in time required to perform all experiments. So our results might not be applicable when using sampling.
The last issue is the influence of randomness in specific machine-learning techniques. For example, influences from randomness can appear in the CART technique(Section 2.5) when using random splits with the *random* parameter(Table 4.3) or in the Random Forest technique(Section 2.6) when the learning set for a tree is generated. For our experiments, we used a hard coded random seed to ensure consistent and replicable results. Nonetheless, different seeds lead to different results, so the influence of parameters dependent on the randomness also depend on the value of the random seed. Albeit this, we can assume that a random approach can never be better than the exhaustive approach as seen in 5.4(a). So in cases where both approaches exist, only the magnitude of their difference will vary.
Since we also use machine-learning to create performance-influence models, the result we present in our results might also differ depending on the parameter setting of the Stepwise Regression technique. To ensure consistent results within our experiments, we always used the default parameter selection to create performance-influence models at the second level of our metalearning approach.

7. Related Work

Several work has been done using machine-learning to predict the performance of configurable software systems.
Westermann et al. [WHKF12] used CART among others to build models and predict the performance of configurations. They predicted the performance of the Java Virtual Machine and an SAP ERP system. In their results, they found that despite being fast the CART technique was very inaccurate and unreliable when used for performance prediction. However, they did not take the influences of machine-learning parameters on the prediction accuracy into account.
Guo et al. [GYS+18] also used a machine-learning technique based on CART to predict the performance of an extensive set of configurable software systems. In addition to the normal CART technique, their approach included performing parameter tuning on the machine-learning parameters to find the optimal configuration of the machine-learning technique. However, their parameter tuning approach was only based on search strategies and no knowledge on the influence of the machine-learning parameters was used or extracted. Predicting the performance of an extensive set of configurable software systems including AJStats, x264, the LLVM compiler, Berkeley DB and others, their approach with parameter tuning outperformed the traditional CART in 8 out of 10 cases in terms of prediction accuracy. In the other two cases both approaches performed very similar.
Another work in this area was performed by Siegmund et al. [SGAK15]. They created expressive performance-influence models to describe and predict the performance of configurable software by using their machine-learning software *SPLConqueror*. With these models they could predict the performance of configurable software systems such as x264 and Dune with an error mostly between 10% and 19%. However, they could also observe higher error for certain case studies and sampling techniques.

In the area of using machine-learning to find the performance-optimal parameters for machine-learning techniques, one of the earliest work was executed by Soares et al. [SBK04]. They used information from previous machine learning, such as traits of the learning set and accuracy of the different parameter settings, to try to predict

the optimal parameters for Support Vector Machines using the K-Nearest Neighbors technique. In their experiments they found that this approach is both more robust and accurate than using a heuristic to compute the optimal parameters.
Gomes et al. [GPS+12] and Reif et al. [RSD12] both proposed similar approaches to enhance the prediction of machine-learning parameters for Support Vector Machines. Both approaches initially try to predict the optimal configuration similar to Soares et al., but instead of using the result of the prediction as the final setting, they use it as basis for an search strategy and try to find better parameter settings. Gomes et al. and Reif et al. both found that such approaches finds good parameters faster than traditional search strategies, while the resulting error remained similar. Reif et al. could also achieve the same accuracy as an extensive Grid Search, while requiring less time compared to the extensive search.
However, none of these approaches extracts information about the influence of the machine-learning parameters on the accuracy and their tendency. The biggest limitation of the approach proposed by Soares et al. is that they had to limit the values of machine-learning parameters to a discrete value range. Therefore, they could only find parameters within that range, even if the accuracy increases for values they did not consider as they did not explore how the parameters influence the accuracy. Gomes et al. and Reif et al. aim at overcoming this limitation by additionally using search strategies to find better parameters. However, this search can also be quite cost intensive as machine-learning has to be performed for each candidate. Additionally, the search has to be performed for each new data set that is used for learning as they do not extract any previous knowledge about the influence of the parameters.
Vanschoren et al. [VBPH12] investigated how the parameters of SVM and Random Forest influence the accuracy by comparing the accuracy of different parameter settings across a database of data sets for machine-learning. In their experiments on SVM they could observe two different behaviors depending on the number of features in the data set that was used learning. For Random Forest they could observe an increase in accuracy as the size of the forest grows for most data sets. However, for a single data set with only binary features they could observe a decrease in accuracy as the size of the forest grows.

8. Conclusion and Future Work

8.1 Conclusion

In this thesis, we examined the influence of machine-learning parameters on the prediction accuracy and learning effort for four different machine-learning techniques. For this, we performed machine learning with the Stepwise Regression, CART, Random Forest and K-Nearest Neighbors and measured the accuracy and effort for different parameter setting. To investigate the influence of the parameters and answer our research question, we performed machine-learning again to create performance-influence models describing the influences of machine-learning parameters. This procedure was performed by learning on five different configurable software systems.

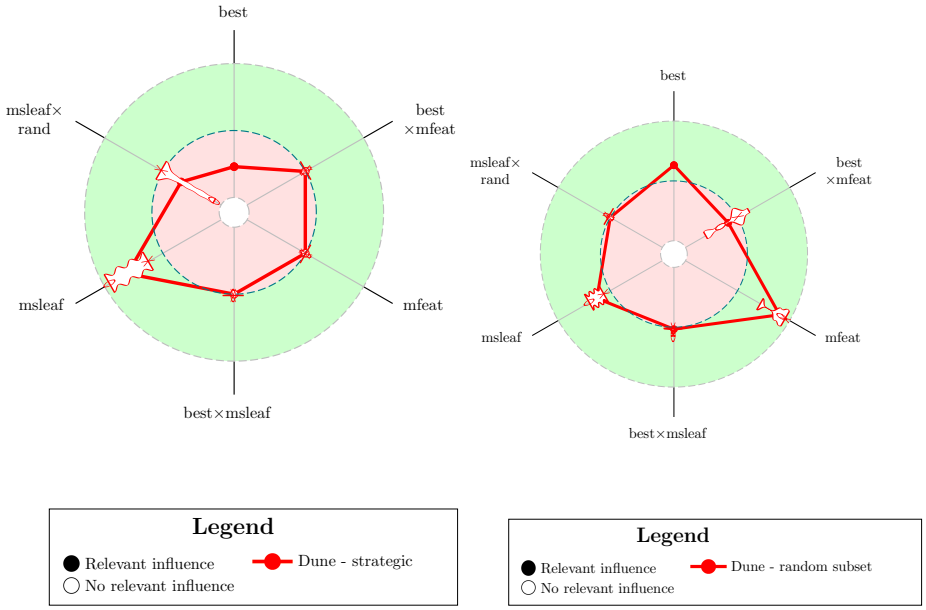
In the first research question, we investigated which machine-learning parameters influence the accuracy and effort and their effect on those metrics. We found that most machine-learning parameters have an influence on accuracy or effort. Further, we found that most parameters that increase the accuracy also increase the effort required to learn.
For the second research question, we investigated if some machine-learning parameters have stronger impact on the accuracy and effort than others. We observed that a small set of parameters has the strongest influence on those metrics, while the rest is noticeable less relevant compared to those.
Last, we compared the relevancy of the machine-learning parameters across several data sets used for learning. We found that the most relevant parameters generally were the same for all data sets. However, some differences could occur in the relevancy of some relevant parameters depending on the case study. Those parameters mostly depended on specific characteristics of the data set that was used for learning such as the absence of numeric configurations options when supporting logarithmic influences.

8.2 Future Work

Here, we discuss possibilities for future work based on this thesis.

In Section 3.1, we mentioned that we used no sampling for our approach. However, sampling is used to decrease the effort required to measure a configurable software system for performance prediction. Therefore, a study on how sampling affects the results of the metalearning could be conducted in the future. For this, sampling strategies such as *Pair-Wise*, *Option-Wise*, *Random* and *Plackett-Burman*, which have been used by Siegmund et al. [SGAK15] on configurable software systems, could be used. In early experiment results on CART - found in 8.1(a) and 8.1(b) - using sampling strategies we could find indications that there exist dependencies between the sampling strategy that is used and the optimal parameters. For example, differences in the influences of the parameters could be observed when using a full random sampling compared to the influences found when learning with the whole population. For this sampling set, parameters that introduce some randomness to the CART technique could provide higher accuracy, because the *best* parameter decreases the accuracy. This might indicate that for some sampling strategies such as random sampling, which preserve less of the structure of the data set used for learning, overfitting is a problem that influences the impact of machine-learning parameters. On the other hand, we could observe that for the more strategic sampling ,using the combination of the *Plackett-Burman* and *Pair-Wise* sampling strategies, the number of configuration options *max_features* used for learning does not influence the accuracy. On the other hand, the stopping criterion *min_samples_leaf*, that determines the minimum percentage of the total configurations a node can have before it has to be a leaf, increases in relevancy. Further, the accuracy increases the stricter this stopping criterion is. This could be another indicator for possible dependencies between the structure of the sampling and the influence of parameters, because pairwise sampling creates a sample set where each possible combination of two binary options is present. With such a sample set, splitting it might be possible even with a low number of options available, yet it might be very prone to overfitting due to the structure. However, further experiments are required to give tangible answers to research questions as we currently can not distinguish if some of these influences are specific to certain case studies, are due to randomness and are caused by the sampling strategies.

As multiple non-functional properties were measured for the data sets we used, future work could also investigate if there is a connection between the influence of the machine-learning parameters and the non-functional properties that were used. This could open the possibility to find further criteria for the transferability of our results, since the correlation between the non-functional properties could be measured.



(a) Strategic sampling (b) Random samping

Figure 8.1: Influences of parameter on the CART technique using sampling.

Another issue we mentioned in Chapter 6, was the randomness of some machine-learning techniques. To ensure consistent results we used a single fixed random seed for all of our experiments. However, we could not examine to which extent our results were influence by the random value we used. To solve this issue in the future, the experiments could be repeated several times with a set of different random seeds to eliminate the influence of the randomness.

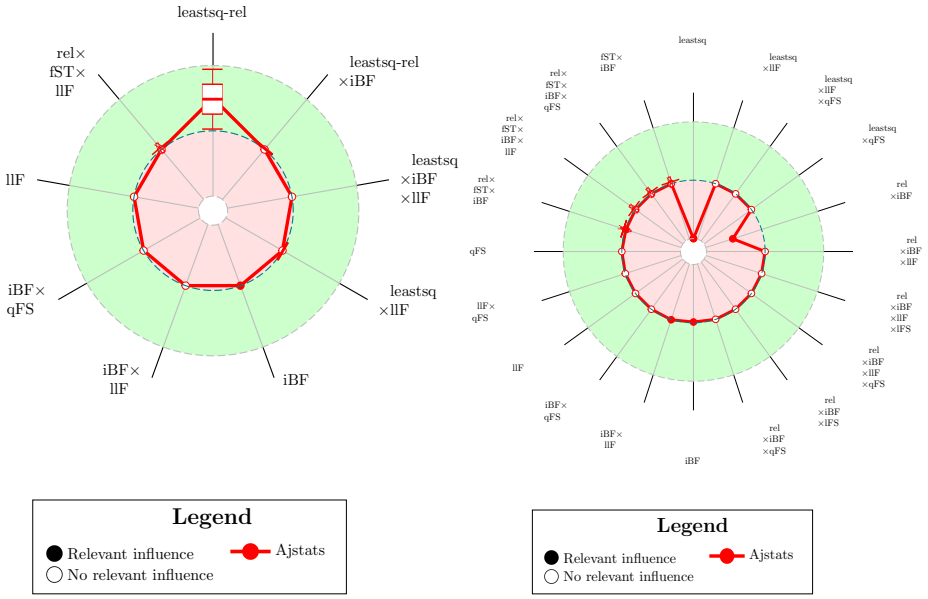
A. Appendix

A.1 Content of DVD

A DVD with the scripts and tools used for this thesis and the results of our measurements and metalearning is added to this thesis. The DVD includes:

- thesis.pdf - A digital version of this thesis
- SPLConqueror.7z - The SPLConqueror machine-learning software
- SPLConquerorProfiler.7z - A modified version of SPLConqueror with a profiler
- scripts - The folder containing all the scripts that were used. Additionally, a README is added explaining the scripts
- results - Contains the results of our measurements and metalearning. Also contains a README.
- misc - Contains Feature Models of the machine-learning techniques.
- workload.7z - Contains the workloads used for learning.
- ViPe - Contains the visualization tool.

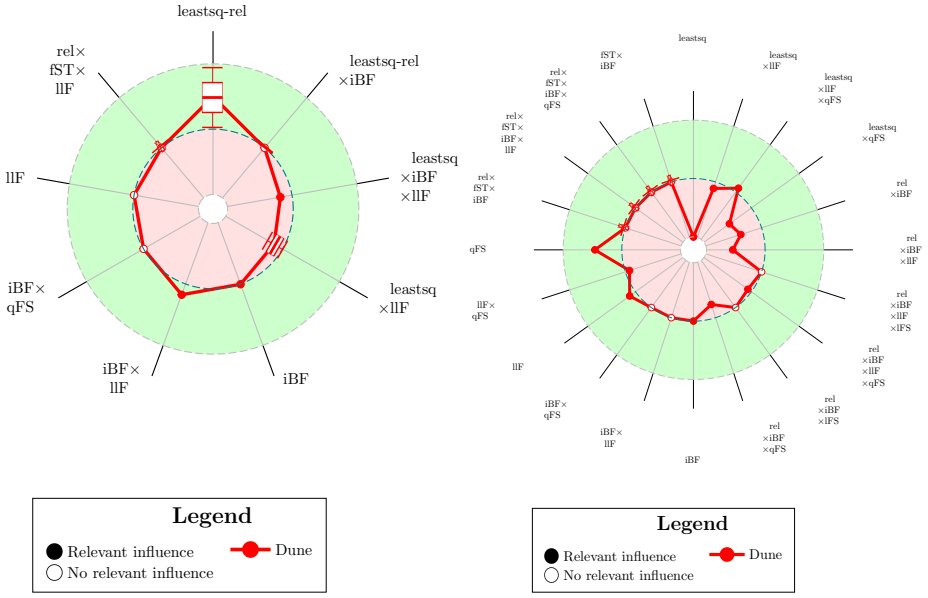
A.2 Additional Star Plots



(a) Influences on the accuracy for Stepwise Regression and Ajstats.

(b) Influences on the effort for Stepwise Regression and Ajstats.

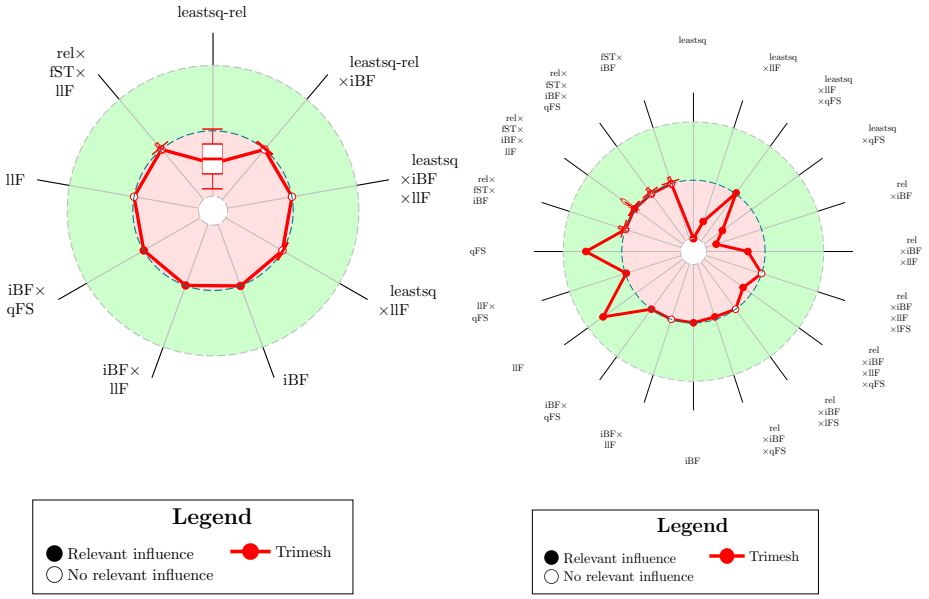
Figure A.1: Stepwise Regression and Ajstats



(a) Influences on the accuracy for Stepwise Regression and Dune.

(b) Influences on the effort for Stepwise Regression and Dune.

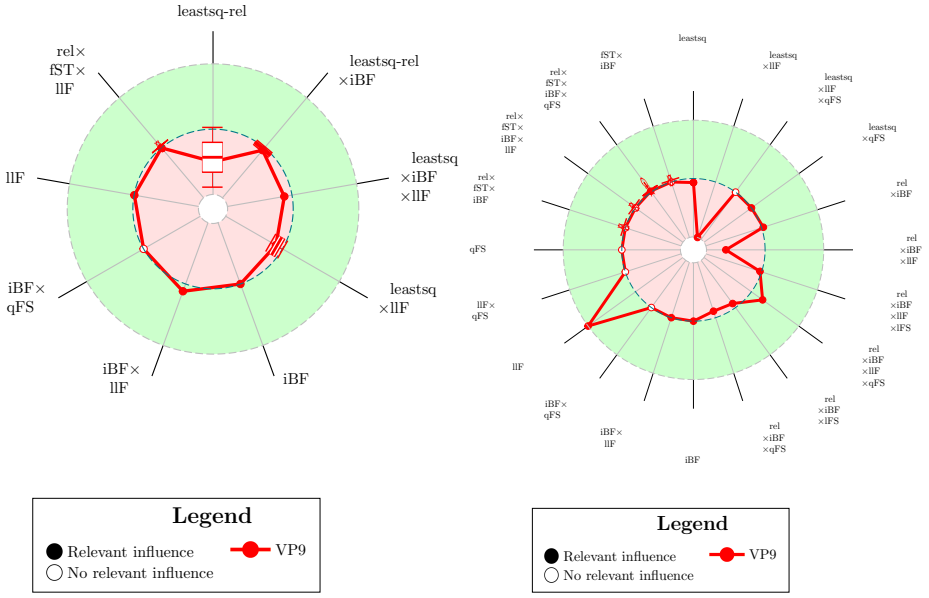
Figure A.2: Stepwise Regression and Dune



(a) Influences on the accuracy for Stepwise Regression and Trimesh.

(b) Influences on the effort for Stepwise Regression and Trimesh.

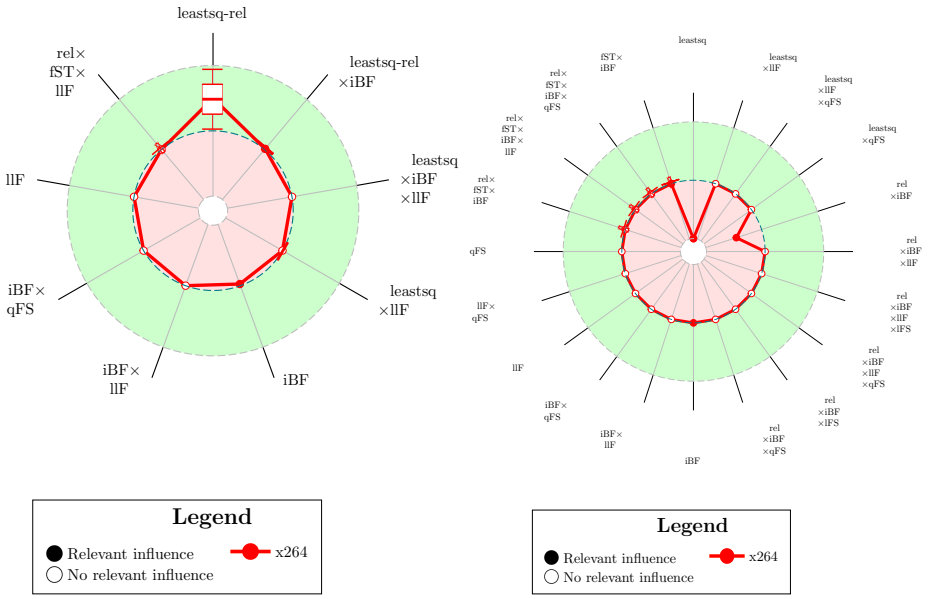
Figure A.3: Stepwise Regression and Trimesh



(a) Influences on the accuracy for Stepwise Regression and VP9.

(b) Influences on the effort for Stepwise Regression and VP9.

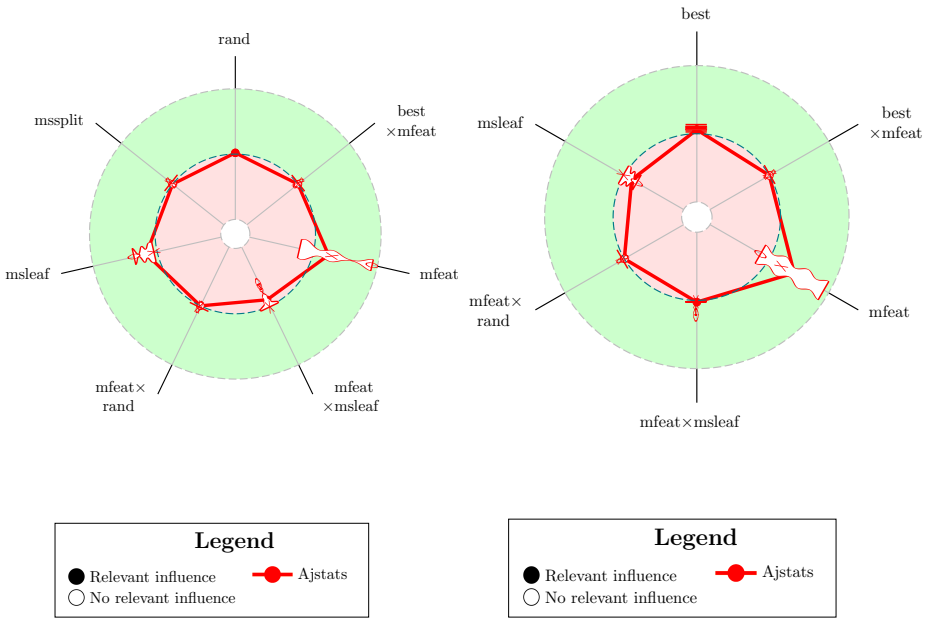
Figure A.4: Stepwise Regression and VP9



(a) Influences on the accuracy for Stepwise Regression and x264.

(b) Influences on the effort for Stepwise Regression and x264.

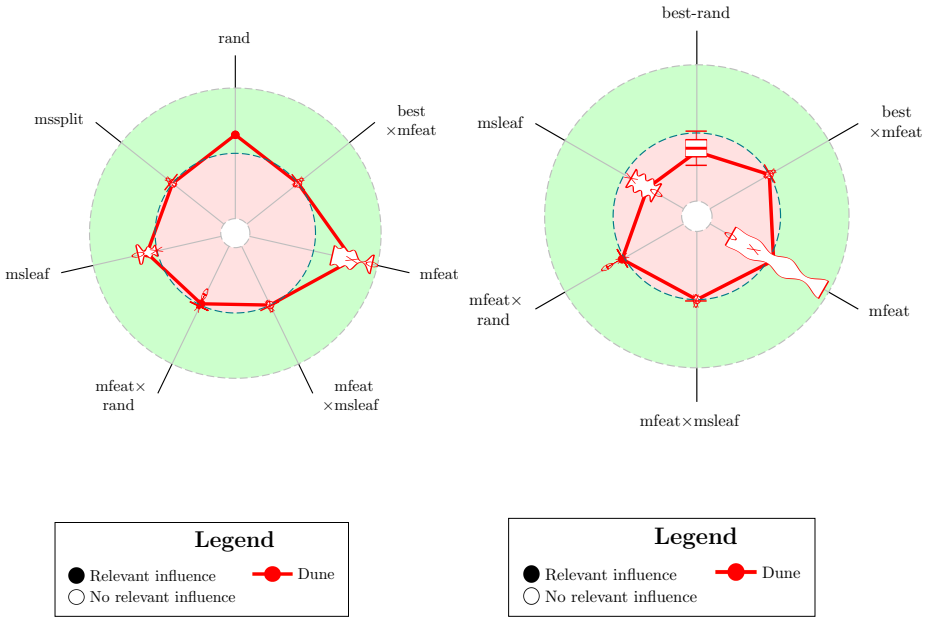
Figure A.5: Stepwise Regression and x264



(a) Influences on the accuracy for CART and Ajstats.

(b) Influences on the effort for CART and Ajstats.

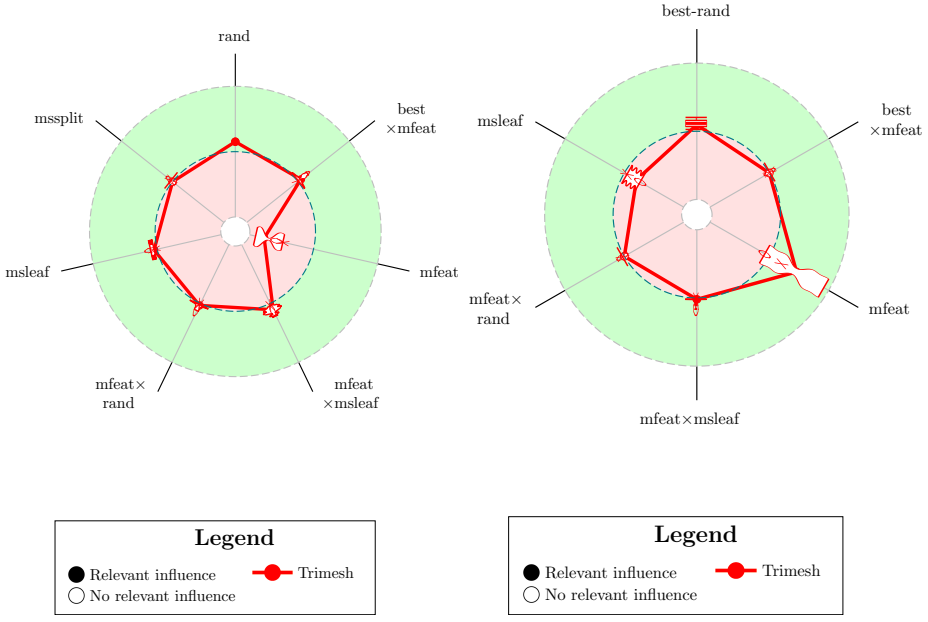
Figure A.6: CART and Ajstats



(a) Influences on the accuracy for CART and Dune.

(b) Influences on the effort for CART and Dune.

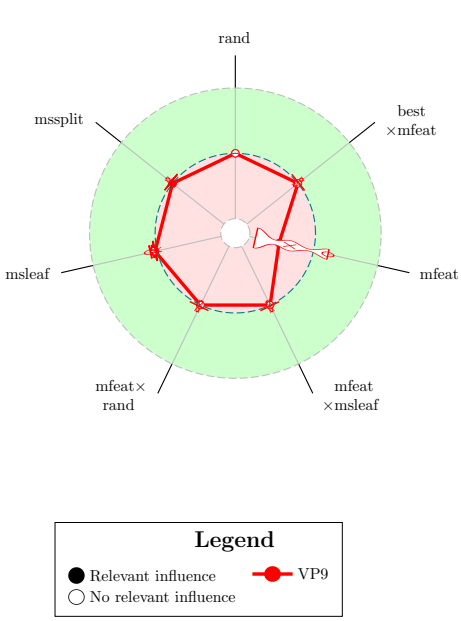
Figure A.7: CART and Dune



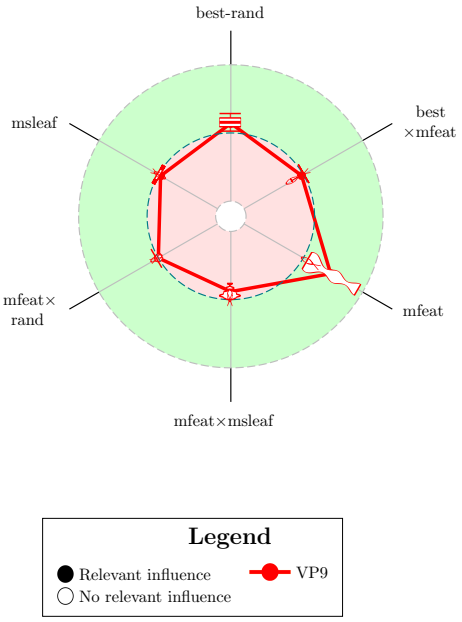
(a) Influences on the accuracy for CART and Trimesh.

(b) Influences on the effort for CART and Trimesh.

Figure A.8: CART and Trimesh

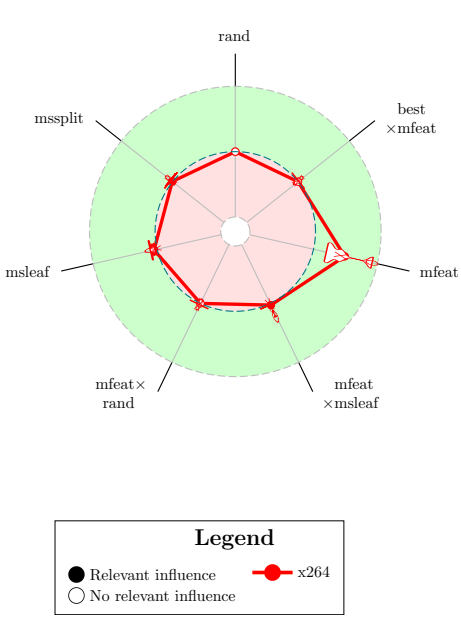


(a) Influences on the accuracy for CART and VP9.

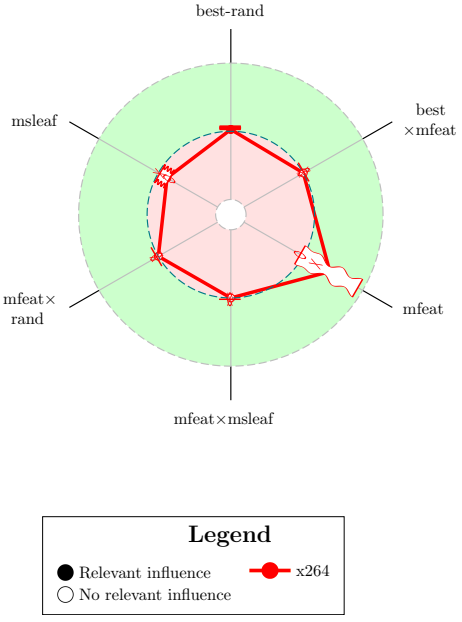


(b) Influences on the effort for CART and VP9.

Figure A.9: CART and VP9

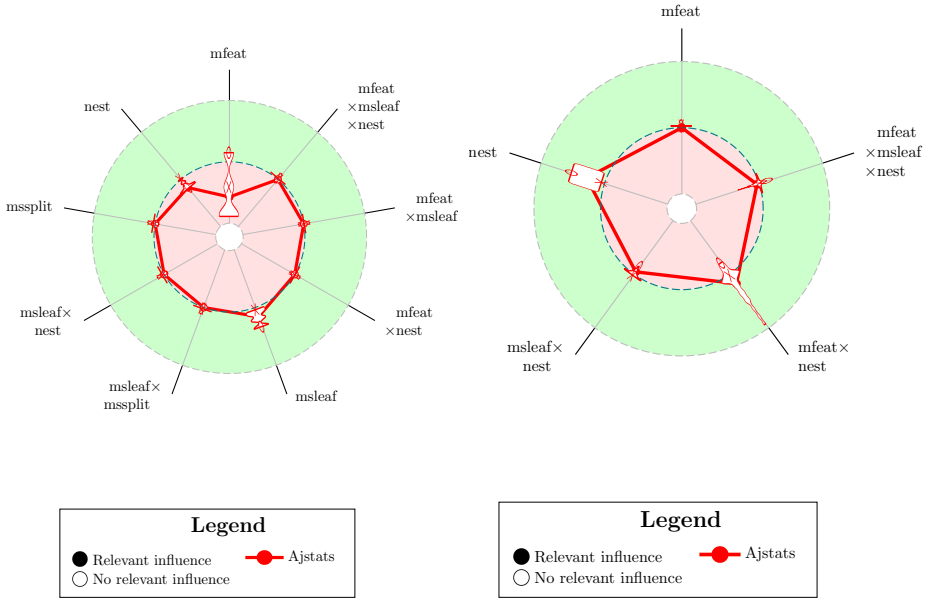


(a) Influences on the accuracy for CART and x264.



(b) Influences on the effort for CART and x264.

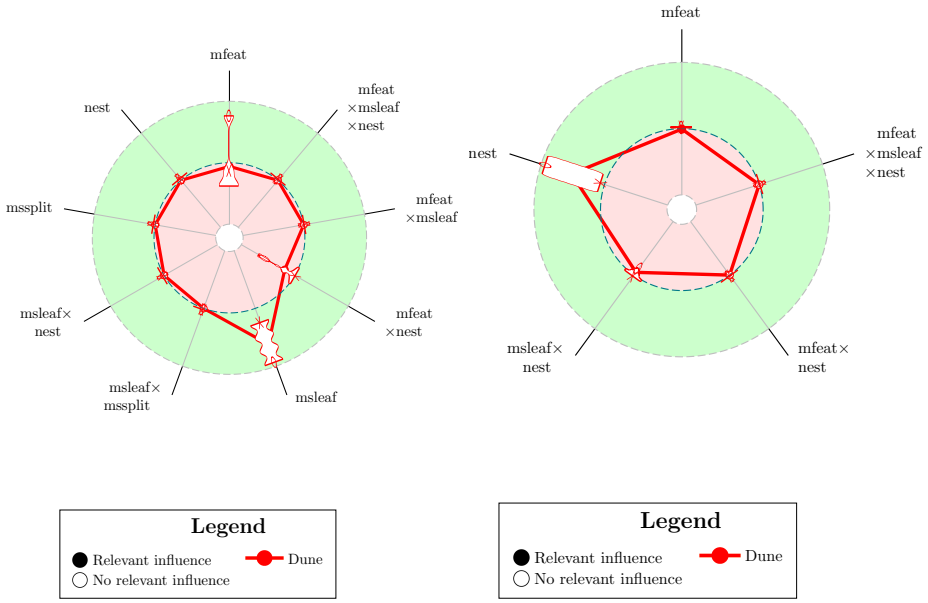
Figure A.10: CART and x264



(a) Influences on the accuracy for Random Forest and Ajstats.

(b) Influences on the effort for Random Forest and Ajstats.

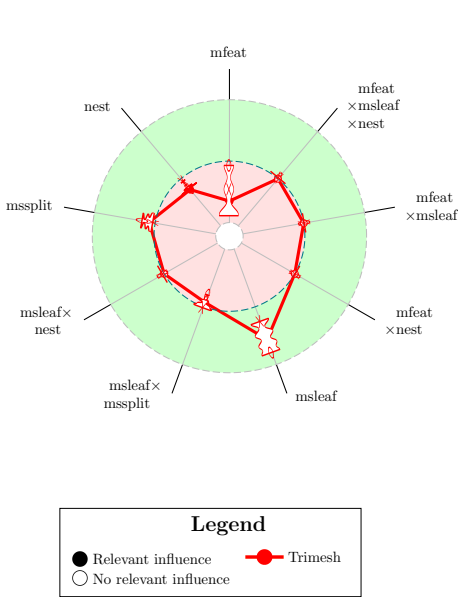
Figure A.11: Random Forest and Ajstats



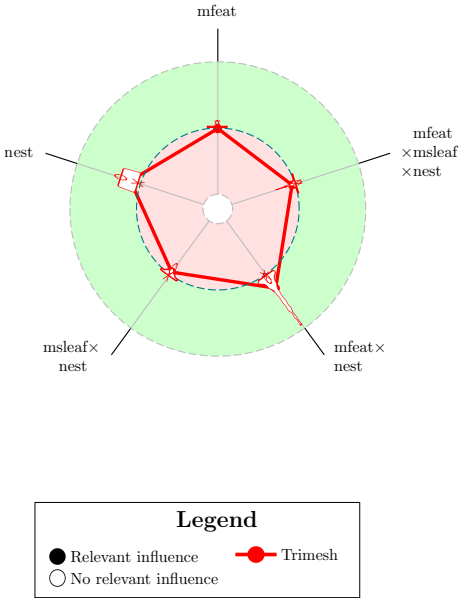
(a) Influences on the accuracy for Random Forest and Dune.

(b) Influences on the effort for Random Forest and Dune.

Figure A.12: Random Forest and Dune

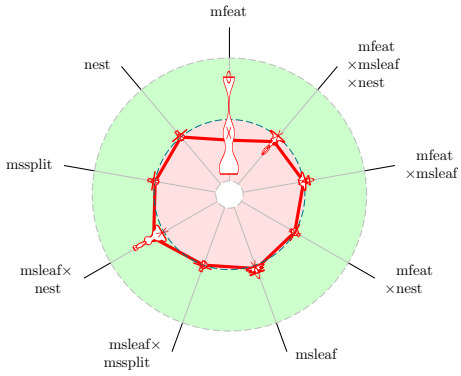


(a) Influences on the accuracy for Random Forest and Trimesh.

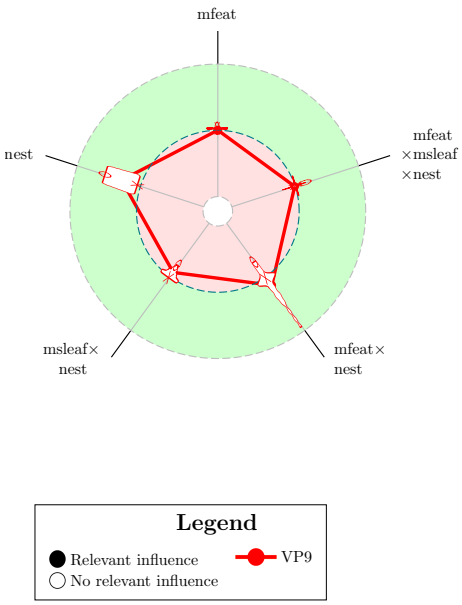


(b) Influences on the effort for Random Forest and Trimesh.

Figure A.13: Random Forest and Trimesh

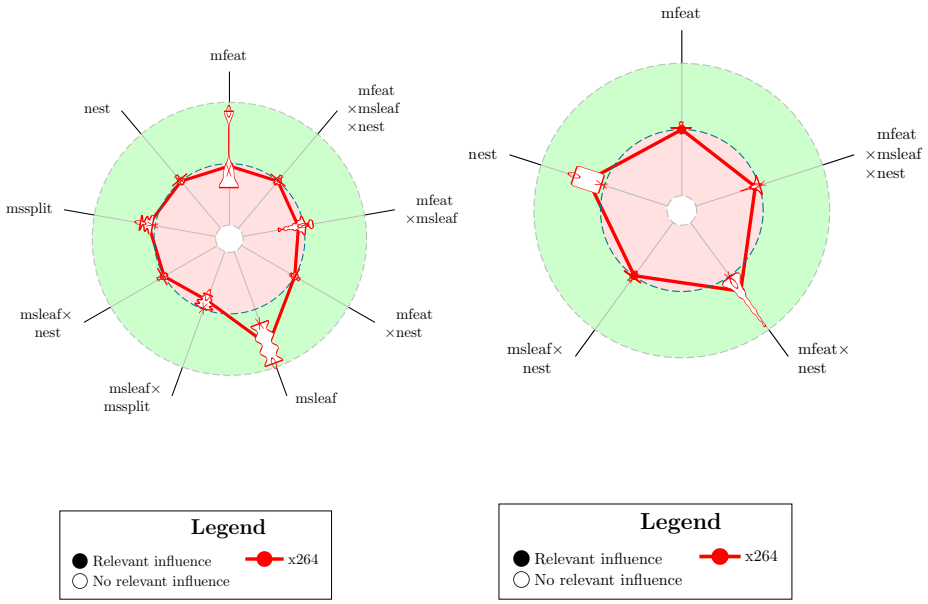


(a) Influences on the accuracy for Random Forest and VP9.



(b) Influences on the effort for Random Forest and VP9.

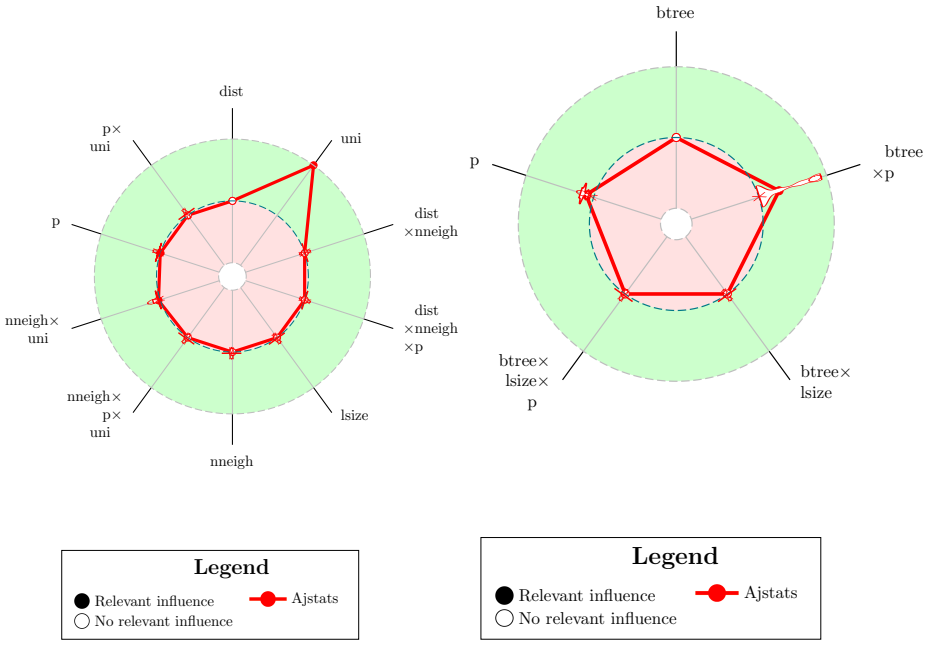
Figure A.14: Random Forest and VP9



(a) Influences on the accuracy for Random Forest and x264.

(b) Influences on the effort for Random Forest and x264.

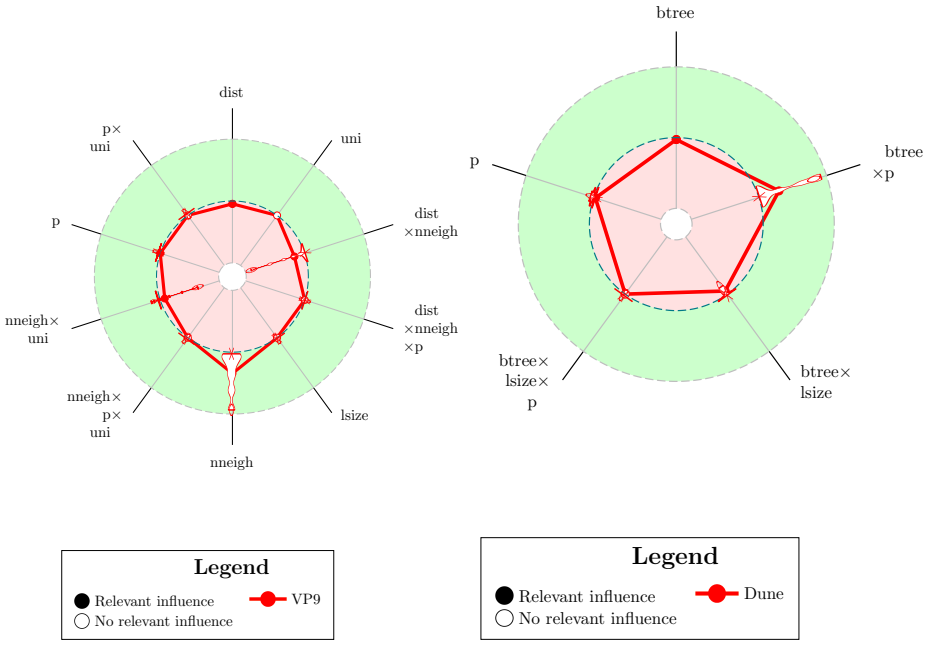
Figure A.15: Random Forest and x264



(a) Influences on the accuracy for K-Nearest Neighbors and Ajstats.

(b) Influences on the effort for K-Nearest Neighbors and Ajstats.

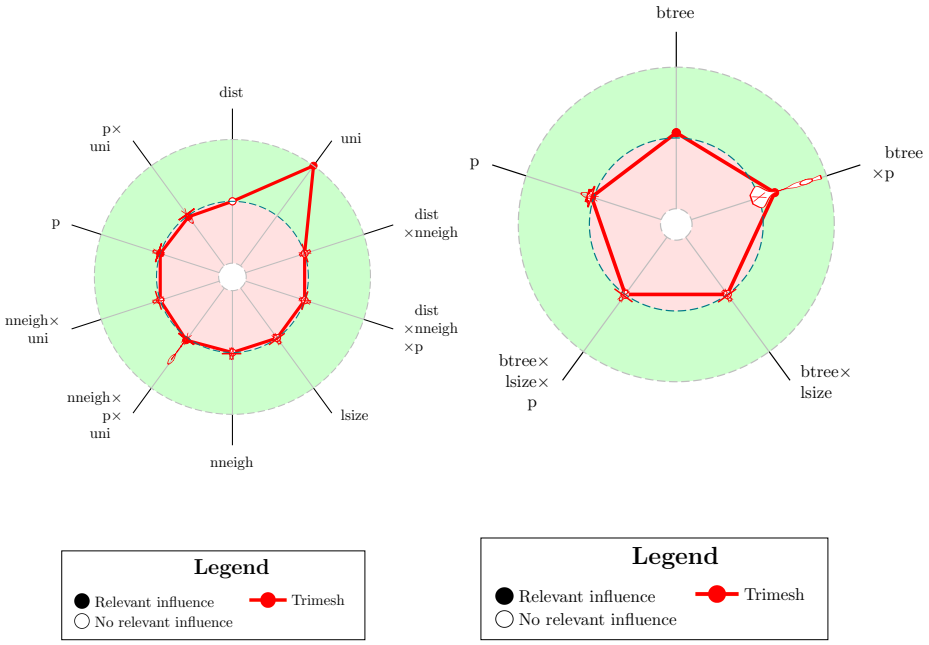
Figure A.16: K-Nearest Neighbors and Ajstats



(a) Influences on the accuracy for K-Nearest Neighbors and Dune.

(b) Influences on the effort for K-Nearest Neighbors and Dune.

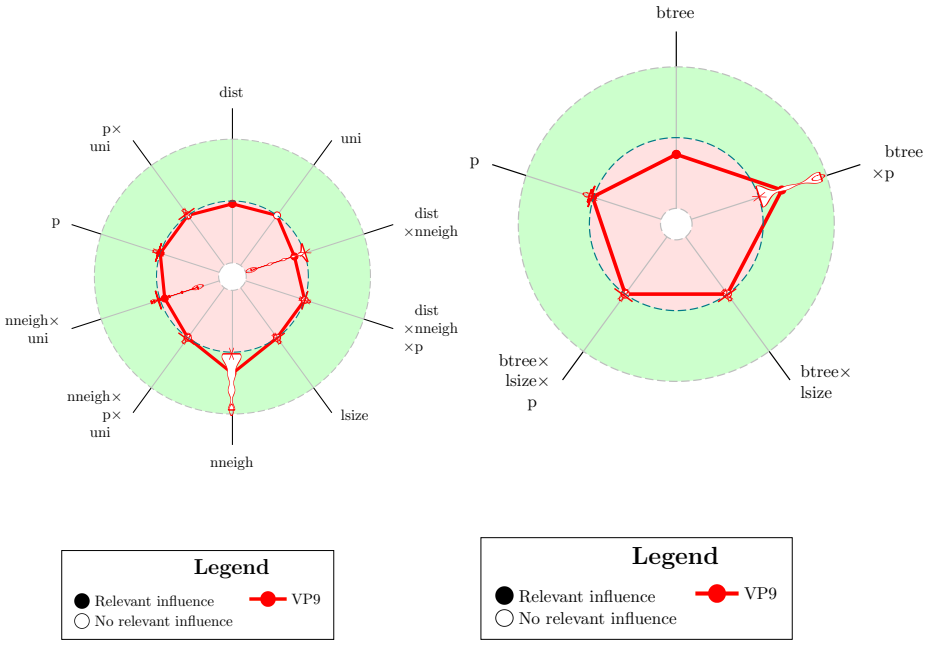
Figure A.17: K-Nearest Neighbors and Dune



(a) Influences on the accuracy for K-Nearest Neighbors and Trimesh.

(b) Influences on the effort for K-Nearest Neighbors and Trimesh.

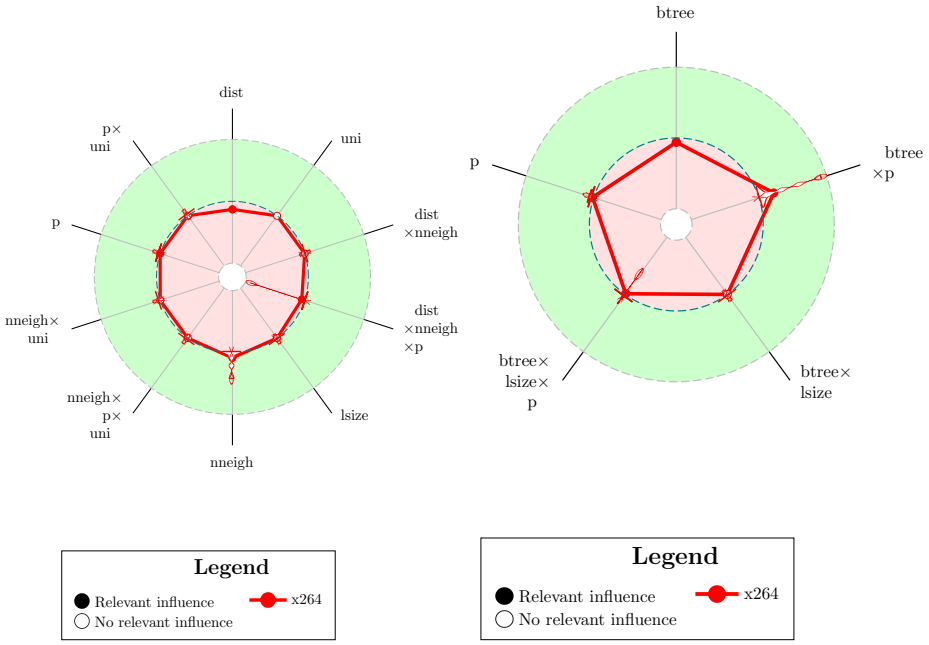
Figure A.18: K-Nearest Neighbors and Trimesh



(a) Influences on the accuracy for K-Nearest Neighbors and VP9.

(b) Influences on the effort for K-Nearest Neighbors and VP9.

Figure A.19: K-Nearest Neighbors and VP9



(a) Influences on the accuracy for K-Nearest Neighbors and x264.

(b) Influences on the effort for K-Nearest Neighbors and x264.

Figure A.20: K-Nearest Neighbors and x264

Bibliography

[ABKS16] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-oriented software product lines*. Springer, 2016. (cited on Page 5)

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012. (cited on Page 2)

[BFSO84] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984. (cited on Page 1 and 9)

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. (cited on Page 1 and 11)

[FMS16] Wei Fu, Tim Menzies, and Xipeng Shen. Tuning for software analytics: Is it really necessary? *Information and Software Technology*, 76:135–146, 2016. (cited on Page 2)

[GPS+12] Taciana AF Gomes, Ricardo BC Prudêncio, Carlos Soares, André LD Rossi, and André Carvalho. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, 75(1):3–13, 2012. (cited on Page 44)

[GRS+17] Alexander Grebhahn, Carmen Rodrigo, Norbert Siegmund, Francisco J Gaspar, and Sven Apel. Performance-influence models of multigrid methods: A case study on triangular grids. *Concurrency and Computation: Practice and Experience*, 29(17):e4057, 2017. (cited on Page 7)

[GYS+18] Jianmei Guo, Dingyu Yang, Norbert Siegmund, Sven Apel, Atrisha Sarkar, Pavel Valov, Krzysztof Czarnecki, Andrzej Wasowski, and Huiqun Yu. Data-efficient performance learning for configurable systems. *Empirical Software Engineering*, 23(3):1826–1867, 2018. (cited on Page 43)

[HDO+98] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998. (cited on Page 1)

[IB13] Sadegh Bafandeh Imandoust and Mohammad Bolandraftar. Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background. *International Journal of Engineering Research and Applications*, 3(5):605–610, 2013. (cited on Page 12)

[LAL+10] Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. An analysis of the variability in forty preprocessor-based software product lines. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 105–114. ACM, 2010. (cited on Page 37)

[LBG15] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: a survey of trends and technologies. *Artificial intelligence review*, 44(1):117–130, 2015. (cited on Page 6)

[LSB+10] Rafael Lotufo, Steven She, Thorsten Berger, Krzysztof Czarnecki, and Andrzej Wąsowski. Evolution of the linux kernel variability model. *Software Product Lines: Going Beyond*, pages 136–150, 2010. (cited on Page 1)

[RSD12] Matthias Reif, Faisal Shafait, and Andreas Dengel. Meta-learning for evolutionary parameter optimization of classifiers. *Machine learning*, 87(3):357–380, 2012. (cited on Page 44)

[SBK04] Carlos Soares, Pavel B Brazdil, and Petr Kuba. A meta-learning method to select the kernel width in support vector regression. *Machine learning*, 54(3):195–209, 2004. (cited on Page 7 and 43)

[SGAK15] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 284–294. ACM, 2015. (cited on Page 1, 7, 8, 43, and 46)

[VBPH12] Joaquin Vanschoren, Hendrik Blockeel, Bernhard Pfahringer, and Geoffrey Holmes. Experiment databases. *Machine Learning*, 87(2):127–158, 2012. (cited on Page 7 and 44)

[WHKF12] Dennis Westermann, Jens Happe, Rouven Krebs, and Roozbeh Farahbod. Automated inference of goal-oriented performance prediction functions. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 190–199. ACM, 2012. (cited on Page 43)

[XJF+15] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 307–319. ACM, 2015. (cited on Page 1)

Eidesstattliche Erklärung:

Hiermit versichere ich an Eides statt, dass ich diese Bachelorarbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und dass alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie dass ich die Bachelorarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Fleischmann Bastian

Passau, den 24. Juli 2018