Ludwig-Maximilians-University of Munich

Faculty of Mathematics, Informatics and Statistics

BACHELOR THESIS

Hyperparameter dependent modelling of runtime behaviour of machine learning algorithms

Maria Erdmann

Department of Statistics

Supervisor of the bachelor thesis: Prof. Bernd Bischl

Study programme: Statistics (Bachelor)

Munich, 31 August 2016

Statutory Declaration

I declare that I have developed and written the enclosed Bachelor's Thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Bachelor's Thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

Munich, 31 August 2016

.....

Maria Erdmann

Abstract

In the age of big data, the analysis of data with accurate and efficient algorithms has become indispensable. Algorithm analysis and portfolio-based algorithm selection are important applications in order to find appropriate algorithms for an unknown problem. The suitability of an algorithm depends on its performance, in terms of prediction accuracy. However, experimental procedures with these algorithms are often computational demanding. Thus, the prediction and modelling of runtime are affecting algorithm selection. Therefore runtime becomes a performance measure. It can be split into two parts: the training time, which indicates, how long it takes to train an algorithm on a certain dataset; and the prediction time, which indicates, how long it takes to perform a prediction. In this thesis, runtime of machine learning algorithms is modelled separately for training and prediction time. Besides dataset characteristics, the influence of the hyperparameter values of an algorithm on runtime is analysed. The concept of meta-learning, which is an approach to predict the suitability of an algorithm, are applied to relate the dataset characteristics and the hyperparameter values of the algorithm to runtime.

The empirical study considered 6 classification algorithms, 65 dataset from OpenML, and 4 different regression models, with a focus on gradient boosting with componentwise linear models. For comparing gradient boosting with smooth components, a generalised linear model and random forest was used.

The experimental results demonstrate that the best prediction performance on training data is achieved by random forest. Moreover, the prediction results of all models are better than a commonly used baseline. The results for gradient boosting algorithms advise using more sophisticated variable selection methods in order to get sparser models. Overall, the concept of meta-learning is a valid method to predict runtime. This encourages using this method to its full extent for further developments, including methods to generalise prediction accuracy of the regression methods for future instances.

Contents

1 Introduction **1**

2 Theoretical Background **4**
2.1 Classification algorithms 4
2.1.1 Naive Bayes 5
2.1.2 Recursive Partitioning 5
2.1.3 Random forest 6
2.1.4 Lasso and elastic-Net regularised generalised linear models . 6
2.1.5 Generalised boosting models 7
2.1.6 Neural networks with a single hidden layer 8
2.2 Regression methods for modelling runtime 9
2.2.1 Model-based boosting 9
2.3 Meta-learning 14

3 Methods **15**
3.1 Data and data generating process 15
3.1.1 Collecting datasets from OpenML 15
3.1.2 Construction and implementation of the experiments 16
3.2 Statistical analysis 18

4 Results **22**
4.1 Explorative Analysis 22
4.2 Model results 24

5 Conclusion **37**

Appendices **40**

A Graphs **41**

B Tables **67**

6 Structure of the repository **81**

List of Figures

2.1 Structure of a multi-layer perceptron 8

3.1 Meta-learning approach for modelling runtime 16

4.1 Boxplots of the distribution of runtimes and the respective coefficient of variation for each classifier 23
4.2 Boxplot of the distribution of the mean misclassification error and of its respective coefficient of variation for each classifier 24
4.3 Quantitative comparison of the regression models' prediction accuracy on training data for each classifier. The straight line is the bisector of the coordinate axes. For a perfect fit all data points lie on that line. 27
4.4 Combined functional estimates of glm modelling training time . . . 34

A.1 Partial effects obtained from glmboost training time for classifier naiveBayes 41
A.2 Partial effects obtained from glmboost prediction time for classifier naiveBayes 41
A.3 Partial effects obtained from glmboost training time for classifier rpart 42
A.4 Partial effects obtained from glmboost prediction time for classifier rpart 42
A.5 Partial effects obtained from glmboost training time for classifier ranger 43
A.6 Partial effects obtained from glmboost prediction time for classifier ranger 43
A.7 Partial effects obtained from glmboost training time for classifier glmnet 44
A.8 Partial effects obtained from glmboost prediction time for classifier glmnet 44
A.9 Partial effects obtained from glmboost training time for classifier gbm 45
A.10 Partial effects obtained from glmboost prediction time for classifier gbm 45
A.11 Partial effects obtained from glmboost training time for classifier nnet 46

A.12 Partial effects obtained from glmboost prediction time for classifier nnet . . . 46
A.13 Partial effects obtained from gamboost on training time for classifier naiveBayes . . . 47
A.14 Partial effects obtained from gamboost on prediction time for classifier naiveBayes . . . 48
A.15 Partial effects obtained from gamboost on training time for classifier rpart . . . 49
A.16 Partial effects obtained from gamboost on prediction time for classifier rpart . . . 50
A.17 Partial effects obtained from gamboost on training time for classifier glmnet . . . 51
A.18 Partial effects obtained from gamboost on prediction time for classifier glmnet . . . 52
A.19 Partial effects obtained from gamboost on training time for classifier gbm . . . 53
A.20 Partial effects obtained from gamboost on prediction time for classifier gbm . . . 54
A.21 Partial effects obtained from gamboost on training time for classifier nnet . . . 55
A.22 Partial effects obtained from gamboost on prediction time for classifier nnet . . . 56
A.23 Variable importance plot of randomForest modelling training time for classifier naiveBayes . . . 57
A.24 Variable importance plot of randomForest modelling prediction time for classifier naiveBayes . . . 57
A.25 Variable importance plot of randomForest modelling training time for classifier rpart . . . 57
A.26 Variable importance plot of randomForest modelling prediction time for classifier rpart . . . 58
A.27 Variable importance plot of randomForest modelling training time for classifier ranger . . . 58
A.28 Variable importance plot of randomForest modelling prediction time for classifier ranger . . . 58
A.29 Variable importance plot of randomForest modelling training time for classifier glmnet . . . 59
A.30 Variable importance plot of randomForest modelling prediction time for classifier glmnet . . . 59
A.31 Variable importance plot of randomForest modelling training time for classifier gbm . . . 59
A.32 Variable importance plot of randomForest modelling prediction time for classifier gbm . . . 60
A.33 Variable importance plot of randomForest modelling training time for classifier nnet . . . 60

A.34 Variable importance plot of randomForest modelling prediction time for classifier nnet . . . 60
A.35 Partial dependence plot of randomForest modelling training time for classifier naiveBayes . . . 61
A.36 Partial dependence plot of randomForest modelling prediction time for classifier naiveBayes . . . 61
A.37 Partial dependence plot of randomForest modelling training time for classifier rpart . . . 62
A.38 Partial dependence plot of randomForest modelling prediction time for classifier rpart . . . 62
A.39 Partial dependence plot of randomForest modelling training time for classifier ranger . . . 63
A.40 Partial dependence plot of randomForest modelling prediction time for classifier ranger . . . 63
A.41 Partial dependence plot of randomForest modelling training time for classifier glmnet . . . 64
A.42 Partial dependence plot of randomForest modelling prediction time for classifier glmnet . . . 64
A.43 Partial dependence plot of randomForest modelling training time for classifier gbm . . . 65
A.44 Partial dependence plot of randomForest modelling prediction time for classifier gbm . . . 65
A.45 Partial dependence plot of randomForest modelling training time for classifier nnet . . . 66
A.46 Partial dependence plot of randomForest modelling prediction time for classifier nnet . . . 66

List of Tables

2.1 Boosting algorithm as implemented in `gbm` 7

3.1 Summary of dataset characteristics 16

4.1 Average runtimes of the five classifiers 22
4.2 Relative absolute of the four regression models for each target classifier 25
4.3 Root mean squared error of the four regression models for each target classifier 25
4.4 Results of variable selection: excluded variables from models on training time 29
4.5 Results of variable selection: excluded variables from models on prediction time 30

B.1 Collection of datasets from OpenML 67
B.2 Investigated classifiers and their hyperparameters 69
B.3 Number of samples in the meta-datasets used within the evaluation of the target classifiers 71
B.4 Number and types of errors obtained during the data-generating process 72
B.5 Number of expired jobs 72
B.6 Summary of runtimes and mean misclassification error 73
B.7 Summary of coefficient of variation of runtime and mean misclassification error 74
B.8 Estimated coefficients of the glmboost and the glm model for target classifier naiveBayes 75
B.9 Estimated coefficients of the glmboost and the glm model for target classifier rpart 76
B.10 Estimated coefficients of the glmboost and the glm model for target classifier ranger 77
B.11 Estimated coefficients of the glmboost and the glm model for target classifier glmnet 78
B.12 Estimated coefficients of the glmboost and the glm model for target classifier gbm 79
B.13 Estimated coefficients of the glmboost and the glm model for target classifier nnet 80

1 Introduction

In the past decade industrial and academic fields have changed focus from data gathering to data analysis (Priya et al., 2011). Having collected 90% of the current data in the last couple of years, the main challenge nowadays is analysing these data. A central question arises from this task: Which algorithm out of a plethora of algorithms should be used for a given problem instance? Answering this question often requires costly empirical processes or deep expert knowledge, particularly, if several problems need to be solved in one analysis (Priya et al., 2012).

The suitability of an algorithm for a distinct problem depends mostly on the algorithm's performance in terms of prediction accuracy. Besides, the computational efficiency of an algorithm is increasingly taken into account. Thus, computational efficiency – also referred to as runtime – becomes a performance measure, indicating how long it will take to train a specific algorithm with its proprietary set of parameters and how long it will take to perform a prediction. The reasons, why runtime becomes essential for the decision on an appropriate algorithm, are manifold.

These days, data-based decisions and data-driven optimisation of services for customers play an increasingly important role to most companies. The speed, at which data needs to be available, and at which analyses need to be implemented, plays a crucial role for companies, for example, when the aim is to predict customer behaviour in real-time. At the same time, companies try to keep expenditures for computational resources minimal (Reif et al., 2011). But also academics, scientists and data analysts are confronted with limited time and available resources.

Another important aspect arises from the need for resource and workload management (also known as "scheduling") in shared environments like high performance clusters. In such an environment many different users implement their experiments and analyses, and thus, available resources need to be shared fairly among the users. This task is undertaken by a scheduler. It initially queues the production jobs and executes, them whenever the requested resources are available. These resources are specified by runtime and memory. Therefore, each user is responsible to determine the resources of his production jobs in advance (Leibniz-Rechenzentrum, 2015). Using a predicted runtime, enhances job scheduling and avoids termination of jobs before completion. Moreover, overall execution time can be minimised (Priya et al., 2011). In addition, predicting runtime facilitates

planning of experiments and work in general.

Last but not least, there is also an environmental aspect behind runtime prediction. Since trial-and-error experiments are time consuming, they consume a great amount of energy, and thus, lead to high $C0_2$ emissions. Therefore, Al-Jarrah et al. (2015) proposes the "sustainable data modelling". This approach aims to maximise learning learning accuracy, while minimising computational costs at the same time.

In summary, the goal of every data mining project is to be as accurate and as efficient as possible (Doan and Kalita, 2016). This implies finding appropriate machine learning algorithms in terms of accuracy and efficiency with respect to the problem that needs to be analysed. In practice, there is often a trade-off between high prediction accuracy and low computational costs. Thus, a major goal in the field of runtime analysis is to find an algorithm technique that chooses the best algorithm in terms of high prediction accuracy and a small runtime for a given problem. Hence, accuracy and runtime need to be predicted at the same time. However, prediction of runtime with regression methods is still a young science area, with its beginnings dating back to the mid 1990s (Hutter et al., 2014).

Priya et al. (2011) apply the approach of meta-learning by relating dataset characteristics and the current machine state to the actual runtime. Runtime of 6 classification algorithms was assessed on 78 publicly available datasets. Four regression models (linear regression, decision tree M5P, k-nearest neighbour, support vector machine) were used to predict runtime and the prediction quality was evaluated using Leave-One-Out Cross Validation (LOOCV). Prediction performance of the models was evaluated by comparing the mean absolute error (MAD) to a commonly used baseline. The lowest MAD values were obtained for support vector machines (SVM) and k-nearest neighbours (K-NN).

In a subsequent study Priya et al. (2012) further investigated SVMs for runtime prediction. They optimised two SVMs, each with a different kernel, with a genetic algorithm that performed joint Feature Subset Selection (FSS) and Parameter Optimisation (PO). With this method predictions improved significantly. A combination of both SVMs to one regression method performed similar to the genetically evolved methods.

Another slightly different approach was chosen by Reif et al. (2011). Here, the meta-learning approach was used to estimate the time needed for a grid search over multiple parameters, because, in most applications, a time consuming search for an optimal hyperparameter set precedes the actual modelling process. For prediction of runtime different meta-features related to the dataset and algorithm were assessed, and their influence analysed. In order to account for the performance of the user's computer, computation time of some meta-features was measured and then stored together with the other meta-features in the meta-dataset.

The most recent research on runtime prediction comes from Doan and Kalita (2016) and is structurally similar to Priya et al. (2011). This study was conducted

using 50 datasets, 28 classification algorithms and 8 regression models for predicting CPU-time. Meta-features that were used for the prediction model included different dataset characteristics. The prediction performance of the regression models were compared to each other by root mean squared error (RMSE), mean absolute error (MAE) and MAD. The results demonstrate that multivariate adaptive regressions splines (MARS) is a good solution for predicting CPU-times.

Research from Hutter et al. (2014) includes a broad review on literature on runtime research, as well as an investigation of random forests, neural networks, ridge regression and newer methods like ridge regression with forward-backward selection (SPORE-FoBa) and gaussian process regression for prediction of runtime. The performance of the regression methods was assessed by 10-fold cross-validation and calculating the RMSE. Additionally, new instance features for propositional satisfiability (SAT), travelling salesperson (TSP) and mixed integer programming (MIP) problems were investigated. The prediction methods and new features were evaluated for 11 algorithms and 35 instances distributions spanning a wide range of combinatorial problems (SAT, TPS, MIP). Among a series of results, they found out that random forest is the best method for predicting new instances, as well as for predicting new instances and new configurations. Hyperparameter optimisation of the prediction models improved prediction accuracy slightly, but the computational costs were drastically higher, compared to the respective improvement of performance.

In this thesis, runtime behaviour of six different classification algorithms is analysed. Since runtime depends on the dataset and the hyperparameter values of an algorithm (Reif et al., 2011), dataset properties and the algorithm's hyperparameters are going to be the features used to predict runtime by using a meta-learning approach. This includes three parts: 1) Measuring runtime of the 6 classifiers on different datasets with different hyperparameter settings; 2) the generation of a meta-dataset by merging the characteristics of the dataset, the hyperparameters of the algorithm and the runtime and; 3) applying several regression methods to the meta-dataset in order to predict runtime. Four regression modelling techniques are described and compared to each other with a focus on gradient boosting with component-wise linear models and gradient boosting with smooth effects. The other regression methods include a generalised linear model and a random forest, as a non-parametric approach.

The thesis is organised as follows: chapter 2 provides a short description of the six classifiers, and an overview of the regression methods used, with a focus on model-based boosting. Moreover, the concept of meta-learning is introduced. Chapter 3 gives a detailed description of the experimental approach. In chapter 4 the results of this approach are presented. Chapter 5 summarises and discusses these results, and provides an outlook for further research.

2 Theoretical Background

2.1 Classification algorithms

The aim of machine learning is to develop and employ techniques that identify patterns and regularities in data, in order to extract useful information from that data (Priya et al., 2011). Depending on the type of available data, machine learning is usually categorised into supervised and unsupervised learning. In supervised learning, data consists of examples. Each example refers to one observation and comprises the predictor vector (or input vector) and an associated response (or output). A machine learning algorithm learns from that data and relates the response to the predictors. Supervised learning aims to make predictions for future observations, based on the predictors, and to understand the underlying relationship between response and predictors (inference). In contrast, unsupervised learning uses a vector of predictors for each observation but no associated response (James et al., 2015).

Supervised learning distinguishes between classification and regression problems. Whenever the response is qualitative (categorical), taking values in k different classes, this is denoted to be a classification problem. If the response variable is quantitative, hence, is taking numerical values, it is a regression problem. Accordingly, classification algorithms (classifiers) solve classification problems and regression algorithms (regressors) solve regression problems by predicting numerical data (James et al., 2015).

For many algorithms the computational complexity is known. It is expressed with the Big O notation, which is a formal way to express asymptotic behaviour of an algorithm with respect to the number of inputs and its hyperparameters (Louppe, 2014). In these theoretical considerations, constant terms are neglected and the practical benefit in the application of algorithms is often limited (Reif et al., 2011).

In this thesis, runtime of six classification algorithms was analysed and four regression methods were compared to each other. The following chapters provide a short overview of the six classification algorithms and an introduction into model-based boosting, which is the main regression method for analysing runtime in this thesis. Whenever possible, computational complexity of the classifiers is specified.

2.1.1 Naive Bayes

The naive Bayes classifier is an effective and fast classification technique (Friedman et al., 1997). It computes conditional a-posteriori probabilities for each class of a categorical response variable by using the Bayes' theorem (Tutz, 2012; Meyer et al., 2015). The naive Bayes classifiers assumes independence of the predictor variables for a given class, and is therefore called "naive". Usually, this assumption is not true in general real-world applications. However, the naive Bayes classifier is a popular method for high-dimensional data and for multiclass prediction. Often this classifier performs better than sophisticated alternatives (Hastie et al., 2016). However, dealing with a categorical variable is problematic, if the test set includes a category that was not observed in the training set. In this case, the classifiers assigns a zero probability. This problem can be solved with smoothing techniques like the Laplace estimation (Sunil, 2015).

In this thesis, `naiveBayes` from the R package `e1071` (Meyer et al., 2015) is used as an application for the naive Bayes classifier.

For the naive Bayes classifer, time complexity for training is essentially optimal. It is $O(np)$ with n being the number of instances in the training set, and p being the number of features. Thus, time complexity is independent of the values the features can take (Elkan, 1997).

2.1.2 Recursive Partitioning

The R package `rpart` (Therneau et al., 2015) provides recursive partitioning for classification, regression and survival trees, and implements, in general, the concept of classification and regression trees (CARTs), as described by Breiman et al. (1998). The principle of the method is simple: the feature space is partitioned into a set of rectangles such, that they are as homogeneous as possible with respect to the response variable. On each rectangle a simple model is fitted (Hastie et al., 2016).

The tree building process starts with partitioning the whole feature space (root node) by a single variable and an associated split-point. The data is now separated into two subsets (child nodes) and the process is recursively applied again. This is repeated until some stopping criteria is met. Each partition of a node into two child nodes is determined by one variable and one split point. The features and the split-point that partitions a node is searched by exhaustive search: the algorithm iterates over all features and all split-points to find the best feature and the best split-point according to a split-criterion. Split-criteria can be approaches based on test statistics or impurity measures, like the Gini index (Tutz, 2012).

Computational complexity of training a decision tree without pruning is $O(np^2)$ (Elkan, 1997). Another specification of time complexity is made by Louppe (2014), where the overall within-node complexity is $O(pn\,log(n))$.

2.1.3 Random forest

Decision trees, as introduced in the preceding chapter, have some valuable advantages: they can easily be interpreted and visualised, can handle missing data and can be applied on high-dimensional data. On the contrary, they are unstable, meaning that a small change in data can lead to a totally different tree. Therefore, their predictive accuracy is often inferior in comparison to other methods. This can be overcome by aggregating many decision trees, which leads to methods like bagging, random forest, and boosting.

A random forest is an ensemble method, which means that various predictors are aggregated (Tutz, 2012). The prediction of random forests is based on a combination of several decision trees, leading to as many predictions as there are trees in the ensemble. The final prediction is obtained, for example, by majority vote for the most common class among all predictions.

Time complexity of random forest, as proposed by Breiman (2001) is $O(kn\,log(n))$ (Louppe, 2014). In this thesis, R package `ranger` (Wright, 2016), a fast implementation of random forests for high-dimensional data is used. Besides analysing the runtime behaviour of `ranger`, random forest is also used as regression model for predicting runtime. For this purpose, the R package `randomForest` (Liaw and Wiener, 2002) was employed.

2.1.4 Lasso and elastic-Net regularised generalised linear models

`glmnet` (Friedman et al., 2010a) is a package that fits generalised linear models that combine L1-norm and L2-norm penalisation. The following problem is solved for different values of λ:

$$\min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda[(1-\alpha)\|\beta\|_2^2/2 + \alpha\|\beta\|_1]$$

$w_i l(y, \beta_0 + \beta^T x)$ is the weighted negative log-likelihood contribution for observation i and α is the elastic-net penalty. Setting α to one (the default) leads to lasso regression, a regression model that is penalised with the L1-norm, thus, with the sum of absolute coefficients. Hence, coefficient values are shrunken, and explanatory variables that are minor to the response become zero. On the other hand, setting α to zero creates the ridge regression model, a regression model that is penalised with the L2-norm, which is the sum of squared coefficients. This also leads to shrinkage of the coefficients of correlated predictors. But in contrast to lasso regression, the coefficients of the explanatory variables with minor contribution to the response get close to zero. The elastic-net penalty can combine penalisation with L1- and L2-norm. Thus, coefficients can be shrunken effectively – as in ridge regression – and at the same time some coefficients are set to zero – as in lasso regression

(Brownlee, 2014). The parameter λ controls the overall strength of penalty. The `glmnet` algorithm is fast and fits linear, logistic, multinomial, poisson, Cox regression and multi response linear regression models. Regarding computational complexity Friedman et al. (2010b) state, that computation is "roughly linear in n, but grows faster than linear in p".

2.1.5 Generalised boosting models

Gradient boosting, as it is implemented in the R package `gbm` Ridgeway (2015), is a tree-based method. Particularly, `gbm` is an implementation and extension of the AdaBoost algorithm of Freund and Schapire (1997) and the gradient boosting machine of Friedman (2000). In the following brief description of the algorithm the important hyperparameters of `gbm` are integrated whenever possible and highlighted by typewriter font. The algorithm is described in 2.1.

Similar to random forest, a defined number of trees (`n.trees`) are grown. But instead of applying bagging, the trees in boosting are grown sequentially, and each tree uses the information from the previously grown tree. The trees are built on a subset of `bag.fraction` * n cases of the dataset and are fitted to the negative gradient of the loss function, also called working response. The size of the tree is determined by `interaction.depth`, which is the number of terminal nodes that needs to be specified. Since `interaction.depth` is a small integer, only small trees are built, which leads to slow improvements of the function estimate $\hat{f}$, when the new decision tree is added to the fitted function (see step 2d of the algorithm 2.1). Additionally, the `shrinkage` parameter slows down the process. The selection of the `shrinkage` parameter λ depends on the task. Choosing small values for λ might require specifying high values for `n.tree`. Moreover, finding an appropriate number of trees should be assessed by cross-validation to prevent overfitting.

Table 2.1: Boosting algorithm adopted from Hastie et al. (2016) and Ridgeway (2007)

Boosting algorithm as implemented in `gbm`
1. Initialize $\hat{f}(x)$ to be constant.
2. For $b = 1, 2, \ldots,$ `n.trees` apply:
a) Compute the negative gradient of the loss function as working response.
b) Randomly select `bag.fraction` * n cases from the dataset
c) Fit a tree $\hat{f}^b$ to the working response with k (= `interaction.depth`) terminal nodes with the randomly selected observations from step b)
d) Update $\hat{f}$ adding the new tree, that is shrunken with `shrinkage` parameter λ: $\hat{f}(x) = \hat{f}(x) + \lambda \hat{f}^b(x)$
3. Output the boosted model: $\hat{f}(x) = \sum_{b=1}^{\texttt{n.trees}} \lambda \hat{f}^b(x)$

Another approach of gradient boosting is model-based boosting, which is implemented in the R package `mboost` (Hothorn et al., 2016). Since this is one of the regression methods for prediction of runtime, a more detailed description of this model is given in 2.2.1.

In boosting, `n.trees` decision trees are fitted and the computational complexity of one decision tree is $O(np^2)$ (Elkan, 1997). Hence, computational complexity of the `gbm` algorithm is $O(Tnp^2)$ with T being the number of trees.

2.1.6 Neural networks with a single hidden layer

A neural network is a regression method inspired by the information processing in the human brain. One of the most popular representatives of neural networks is the multi-layer perceptron (MLP) with one hidden layer. Figure 2.1 shows the structure of the MLP.

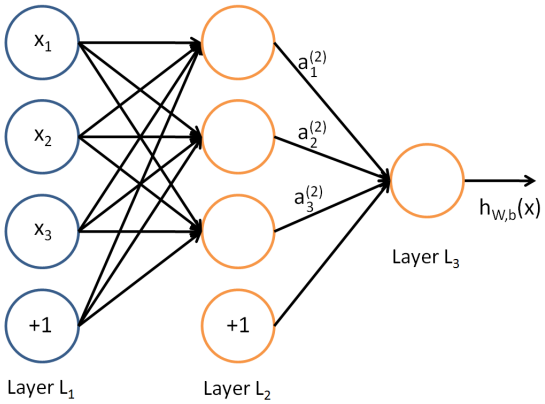


Figure 2.1: Structure of a MLP. Adapted from Ng et al. (NA)

The three layers of the MLP (input layer L_1, hidden layer L_2, output layer L_3) are organised in a way that the outputs of every unit (or neuron, represented as a circle) in one layer become the inputs for the units of the next layer. Each unit of the hidden and output layers has an associated weight vector w_i and a bias term b_i, and computes the function $w_i^T a_i + b_i$, with a_i being the input vector of this unit. For units in the hidden layer, an activation function $h_{w,b}$, is applied in addition. In summary, a MLP with one single hidden layer of k neurons and a single output neuron computes for a given input $x = [x_1, \ldots, x_p]$:

$$\hat{f}(x) = \left(\sum_{j=1}^{k} h(w_j^T x + b_j) \cdot w_{h+1,j} \right) + b_{h+t}$$

The weight terms and bias terms can be combined to a single weight vector which is then used to reduce the prediction error (Hutter et al., 2014). The R package `nnet` (W. N. Venables and B. D. Ripley, 2002), which is used in this thesis as an application of neural networks, uses feed-forward propagation for this purpose.

2.2 Regression methods for modelling runtime

Four regression methods were used to model runtime: model-based boosting with component-wise linear models, model-based boosting with smooth components, a generalised linear model and random forest. Random forest on regression problems do not differ greatly from random forest on classification problems, except for the final prediction, which is obtained by averaging (Hastie et al., 2016). Therefore, the description of the random forests in chapter 2.1.3 also applies for random forests as regression methods. A description of the generalised linear model, one of the most common models, shall not be within the scope of this thesis. The interested reader is referred to Fahrmeir et al. (2009).

2.2.1 Model-based boosting

Model-based boosting, as implemented in the R package `mboost` (Hothorn et al., 2016) optimises general risk functions. Boosting applies component-wise (penalised or unpenalised) least square estimates or regression trees as base-learners for fitting potentially high-dimensional data. With boosting one can fit generalised linear, additive and interaction models (Hothorn et al., 2010).

Data and model

Consider observations $(y_i, x_i), i = 1, \ldots, n$ with y_i being the response and x_i^T being the covariate vector, which can contain covariates of different types (continuous, binary, categorical). In component-wise boosting for regression models a structured regression model is considered. In this model, the conditional expectation is determined by $E(y|x) = h(\eta(x))$ with h being the (known) response function and $\eta(x)$ being the structured additive predictor of the form

$$\eta(x) = \beta_0 + \sum_{j=1}^{p} f_j(x)\,, \tag{1}$$

where $f_j(x)$ are generic representations of modelling alternatives, which can be linear, categorical, or smooth effects. Other alternatives include spatial, random effects or tree stumps (Hofner et al., 2011). These modelling alternatives are specified by base-learners (Hofner et al., 2014), which are described in more detail in the subsection *Base-learners* of this chapter.

One regression model can combine all different kinds of effects. Moreover, competing effects can be included into the model for one covariate. For example, a linear and a smooth effect can be modelled for one covariate. Finally, the model is the sum of all these effects.

Principle

The component-wise boosting algorithm seeks the solution of the following optimisation problem (Hofner et al., 2014):

$$\eta* := \underset{f}{\operatorname{argmin}}\, E_{Y,X}\Big[\rho\big(y,\, \eta(x^T)\big)\Big], \tag{2}$$

with $\rho(y,\, \eta(x^T))$ being a suitable loss function, such as the negative log-likelihood function or the L2-loss of the outcome distribution (Kneib et al., 2009). Hence the optimisation problem is a minimisation problem, where the expected loss with respect to the structured predictor η is minimised (Hofner et al., 2011). However, in practice the expected loss is unknown, since one deals with realisations of random variables and not the random variables themselves (Hofner et al., 2014). Therefore, the expected loss is replaced by the empirical risk function (Hofner et al., 2011):

$$R := n^{-1} \sum_{i=1}^{n} \rho(y_i, \eta_i)) \tag{3}$$

Boosting minimises this empirical risk R via functional gradient descent in a stage-wise fashion.

Algorithm

To minimise the empirical risk R over η, the following algorithm is applied: First, the base-learners need to be specified. It is possible to specify more than one base-learner for each covariate to model competing effects of the covariate. Now let J be the number of specified base-learners and m the number of iterations.

1. $m = 0$: Function estimates $f^{[0]}$ are initialized with some offset values. $f^{[0]}$ is a vector of length n (Hofner et al., 2014).
2. $m + 1$:
   (a) Compute negative gradient of the loss function and evaluate it at the estimate of the previous iterated $\hat{\eta}^{[m-1]}(x_i^T)$, $i = 1, \ldots, n$ (Hofner et al., 2011):
   $$u_i^{[m]} = -\frac{\partial \rho(y_i, \eta)}{\partial \eta}\bigg|_{\eta=\hat{\eta}^{[m-1]}(x_i)}, i = 1, \ldots, n \tag{4}$$
   The result is a negative gradient vector with n components $u^{[m]} = (u_1^{[m]}, \ldots, u_n^{[m]})'$, also called the working response.
   (b) The real-valued base-learners g_j are fitted to this working response. In other words, the negative gradient vector is estimated separately by the base-learners:
   $$u^{[m]} = \hat{g}_j^{[m]}(x_j) + \epsilon_j, \quad j = 1, \ldots, J \tag{5}$$

The result is a least squares or a penalised squares estimation. Having defined J base-learners, one receives J vectors of length n of predicted values $u^{[m]}$ (Hofner et al., 2014).

(c) The base-learner that fits $u^{[m]}$ best according to RSS (residual sum of squares) is selected (Hofner et al., 2011):

$$j* = \underset{1 \leq j \leq J}{\operatorname{argmin}} \sum_{i=1}^{n} \left(u_i^{[m]} - g_j(x_i)\right)^2 \tag{6}$$

(d) The current additive predictor $\hat{\eta}^{[m]}(\cdot) = \hat{\eta}^{[m-1]}(\cdot) + \nu \cdot \hat{g}_{j*}^{[m]}(\cdot)$ and the function estimate $\hat{f}_{j*}^{[m]}(\cdot) = \hat{f}_{j*}^{[m-1]}(\cdot) + \nu \cdot \hat{g}_{j*}(\cdot)$ are updated, while all other function estimates $f_j, j \neq j*$ remain unchanged. $0 \leq \nu \leq 1$ is a real-valued step length factor. This step length factor yields that only a fraction of the fitted values is added (Hofner et al., 2014).

3. Iterate steps 3 + 4 until the stopping criterion m_{stop} is reached (Hofner et al., 2014). For more information about m_{stop}, which is the number of boosting iterations, refer to subsection *Early stopping strategy* of this chapter.

Variable and model selection are a crucial elements of component-wise boosting and take place in step 3c: In this step only the base-learner that best improves the fit of the negative gradient of the loss-function is selected. If one base-learner is specified for each covariate, only one covariate of relevance in terms of this improvement is selected in this step (hence the name "component-wise boosting"). Furthermore, if one specifies competing base-learners differing in their type of modelling effects, this step also leads to model selection (Kneib et al., 2009). If a base-learner was selected multiple times, then the final function estimate $f_j, j \in 1, \ldots J$ is the sum of each individual estimate $\nu \cdot u^{[m-1]}$, which is obtained from those iterations, where the corresponding base-learner was selected (Hofner et al., 2014). Variable and model selection is also supported by the early stopping strategy, which is described in the respective subsection of this chapter. Step 3d, where the estimate of the true negative gradient of the empirical risk $u^{[m]}$ is added to the current estimate, leads to the fact, that the boosting-algorithm decends along the gradient of the empirical risk R. Thus, the empirical risk is minimised in a stage-wise fashion (Hofner et al., 2014).

Finally, we obtain an additive predictive function, the final boosting estimate:

$$\hat{f} = \hat{f}_1 + \ldots, \hat{f}_p \tag{7}$$

Base-learners

The base-learners specify the modelling alternatives for the functions $f_j(\cdot)$ (1) of the statistical model (Hofner et al., 2014). They are penalised or unpenalised least square models, and their input is either a single covariate or a small subset of

covariates. Aiming for variable selection, one base-learner should be specified for each covariate. The estimates can be expressed as follows:

$$g_j(\mathbf{x}) = \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{K})^{-1}\mathbf{X}^T u = Su \tag{8}$$

with S being the hat-matrix, $\mathbf{X}$ being the design matrix and $\mathbf{K}$ the penalty matrix. λ is the corresponding smoothing parameter, controlling the amount of penalisation. $\lambda = 0$ yields unpenalised least squares estimates, while $\lambda > 0$ yields penalised least squares estimates. Modelling alternatives include, besides linear, categorical and smooth effects, also spatial, random and many more effects. Subsequently, we focus on the linear, categorical and smooth effects, since these effects are used for modelling runtime. The linear base-learners cover – among others – continuous and categorical effects. The smooth effects are expressed with P-Splines, which are B-Splines with penalty differences (Kneib et al., 2009). For more details on P-Splines and the estimation by the penalised least squares criterion refer to Eilers and Marx (1996). In the `mboost` (Hothorn et al., 2016) package, the default setting for the penalised least squares estimates is ridge penalty for continuous and unordered categorical covariates, first order difference penalty for ordered categorical covariates and second order difference penalty for P-spline base-learners.

Unbiased variable selection

For each variable and effect type, separate base-learners need to be specified. The boosting procedure determines, whether these base-learners enter the model, and whether the specified effect is of relevance. But variable and model selection can be "seriously biased if the competing base-learners have different degrees of freedom" (Hofner et al., 2011).

Variable selection bias may occur if there are competing categorical covariates with different numbers of categories. The categorical covariate with the most categories is more flexible and thus is preferred by the boosting process, when using unpenalised least squares base-learners.

Model selection bias means that the smooth effects are favoured compared to linear effects. To identify whether a covariate has a linear or smooth effect on the response, a linear base-learner and a smooth base-learner needs to be specified. Then the boosting process decides which effect is relevant. But since the smooth effect offers more flexibility than the linear effect, and, moreover, incorporates the linear effect, the smooth effect is always preferred by the algorithm. In order to solve this bias, the competing effects need to be made comparable with respect to their flexibility, which is measured by the degrees of freedom (df).

To prevent selection and model bias equal degrees of freedom should be specified for all base-learners. Hence, degrees of freedom need to be set to one single free parameter ($df = 1$). For categorical effects, this can be achieved by setting $df = 1$ for all base-learners, which leads to unpenalised least squares models. But for

smooth effects *df* cannot be made arbitrarily small. For λ approaching infinity, a polynomial of order $d-1$ remains unpenalised when using a d-th order penalty. Applying, for instance, second order differences, which is common in practice, a linear effect (a polynomial of first order) remains unpenalised and therefore, $df \geq 2$ for all λ (Hofner et al., 2011). However, it is possible to specify base-learners with $df = 1$ by applying a reparametrisation of the model (Kneib et al., 2009). This is also called P-spline decomposition. Thereby, the base-learner is decomposed into a parametric part and a non-parametric part:

$$g(x) = g_{param}(x) + g_{smooth,\,centered}(x) \tag{9}$$

The paramtric part $g_{param}(x)$ captures the $(d-1)$th order polynomial that remains unpenalised for λ approaching infinity. Kneib et al. (2009) recommends defining separate base-learners for each of the lower-order polynomials: $g_{param}(x) = \beta_1 x + \ldots + \beta_{d-1}x^{d-1}$. The non-parametric part $g_{smooth,\,centred}(x)$ is the smooth deviation from the polynomial. Thus the base-learner is a P-spline with centred effect and $df = 1$. For second order differences one obtains (Hofner et al., 2014):

$$g(x) = \beta_0 + \beta_1 x + g_{smooth,\,centered}(x) \tag{10}$$

Whenever the degrees of freedom of base-learners are made comparable by setting $df = 1$ for all base-learners and by applying the P-spline decomposition, a separate base-learner for the intercept should be specified. Therefore, the intercept needs to be excluded from the linear effects and the covariates need to be centred. Otherwise, linear effects of base-learners would be forced through the origin with no data lying there (Hofner et al., 2011).

In conclusion, penalisation is applied to reduce the selection bias for non-informative covariates. Another useful effect of penalisation is the shrinkage of effects of the influential covariates (Hofner et al., 2011).

Early stopping strategy

To prevent overfitting, the boosting algorithm should not be run until convergence. Therefore, a stopping parameter m_{stop}, which is a tuning parameter, is used to specify the optimal number of boosting iterations (Hofner et al., 2014). The optimal value of this stopping parameter is chosen by resampling approaches or AIC-based techniques. The resampling approaches assess the appropriate number of boosting iterations by comparing cross-validated out-of-bag (OOB) empirical risk estimates for different stopping parameters (Hothorn et al., 2016). In practice m_{stop} can be obtained by bootstrap samples, k-fold cross-validation or subsampling, as is implemented in the `mboost` package. The early stopping strategy together with a small value of step-length ν leads to a small increase of the effect estimates. Hence, with an optimal number of boosting iterations, the effect is shrunken such that the predictive power of the model is maximal. Although this shrinkage technique

leads to a slightly increased bias, it leads to better prediction accuracy because of the reduced variance. Moreover, the early stopping strategy stabilises the effect estimates and avoids multicollinearity (Hofner et al., 2014). As mentioned earlier, it also supports the process of variable and model selection, since it leads to sparse models (Hothorn et al., 2010).

2.3 Meta-learning

As proposed in publications by Hutter et al. (2014), Reif et al. (2011), Priya et al. (2011), Priya et al. (2012) and Doan and Kalita (2016) the approach used for modelling runtime applies the concept of meta-learning. Reif et al. (2011) defines meta-learning as using "knowledge about already solved problem instances to gain information about unknown problems regarding one ore multiple learning schemes". Therefore, meta-learning is often applied for predicting the suitability or performance of classifiers for a given problem. Particularly, regression methods (meta-regressors) for meta-learning predict quantitative performance values of a classifier, such as accuracy or runtime, based on a given set of predictors. These predictors are called meta-features, and they are typically properties of the dataset, such as the number of instances and number of features. Other measures include statistical, model-based, or landmarking features (Reif et al., 2011). Additionally, the hyperparamters of an algorithm can also serve as meta-features.

In practice, regression methods predict, which classifier is best or fastest for an unknown dataset, given the meta-features. The meta-learning approach usually involves two steps (Priya et al., 2012): a) the generation of the meta-dataset by extracting meta-features, such as the dataset characteristics and the algorithm's hyperparameters; b) the induction of a meta-learning model by applying a regression model to the meta-dataset.

Usually, performance of the meta-regressors is evaluated, by using Leave-one-out cross-validation (LOOCV), 10-fold cross-validation or similar strategies (Priya et al., 2011). The accuracy of the predictions is assessed with measures like the mean absolute deviation (MAD) (Priya et al., 2011), the root mean squared error (RMSE), the mean absolute error (MAE) (Doan and Kalita, 2016), the relative absolute error (RAE) (Reif et al., 2011), and many more.

3 Methods

3.1 Data and data generating process

The main objective of this thesis is to model and predict the runtime of six different classification algorithms (gbm, glmnet, naiveBayes, nnet, ranger and rpart) depending on their hyperparameter settings and the dataset characteristics. The approach proposed in this thesis uses the concepts of meta-learning. The individual steps are visualised in figure 3.1 and can be divided into two main parts:

1. The generation of the meta-data includes gathering meta-features, like the dataset properties and the algorithm's hyperparameter. Subsequently, the target classifiers are trained on 65 different datasets with several randomly drawn hyperparamter sets. This step is implemented by using the platform OpenML, R packages `mlr` and `batchtools` and the LRZ cluster. The information about the runtime is stored together with the characteristics of the dataset and the hyperparameters to create the meta-data.
2. On the basis of the meta-data, one or more meta-regressors are induced to model the relationship between the meta-features (predictors) and the runtime (response). A separate regression model needs to be trained for each target classifier.

3.1.1 Collecting datasets from OpenML

All datasets were collected from OpenML, which is a platform to share results from machine learning experiments in a standardised way with other users (Vanschoren et al., 2014). The R package `OpenML` (Casalicchio et al., NA) integrates OpenML into R and is available as development version from GitHub.
Currently, there are 19,608 datasets (status on 8 Aug. 2016, Casalicchio et al. (NA)) available on the platform. In the scope of this thesis, the extent of datasets was reduced to a reasonable amount. For this purpose, the selection process for the datasets was first geared to a benchmark study from OpenML (available on OpenML with tag: study_14), which analysed 127 datasets. Of these datasets those with less than 500 features and fewer than 20 classes in the target were selected.

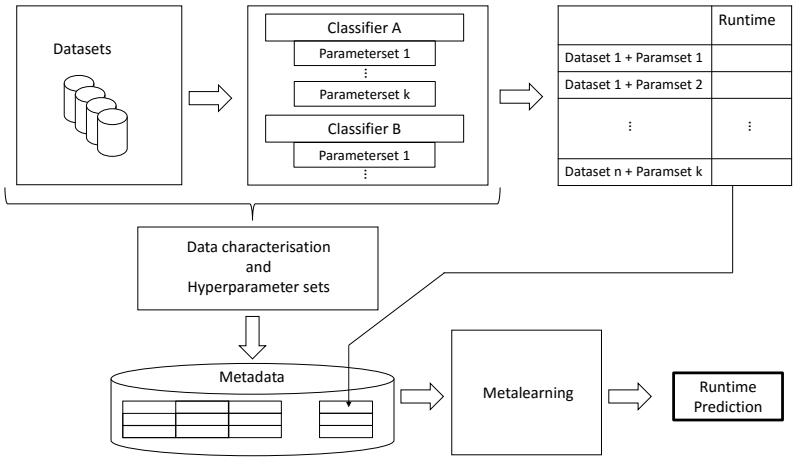


Figure 3.1: Proposed meta-learning approach for modelling runtime of classification algorithms. Diagram was modified and adapted from Priya et al. (2011)

Moreover, only datasets, where the target is a factor variable were chosen in order to ensure that all the classifiers are able to be run on the selected datasets. Two datasets were excluded since the farff.reader[1] did not support those yet and three were excluded because of a missing target description in the dataset description. This selection process led to a total of 65 datasets, which are listed in table B.1. A summary of the properties of these datasets are presented in table 3.1.

Table 3.1: Summary of the dataset characteristics. It shows the minimum (Min.), maximum (Max.), median (Med.), mean, first and third quartile (1st Qu., 3rd Qu.) of the measures that are used for characterising the datasets.

	Min.	1st Qu.	Med.	Mean	3rd Qu.	Max.
NumberOfInstances	500.00	930.00	2000.00	5822.00	5744.00	45310.00
NumberOfFeatures	5.00	11.50	23.00	48.59	49.75	300.00
NumberOfClasses	2.00	2.00	3.00	4.60	6.25	15.00
MajorityClassSize	80.00	285.50	720.50	3385.00	2292.00	33220.00

3.1.2 Construction and implementation of the experiments

The experiments to measure runtime were created with the package `mlr` (Bischl et al., NA), which provides a framework for modelling, hyperparameter optimisation, feature selection, pre- and post processing of data, resampling strategies, benchmark studies and parallelisation.

`mlr` was used to create learners with varying sets of hyperparameters. A learner is basically an interface for machine learning algorithms in R. In the constructor

[1] `farff` (Bischl and Bossek, NA) is the R package with which the arff files are passed.

of the learner, it is possible to specify the hyperparameter set of the algorithm. For the present experiments, these hyperparameter sets were generated randomly for each of the six classification algorithms, respectively learners. Random selection was applied not to all hyperparameters, but to those which are of main importance for the algorithm's performance and to those that were considered to have an impact on runtime. A detailed description of these hyperparameters and the hyperparameter search space is presented in table B.2. The reason for a random selection of hyperparameters is, that it is usually not feasable to consider all combinations of the hyperparameters in a hyperparameter set, especially, if the hyperparameter is continuous. Moreover, with every extra dimension, i.e. additional hyperparameter, the volume increases exponentially. The solution on how to search such metric spaces is an own research area and is widely discussed in Chávez et al. (2001). Within the limits of this thesis, we decided to draw $10 * dimension$ samples from the hyperparameter search space for each classifier on each dataset.

With `mlr`'s `resampling` function, the dataset was split into a training and a test set. The split rate was also chosen randomly within the hyperparameter set. Thus, the split rate varies for each experiment. After splitting the dataset, the learner was trained on the training set and predictions were made on the test set. Additionally, performance measures, like time needed for training (training time), time needed for making predictions (prediction time) and the mean misclassification error (mmce) were assessed. Training and prediction time are measured within the function calls for modelling and predicting in `mlr`. These functions call the R function `system.time()` and pass the real times (in seconds), that it took to evaluate the committed expression, to `mlr`'s time measures.

In order to assess variability of time measurement on a single experiment[2], each experiment was repeated 10 times.

In summary, each classifier needed to be trained on 65 datasets, and on each dataset the classifier needed to be run with $10 * dimension$ different hyperparameter sets. Additionally, each experiment should be repeated ten times. Altogether, this led to 159,900 models that needed to be fitted in order to measure runtime. Therefore, the experiments were run on the LRZ cluster, i.e. a batch computing environment. The R package `batchtools` (Lang and Surmann, NA) provides a comprehensive framework for working on a cluster.

In `batchtools`, experiments are created by defining the problems, algorithms, as well as, problem and algorithm designs. The algorithm design refers to the hyperparameter set of the learner. Subsequently, every design is combined with every problem and every algorithm. In this way each classifier with its various sets of hyperparameters is run on all defined problems. One such combination is called experiment. Repeating an experiments creates several "jobs". Jobs are not submitted when they are created, instead the jobs and all information regarding the

[2] An experiment tags the entity of the problem, algorithm and their respective parameters (Bischl et al., 2015).

problems, algorithms and designs are filed and recorded in a registry. Submission of jobs must be called explicitly.

Besides the API (application programming interface) for creating and submitting jobs, `batchtools` offers an infrastructure to interact with a cluster, especially with its scheduler (Bischl et al., 2015). In this context, resource requirements for each job need to be specified. In order to obtain a valid and interpretable analysis all jobs were run with the same requirements. To get results for every single job, this implies setting the runtime limit according to the job, which takes the longest time, and the memory limit according to the one, which requires the most memory. However, this was not possible because of three reasons: Firstly, the user needs to specify an appropriate segment of the cluster, each having their own resource limits. Secondly, the maximum run time can be estimated by testing some jobs on the greatest dataset locally. But since the runtime of a job, hence, the runtime of an algorithm, depends not only on the number of instances of a dataset, this estimation might be biased. Finally, setting a high general upper limit for runtime and memory leads to long queue times for each job. Therefore, a trade-off between an upper limit being as large as possible and a queue time being appropriate for the schedule of this thesis needed to be determined. This led to following resource requirements: a runtime limit of 1000 seconds and a memory limit of 2.2 GByte. These limits were selected in dependence on the limits of the partition of the LRZ Linux cluster, which was used for this analysis: serial processing was implemented on the CoolMUC2 cluster, on partition serial_mpp2 with 28-way Haswell-based nodes.

3.2 Statistical analysis

Explorative analysis Explorative analysis in terms of summary statistics were implemented on training time, prediction time, the total of training and prediction time and the mean misclassification error.

The variability of runtime was analysed based on the 10 replications per experiment. High variability in runtime measurements within an experiment (entity of one classifier with one distinct set of hyperparameters and one specific dataset), and a small average runtime, should lead to cautious interpretation of the models for predicting runtime. Variability of the runtime measurements within the experiments were assessed with the relative standard deviation (RSD) also known as the coefficient of variation (cv):

$$cv = \frac{s}{\bar{x}}$$

Thus, the empirical standard deviation s is divided by the mean $\bar{x}$. This measure is scale-independent (Kohn, 2005) and is suitable to compare variability between the classifiers.

In addition to variability of runtime measurements, variability of prediction accuracy, which was measured by the mean misclassification error, was analysed.

Runtime modelling For runtime prediction meta-data was aggregated. Thereby, the ten replications of one experiment were summarised by taking the average of these replications. In table B.3 in the appendix the number of observations for the meta-datasets of each classifier are presented before and after aggregating.

For each classification algorithm, runtime was modelled separately with four different regression models: gradient boosting with component-wise linear models (glmboost), gradient boosting with smooth components (gamboost), a generalized linear model (glm) and a random forest. The predictors of those models included:

- dataset characteristics
  - NumberOfInstances: number of instances in the dataset
  - NumberOfFeatures: number of features in the dataset
  - NumberOfClasses: number of classes of the target of the dataset
  - MajorityClassSize: size of the majority class of the dataset
- hyperparameters (For a more detailed description please refer to table B.2 in the appendix.)
  - naiveBayes: no hyperparameters
  - rpart: minsplit, minbucket, cp, maxdepth
  - ranger: num.trees, replace, sample.fraction, mtry, respect.unordered.factors, sub.sample.frac
  - glmnet: alpha
  - gbm: n.trees, interaction.depth, shrinkage, bag.fraction
  - nnet: size, maxit, skip, decay

Responses that were modelled were training time and prediction time.

The accuracy of fit of the regression models were compared to each other with the root mean squared error (RMSE) (Hyndman and Athanasopoulos, 2014)

$$RMSE = \sqrt{(mean(y_i - \hat{y}_i)^2},$$

which is a scale-dependent error measure. y_i is the observed runtime, while $\hat{y}_i$ is the predicted runtime. Lower values of RMSE indicate a better model fit. Moreover, the relative absolute error (RAE) (Reif et al., 2011)

$$RAE = \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{\sum_{i=1}^{N} |\bar{y} - y_i|}$$

was calculated for each regression method on each classifier. The absolute error of the prediction method is divided by the absolute error of the prediction of a baseline method, which is – in the present case – the average runtime of the classifier. A RAE value greater than one indicates that predicting the average is more precise than using the regression method.

In table B.6 summary statistics for runtimes are presented. It shows that the distribution of the training time and the prediction time for all classifiers is right-skewed and long-tailed. Therefore, the `Gamma-Reg()` family (Hothorn et al., 2016) for the gradient boosting approaches was chosen. This family is suitable for non-negative response variables and the implemented loss-function is the negative gamma log-likelihood with logarithmic link function (Buehlmann and Hothorn, 2007). Correspondingly, the `Gamma(link = "log")` family was chosen for the generalized linear model (R Core Team, 2016). Since random forest is a non-parametric approach, no distribution needed to be specified.

Furthermore, for the gradient boosting models a step-size (ν) of 0.3 was used to reduce computing time. For the same reason, the offset value was set to zero. The optimal number of boosting iterations (optimal m_{stop}) was calculated by 10-fold cross-validation and searched for on a grid from one to 100,000. Variable and model selection were implemented by having several base-learners for each covariate, which model the competing effects of a covariate.

To reduce model selection bias in gradient boosting with smooth components (`gamboost`) the P-spline decomposition approach was used. This implied decomposing the smooth effect into an unpenalised polynomial and a smooth deviation from this polynomial (Hofner et al., 2014). The latter being the non-parametric smooth part of the decomposition, which is defined via the `bols()` function. More specifically, these are P-splines with one degree of freedom and 20 knots. The unpenalised polynomial corresponds to the parametric part and is specified via the `bbs()` function. The type of effects specified in the *bols*() function included linear and categorical effects. In the context of the P-spline decomposition, these base-learners needed to be specified without intercept and a separate intercept was defined. Thus, the boosting algorithm could explicitly choose and update the intercept (Hofner et al., 2014). In case the model comprised categorical covariates, penalised least squares for all parametric base-learners (`bols()`-learners) with one degree of freedom ($df = 1$) need to be added to avoid variable selection bias.

In both boosting modelling approaches, the continuous covariates were centred, in order to fasten risk minimisation and to avoid that the algorithm does not converge to the "correct" solution (Hofner et al., 2011).

Variable selection in a generalized linear model glm was achieved with the `step` function, which chooses the best model by AIC with a step-wise algorithm. Step-wise search was performed in a backward-forward manner.

Results of glmboost and gamboost model are presented using partial effect plots. The partial effects are obtained by summing the functional estimates of the mod-

elling alternatives for each covariate that were included in the final model. The modelling alternatives for glmboost included a linear, a quadratic, a square root and a logarithmic effect. With respect to gamboost a linear and a smooth effect for each covariate was fitted. The result is a dot chart, whose dots are linked to each other by a line in order to visualise the shape of the resulting partial effect.

Furthermore, the estimated coefficients of glmboost and glm models on training time and prediction time are tabulated together for each classifier.

Each random forest was fitted with the default settings of `randomForest`. Variable importance was measured and is visualised with the help of variable importance plots. Thereby, two variable importance measures are presented: the mean decrease in accuracy and the mean decrease in node impurity. The latter is only visualised, but will not be interpreted. This is, firstly, because this measure prefers variables with more categories, which is undesirable in cases, where the models has categorical variables. Secondly, in case of correlated features, this measure decides on one of the features as being important, while the others become unimportant, but with no concrete preference of one over the others. Additionally, partial dependence plots are used to illustrate the relationship between an individual covariate and the predicted runtime obtained from the random forest model.

The data generating process and all statistical analyses were performed using the statistical software R, version 3.3.0, and the R packages `batchtools`, version 0.1, `mlr`, version 2.9, `OpenML`, version 1.0, `mboost`, version 2.6-0, `randomForest`, version 4.6-12, `plyr`, version 1.8.4. Both `batchtools` and `OpenML` are not released on CRAN yet, the current development versions are available on GitHub. All charts are produced using `ggplot2`, version 2.1.0., `reshape`, version 0.8.5, `gridExtra`, version 2.2.1., and `grid`, version 3.3.0, or the `plot()`-function of the respective model.

4 Results

4.1 Explorative Analysis

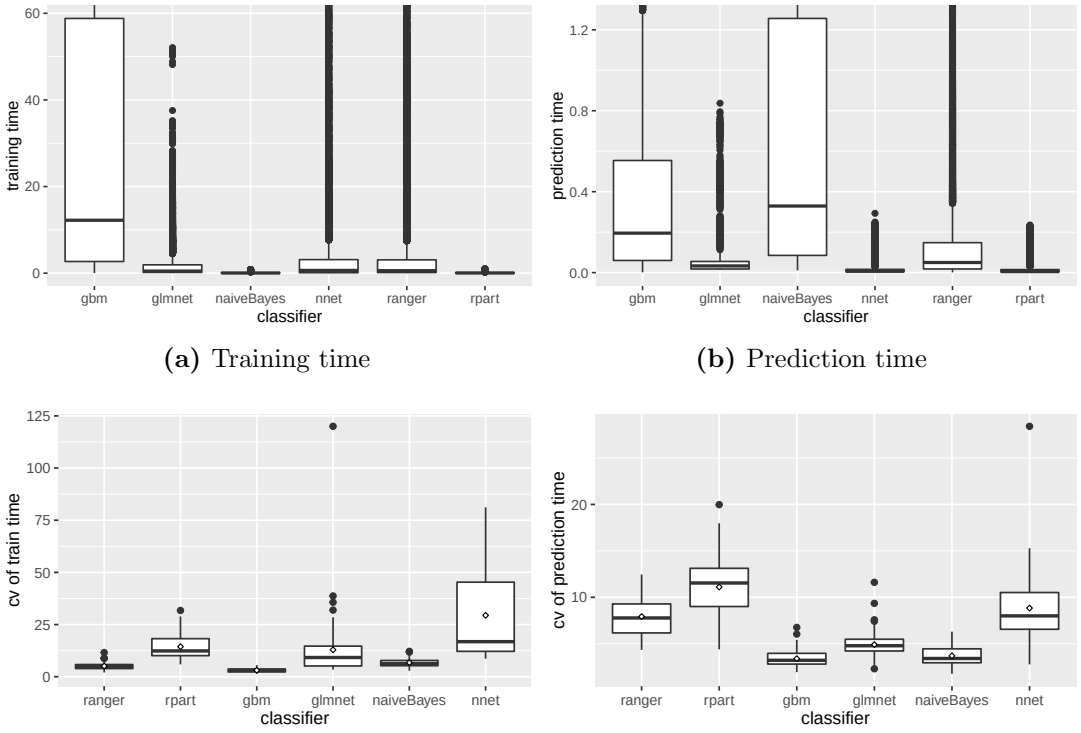
In this analysis, the runtime of five classification algorithms, each trained on 65 datasets, was evaluated. Overall 159,900 jobs were submitted, of which 1,133 were erroneous (= 0.71%) and 2,156 expired (= 1.35%) because of exceeding wall time or memory limit. These capped runtime observations were not considered in the analysis. For more details on the erroneous and expired jobs, see table B.4 and B.5.

The overall mean for training time is 13.364 seconds and for prediction time 0.317 seconds (table 4.1). On average, `naiveBayes` is the fastest algorithm in training

Table 4.1: Mean of training time, prediction time and total runtime of the five classifiers

Classifier	Training	Prediction	Total
gbm	63.832	0.472	64.304
glmnet	2.173	0.062	2.235
naiveBayes	0.081	1.125	1.205
nnet	7.245	0.019	7.264
ranger	6.759	0.204	6.964
rpart	0.091	0.019	0.11
Total	13.364	0.317	13.680

(0.081 sec) but the slowest in prediction (1.125 seconds). `rpart` is the second fastest in training, followed by `glmnet` (2.173 seconds) and `ranger` (6.759 seconds). `gbm` has a comparably high training time of 63.832 seconds. Prediction time is on average – except for naiveBayes – below one second. If training and prediction time are considered together (total), then `rpart` is fastest on average, followed by `naiveBayes`, `glmnet`, `ranger`, `nnet` and `gbm`. The boxplots in figure 4.1a and 4.1b demonstrate the distribution of training time and prediction time, respectively, for the 6 classifiers on the 65 datasets. All classifiers present numerous outliers, of which a great part are not shown in the plot, because they exceed 60 seconds for training time and 1.2 seconds for prediction time. The most de-



(a) Training time

(b) Prediction time

(c) CV on training time

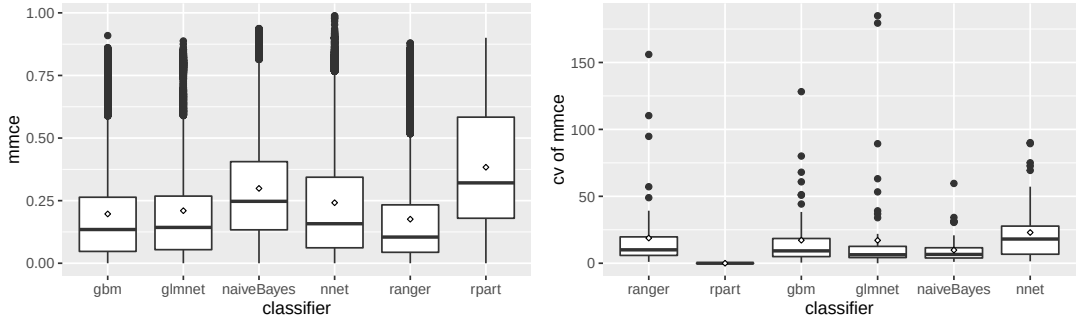
(d) CV on prediction time

Figure 4.1: Boxplots of the distribution of training and prediction time (a and b), and boxplots of the distribution of the respective coefficient of variation (c and d) for each classifiers across the 65 datasets.

viant training time of 1033.790 seconds was observed for a `gbm` (`n.trees` = 5000, `interaction.depth` = 5, `shrinkage` = 0.007, `bag.fraction` = 0.831) on the "mfeat-factors" dataset. With respect to prediction time the most deviant time was 6.110 seconds for `naiveBayes` on the dataset "sylva_agnostic". From the boxplots and the summary of statistical values of run times in B.6 we can infer that the distribution for runtimes is right skewed. Therefore, all parametric predictions models were run with assuming a gamma distribution.

The variability of runtime measurements was assessed by repeating an experiment with the same set of hyperparameters on the same dataset ten times. It is measured with the coefficient of variation (cv) to ensure comparability. Variability of training time is higher than variability of prediction time. (Note the different y-scales in figure 4.1(c) and 4.1(d).) On average, variability of training time is highest for `nnet` and varies the most across the different datasets. The highest variability observed was for `glmnet`, `rpart` and `nnet` (248, 264.9 and 277.5 respectively). These values are not presented in the boxplots of 4.1(c), because values above 125 are not shown in the plot. The highest average variability in prediction time was found for `rpart` (11.1) and `ranger` (7.953). The maximum value for prediction time is 123.4 for `ranger`, which is not displayed in the boxplot of 4.1(d).

To measure prediction accuracy the mean misclassification error (mmce) was assessed on all jobs, but was missing on 140 jobs running classifier `gbm`. The mean of the mmce of classifiers `gbm`, `glmnet`, `nnet` and `rpart` is below the overall average (0.246), while the mmce of `naiveBayes` and `rpart` are above (0.3, and 0.38 respectively). Figure 4.2 (a) and table B.6 demonstrate that the distribution of the mmce is right-skewed for all learners except for `rpart`. This classifier has almost equally



(a) Summary of mmce **(b)** Summary of cv of mmce

Figure 4.2: Boxplots of (a) the distribution of the mean misclassification error (mmce) and (b) of its coefficient of variance (cv) for each classifier across the 65 datasets.

small, medium and high mmce rates. On 70 jobs the mmce exceeded a value of 0.9. `gbm` and `naiveBayes` both showed these high mmce on dataset "sylva_agnostic" and `rpart` on "semeion", while `nnet` showed them on seven different datasets.

Also the variability of the mmce was measured by the coefficient of variation. It is on average lowest for `rpart`, followed by `ranger` and `naiveBayes` (figure 4.2(b)). The interquartile range of `rpart`'s cv is small (0 - 5.465), while it is greatest for `nnet` (0 - 26.810). Maxima of `gbm`, `glmnet`, `nnet` and `rpart` are not presented in the graph and are approximately 300 (see B.7 for a detailed summary).

4.2 Model results

The main intent of this thesis is to predict runtime with respect to the dataset characteristics and the hyperparameters of the algorithm. Four different model approaches were used for prediction of runtime: gradient boosting with component-wise linear models, gradient boosting with smooth components, generalized linear model and random forest. Training time and prediction time were analysed separately. The prediction accuracy of the regression models on the training data for each classifier is shown in table 4.2 and 4.3. The first table presents the results for the relative absolute error (RAE), the second table for the root mean squared error (RMSE). The best values for each classifiers are highlighted. The RAE of all four regression models on all classifiers is below one, indicating that the regression models are more accurate than a commonly used baseline, the average runtime.

Table 4.2: Relative absolute error of the four regression models for runtime predictions on training data for each classifier. Boldface indicates the lowest RAE in each column, thus, for each classifier.

	naiveBayes	rpart	ranger	glmnet	gbm	nnet
Training time						
glmboost	0.407	0.332	0.578	0.417	0.275	0.637
gamboost	0.289	**0.265**	–	**0.267**	0.177	0.414
glm	0.412	0.335	–	0.430	**0.108**	0.628
randomForest	**0.166**	0.376	**0.206**	0.270	0.128	**0.251**
Prediction time						
glmboost	0.369	0.750	0.199	0.769	0.220	0.693
gamboost	0.264	0.578	–	0.573	0.224	0.693
glm	0.222	0.760	0.196	0.773	0.137	0.693
randomForest	**0.220**	**0.236**	**0.083**	**0.372**	**0.120**	0.693

Table 4.3: Root mean squared error (RMSE) of the four regression models for runtime predictions on training data for each classifier. Boldface indicates the best prediction accuracy in each column, thus, for each classifier.

	naiveBayes	rpart	ranger	glmnet	gbm	nnet
Training time						
glmboost	3.483	0.038	25.041	2.423	71.174	40.277
gamboost	3.540	**0.031**	–	**1.704**	39.849	28.037
glm	3.483	0.039	–	2.405	**23.747**	39.988
randomForest	**0.025**	0.044	**10.121**	1.862	24.786	**16.914**
Prediction time						
glmboost	1.332	0.028	0.147	0.066	0.206	42.680
gamboost	1.000	0.022	–	0.046	0.220	42.677
glm	0.941	0.029	0.142	0.067	0.143	42.680
randomForest	**0.764**	**0.009**	**0.062**	**0.033**	**0.121**	**42.676**

A visual comparison of the model fits, thus, a visualisation of the results for the RMSE, is demonstrated in figure 4.3. On the left observed training times are plotted against predicted training times and on the right results for prediction time are shown. Each row corresponds to a classifier. With respect to RMSE random forests (green dots) present the best overall performance for modelling training time, as well as, for modelling prediction time. Gamboost (blue dots) performs second best for modelling training time and glm (black dots) for modelling prediction time. Considering only the parametric approaches, the gamboost model demonstrates the best model fit for training time. With respect to prediction time gamboost and glm have the lowest RMSE values among the parametric models. All regression models show a tendency to get increasingly inaccurate, the greater

the observed runtime. glmboost and glm on training time tend to overestimate prediction time for training times greater than 0.4 seconds for classifier `naiveBayes` and `rpart`. For classifier `gbm` the glmboost overestimates training time for all observed training times. With respect to classifiers `ranger`, `glmnet` and `nnet` the training time is underestimated for values greater than 200 seconds, 20 seconds and 275 seconds respectively. Notably, the meta-dataset for `ranger` has one observation, where observed training time was 50.552 seconds. The best prediction for this observation was made by random forest (predicted training time: 18.682 seconds). However, the difference between the predicted and the observed value is rather larger. Also particularly remarkable, are the model fits on `ranger`. glmboost (red dots) has a RMSE twice as high as random forest. With respect to prediction time, the tendency to over or underestimate is not as distinct as it is for training time. Model fits for classifier `rpart` and `glmnet` are not as accurate as the model fits for the other classifiers. Estimated prediction time for `glmnet` is underestimated starting from a value of 0.3 seconds. Prediction accuracy for `nnet` is good, yet, for prediction times greater than 0.15 seconds all regression models have the tendency to underestimate. However, there are some specific data points, where true prediction time is 0.0778 seconds and the corresponding prediction is five times larger than that value.

A summary on the variables that were excluded by either the boosting process or, in case of glm, the step-selection, is demonstrated in table 4.4 for the parametric models on training time and in table 4.5 for the parametric models on prediction time. Additionally, the optimal number of boosting iterations m_{stop} for the boosting algorithms are tabulated. For `rpart`, `glmnet` and `nnet` an appropriate value for m_{stop} could not be found with a 10fold cross-validation on a grid search from 1 to 100,000. A value near 100,000 indicates, that the search would have been continued, if the grid search was extended to even larger upper boundaries. Since the search for an appropriate m_{stop} on grids of this size takes several hours, no further search was implemented. However, this needs to be considered with respect to the presented results. Particularly, the variable and model selection of the models, where the appropriate m_{stop} was not found, led not to sparser models. Mostly, all variables were included in the final model, or only a few variables were excluded.

Furthermore, results of the glm fit should be interpreted cautiously, when either the glm algorithm itself, or the subsequent step-selection algorithm did not converge. This applies to the glm model on training time for classifier `glmnet`, to the glm model on prediction time for classifier `gbm`, and to the glm model on both runtimes for classifier `ranger`. Moreover, there are no model result for gamboost for classifier `ranger`. The model could neither be fitted on training time nor on prediction time, since Cholesky factorisation failed.

In the following sub-chapters, the results of the regression methods modelling the classifiers runtime are presented separately for each classifier. To stay within the scope of this thesis, results of the best performing regression method and

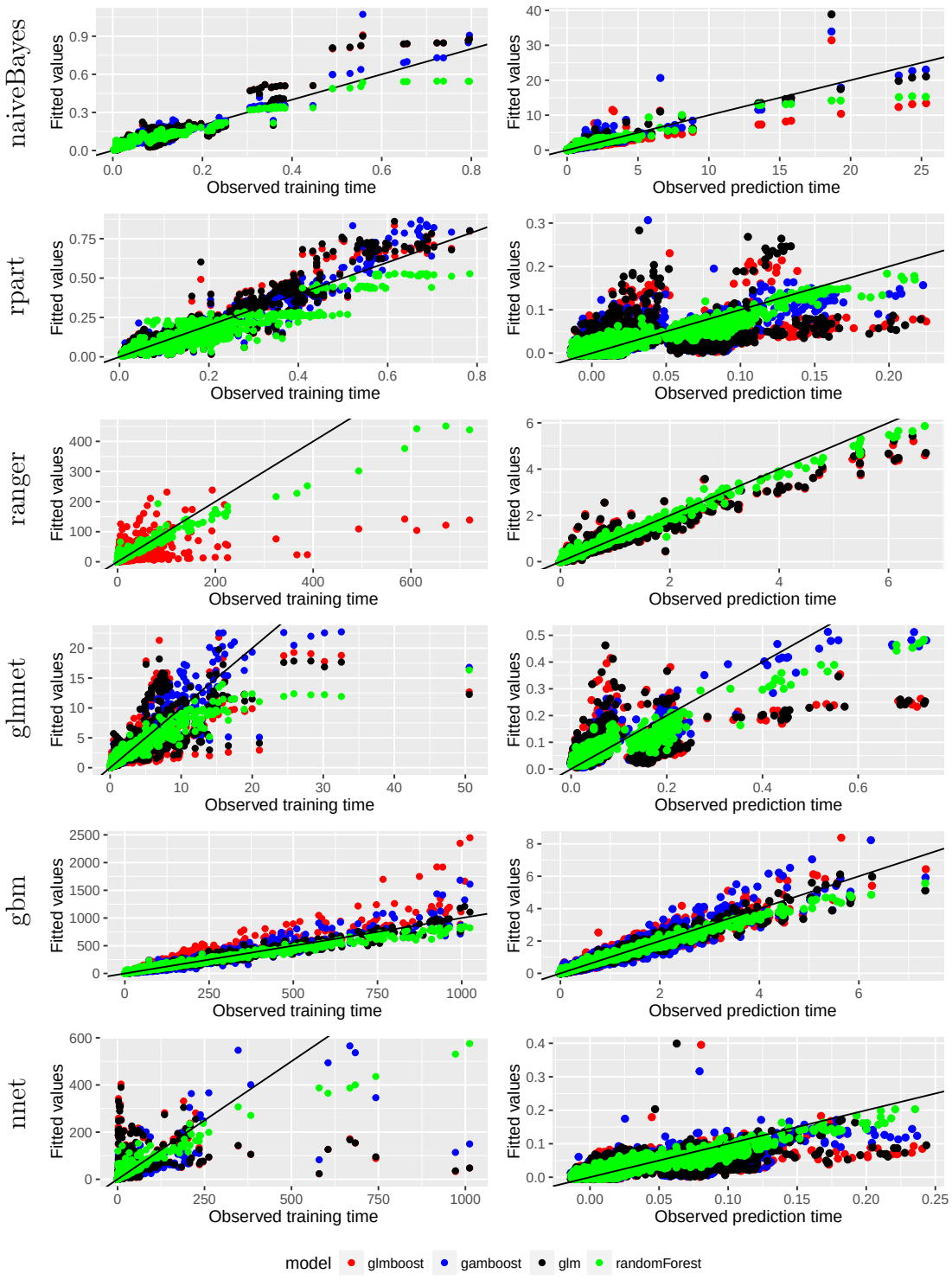


Figure 4.3: Quantitative comparison of the regression models' prediction accuracy on training data for each classifier. The straight line is the bisector of the coordinate axes. For a perfect fit all data points lie on that line.

results of glmboost, which is the method this thesis focused on, will be presented. Additionally, results derived from the random forest fit are pointed out for those classifiers, where random forest is not the best prediction method. In order to keep the thesis complete, the appendix includes: the estimated coefficients of the glmboost and the glm model in table B.8 to B.13. The results of the glmboost model are visualised with partial effect plots for each covariate. The plots for all six classifiers are shown from figure A.1 to A.12. The results of the gamboost models are demonstrated in figure A.13 to A.22. These plots display the single linear and the single smooth partial effect, as well as the combined partial effect. The variable importance plots of random forest are shown in figure A.23 to A.34. Additionally, partial dependency plots are demonstrated from figure A.35 to A.46.

Naive Bayes

Classifier `naiveBayes` has one hyperparameter, the `laplace` parameter. This was set to one, in order to avoid errors due to zero probability values. Hence, the covariates of the model are: NumberOfInstances, NumberOfFeatures, NumberOfClasses and MajorityClassSize.

The glmboost model on training time has a RMSE of 3.483 and fits the data second best, after random forest. The partial effect plots in figure A.1 demonstrate that training time increases for an increasing NumberOfInstances and for an increasing NumberOfFeatures. The increase in training time is, initially, very sharp for both covariates. For NumberOfInstances it then passes into a clear linear increase. An increase in predicted training time can also be noted for NumberOfClasses between 3 and 11 classes. For MajorityClassSize the increase in training time is also initially sharp, but drops after a size of approximately 15,000.

The best model on training time as well as on prediction time is random forest with respect to RMSE. The results of the variable importance measurements of random forest are displayed in A.23 and A.24. In both models NumberOfFeatures is the most important variable, followed by NumberOfInstances. The respective partial dependence plots (see A.35) reveal a quasi linear relationship between the estimated training time and NumberOfFeatures.

The results of glmboost on prediction time are presented in figure A.2. An increase in the NumberOfInstances and NumberOfFeatures, results, initially, in a sharp increase in prediction time, which soon flattens. The other two covariates NumberOfClasses and MajorityClassSize have no influence on prediction time.

Recursive Partitioning

The implementation of a decision tree in R with `rpart` has several hyperparameters that can be tuned. In this analysis the influence of the hyperparameters `maxdepth`,

Table 4.4: Results of variable selection: excluded variables from models on training time. Variable selection was obtained due to boosting (glmboost and gamboost) or step-wise selection regarding `glm` for each classifier. Additionally, the number of boosting iterations (m_{stop}) of the boosting algorithms is tabulated.

	glmboost		gamboost		glm
	m_{stop}	Excluded variables	m_{stop}	Excluded variables	Excluded variables
naiveBayes	51778	–	46524	–	NumberOfClasses^2, NumberOfInstances^2
rpart	47240	sqrt(maxdepth)	99973	–	log(NumberOfInstances), NumberOfClasses, NumberOfClasses^2, log(MajorityClassSize), minbucket^2, minsplit^2, log(minsplit), sqrt(maxdepth), log(maxdepth)
ranger	99999	sqrt(NumberOfClasses), sample.fraction	–	No result: Cholesky factorisation failed.	No result: `step()` could not be applied since algorithm did not converge.
glmnet	99998	–	99992	–	NumberOfInstances^2, MajorityClassSize^2, log(alpha), sqrt(alpha)
gbm	40	MajorityClassSize, NumberOfClasses, NumberOfInstances, NumberOfInstances^2, sqrt(NumberOfInstances), NumberOfFeatures, NumberOfFeatures^2 , sqrt(NumberOfClasses), MajorityClassSize^2, sqrt(MajorityClassSize), log(MajorityClassSize), n.trees, n.trees^2 , interaction.depth, interaction.depth^2, log(interaction.depth), shrinkage, shrinkage^2, sqrt(shrinkage), log(shrinkage), bag.fraction, bag.fraction^2	212	sqrt(NumberOfInstances), shrinkage, sqrt(shrinkage), bag.fraction^2	interaction.depth^2, log(interaction.depth)
nnet	99551	–	99880	–	MajorityClassSize^, size, log(maxit)

Table 4.5: Results of variable selection: excluded variables from models on prediction time. Variable selection was obtained due to boosting (glmboost and gamboost) or step-wise selection regarding glm for each classifier. Additionally, the number of boosting iterations (m_{stop}) of the boosting algorithms is tabulated.

	glmboost		gamboost		glm
	m_{stop}	Not selected variables	m_{stop}	Not selected variables	Not selected variables
naiveBayes	20	NumberOfInstances, NumberOfInstances^2, NumberOfFeatures^2, NumberOfClasses, NumberOfClasses^2, sqrt(NumberOfClasses), log(NumberOfClasses), MajorityClassSize, MajorityClassSize^2, sqrt(MajorityClassSize), log(MajorityClassSize)	561	–	
rpart	99997	maxdepth	99964	–	NumberOfInstances^2, log(NumberOfInstances), minsplit, minbucket, minbucket^2, cp, cp^2, sqrt(cp), log(cp), maxdepth, maxdepth^2, log(maxdepth)
ranger	99117	sample.fraction	–	No result: Cholesky factorisation failed.	sqrt(NumberOfInstances), sample.fraction^2, log(MajorityClassSize), sample.fraction^2, log(sample.fraction), num.trees, replace,
glmnet	99394	–	100000	–	NumberOfInstances^2, log(NumberOfFeatures), sqrt(MajorityClassSize), alpha, alpha^2
gbm	45	NumberOfInstances, NumberOfInstances^2, NumberOfFeatures, NumberOfFeatures^2, sqrt(NumberOfFeatures), log(NumberOfFeatures), NumberOfClasses, NumberOfClasses^2, sqrt(NumberOfClasses), MajorityClassSize, log(MajorityClassSize), n.trees^2, interaction.depth, interaction.depth^2, shrinkage, shrinkage^2, sqrt(shrinkage), log(shrinkage), bag.fraction, bag.fraction^2, sqrt(bag.fraction), log(bag.fraction)	1006	intercept	NumberOfInstances^2, sqrt(MajorityClassSize), log(MajorityClassSize), interaction.depth, interaction.depth^2, sqrt(interaction.depth), shrinkage, shrinkage^2, sqrt(shrinkage), log(shrinkage), log(bag.fraction)
nnet	97595	skip	100000	–	sqrt(NumberOfInstances), MajorityClassSize^2, log(size), maxit^2, log(maxit), sqrt(maxit), decay^2

`minsplit`, `minbucket` and `cp`, as well as the characteristics of the dataset on runtime were analysed.

The results of gamboost on training time, the model with the lowest RMSE, are visualised in A.15. The linear partial effects, indicate that training time increases for NumberOfInstances, NumberOfFeatures, NumberOfClasses and hyperparameter `minsplit`, while it decreases for MajorityClassSize, `minbucket`, and `cp`. Combining the linear effect with the smooth effect, results in partial effect plots, which are quite rough.

A clearer picture of the partial effects, however, can be obtained from the results of the glmboost model plotted in A.3. The effect of NumberOfInstances, NumberOfFeatures and MajorityClassSize on training time is predominately the combination of a logarithmic and a square root effect. Training time increases linearly with the number of classes, while it decreases quadratically for an increase in the depth of a tree (`maxdepth`). A negative quadratic effect can also be observed for the complexity parameter (`cp`) for values of `cp` greater than approximately 0.0125. The effect of `minsplit` and `minbucket` on training time is approximately quadratic, except for values of these hyperparameter that are near zero.

In terms of the variable importance measures obtained from the random forest fit, the four most important variables are the dataset characteristics (left side of figure A.25 and A.26), independently of the response variable. With respect to the associated partial dependence plots (figure A.37), there is a monotonically increasing relationship between the NumberOfFeatures, NumberOfInstances and MajorityClassSize and the predicted training time. With respect to prediction time (figure A.38), a monotonically increasing relationship can only be observed for NumberOfFeatures and NumberOfInstances. The rest of the predictors do not correlate with the predicted runtime.

The only effect that was excluded by the boosting process of glmboost on prediction time was the linear effect of `maxdepth`. However, most of the estimated coefficients tabulated in B.9 have values below 0.0005. Since the coefficients are rounded to the third decimal place for clarity, an effect of 0 is tabulated for values below 0.0005. This concerns almost all quadratic effects, and the linear effects for MajorityClassSize, NumberOfInstances, `minsplit` and `minbucket`. The corresponding partial effects are displayed in A.4. The shape of the partial effect for NumberOfInstances and `maxdepth` are similar and are a combination of a relevant logarithmic and a square root effect. First, prediction time increases, sharply – in case of NumberOfInstances, plateaus for a short distance and slightly drops down for more than 20,000 instances or a tree depth greater than 15. The partial effect for NumberOfFeatures is approximately logarithmic. An increase in NumberOfClasses leads to a linear decrease in prediction time. Prediction time of MajorityClassSize is shortest for an approximate size of 10,000. Class sizes that are smaller or greater yield a greater prediction time. For small values of `minsplit` and `minbucket` prediction time can be very short or quite long. Values of complexity parameter `cp` greater than approximately 0.02 result in a smaller

estimated prediction time.

Random forest

Runtime was also analysed for an R implementation of random forest called `ranger`, which is stated to be a "fast implementation of random forests" (Wright, 2016). The following hyperparameters of `ranger` were analysed regarding their influence on runtime: `num.trees`, `mtry`, `sample.fraction`, `respect.unordered factors` and `replace`.

When fitting a random forest on training time, the most important variable is `respect.unordered.factors`, followed by the dataset characteristics (left side of figure A.27). When fitting it on prediction time, the most important variable is `num.trees`, also followed by the dataset characteristics (left side of figure A.28). But values of mean decrease in accuracy are already quite low for NumberOfFeatures and NumberOfClasses. This is also demonstrated in the partial dependence plots of figure A.40, where an approximate linear relationship between `num.trees`, NumberOfInstances and MajorityClassSize and the estimated prediction time can be assumed, while the other covariates do not have an influence.

The partial effects, obtained from glmboost on training time, for NumberOfInstances, `num.trees`, `sample.fraction` and `mtry` are predominantly a combination of a logarithmic and a square root effect (see figure A.5). This means, that after an initial, sharp increase in training time, for small values of these covariates, the training time stabilises or slightly continues to grow, after reaching the highest point. The partial effect of NumberOfClasses is the sum of an estimated coefficient of -0.154 for the linear, of 0.013 for the quadratic and of 0.489 for the logarithmic effect (see table B.10). Altogether, however, this leads to an almost linear shape for the combined effect. The estimated coefficient for sampling with replacement (`replace = TRUE`) is -0.1. This implies, that sampling with replacement changes training time by a factor of $exp(-0.1) = 0.905$, with all other predictors unchanged. If `respect.unordered.factors` is set to `TRUE` training time increases by a factor of 2.028, with all other predictors held constant.

The results for the glmboost model on prediction time are tabulated in table B.10 and illustrated in figure A.6. For the covariates NumberOfInstances, `num.trees` and `sample.fractions` the shape of the effects are quite similar. While the effect of MajorityClassSize has an approximately positive linear effect on training time, it has a negative quadratic effect on prediction time, with the highest outcome for 9 classes. Also for `mtry` the effect on runtime reverses: for training time an increase in `mtry`, leads to an increase in estimated training time, while an increase of `mtry` for values greater than 200 leads to a decrease in prediction time. If `respect.unordered.factors` is set to TRUE, the on average expected prediction time increases by a factor of 1.038, given all other covariates remain constant. For `replace = TRUE` prediction time changes by a factor of 0.992.

Generalized linear model with lasso or elastic-net regularisation

As representative for generalised linear models, runtime of `glmnet` was analysed. The analysed covariates include the characteristics of the dataset and `alpha`, which is a hyperparameter of `glmnet` controlling the elastic-net penalty.

The gamboost model presents the best performance in terms of prediction accuracy, when modelling training time. The final model contained all effects of all covariates[1]. The smooth effects of the covariates are quite rough, leading also to rough combined partial effects (see A.17). If the fluctuations are not taken into account, training time increases with the NumberOfInstances, NumberOfFeatures, NumberOfClasses and `alpha`. Regarding MajorityClassSize no trend on training time can be suggested as the curve fluctuates greatly. The partial effect of `alpha` on training time, firstly, increases linearly. Then it fluctuates on a high level until it finally drops.

The results for glmboost on training time reveal similar trends for NumberOfInstances, NumberOfFeatures and NumberOfClasses. For MajorityClassSize a negative, linear effect can be determined: an increasing size of the majority class, results in a decrease of training time. Hyperparameter `alpha` has a positive influence on the predicted training time, after an initial small drop.

The glmboost model on prediction time did not exclude any variable in the boosting process and has a m_{stop} value near the upper boundary of the searched grid. The combined partial effects displayed in A.8 demonstrate a shape for the effect of NumberOfInstances on prediction time, that results from a sum of a square root effect of 0.072 and a logarithmic effect of -0.468 (see B.11 for the estimated coefficients). The shape for NumberOfFeatures is a parabola with a negative sign, with the highest point for about 200 features. Regarding NumberOfClasses the effect on estimated prediction time is a combination of all effects. From 2 to 4 classes prediction time increases, from 4 to 8 classes it plateaus and then it increases again. Except for small values of `alpha`, where prediction time could be either low or high, increasing values of `alpha` do not have an effect on the estimated prediction time.

The least important variables for the random forest fit on training time as well as on prediction time is `alpha`. On the contrary, the dataset characteristics, headed by NumberOfFeatures and followed by NumberOfClasses, MajorityClassSize and NumberOfInstances, are important variables (see left side of figure A.29 and A.30).

Generalised boosted regression modelling

For classifier `gbm`, a boosting algorithm, the influence of the characteristics of the dataset and the influence of four of its hyperparameters on runtime were analysed.

[1]Note, that m_{stop} is 99,992 and variable selection probably did not occur, because the appropriate m_{stop} was not found yet.

These hyperparameters include: the number of trees (`n.trees`), the size of the tree (`interaction.depth`), the learning-rate (`shrinkage`), and the fraction of the training set on which the next tree in the expansion is fitted (`bag.fraction`).

The best fit with respect to RMSE for modelling training time was achieved by a glm. The final model after stepwise variable selection included all variables except for the quadratic and logarithmic effect of `interaction.depth`, the quadratic effect of `bag.fraction` and the square root effects of NumberOfInstances and `shrinkage`. The model results are tabulated in table B.12 and visualised in figure 4.4. The only estimated coefficient greater than 0.0005 for NumberOfInstances is

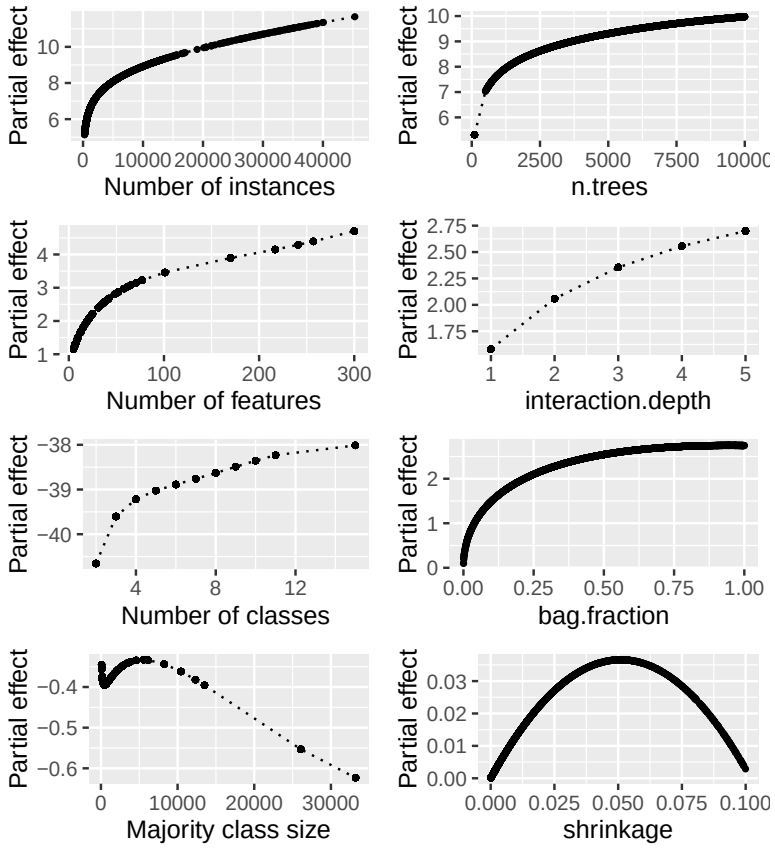


Figure 4.4: Combined functional estimates of `glm` modelling training time.

the estimate for the logarithmic effect. Moreover, the shape of the effects of NumberOfFeatures, NumberOfClasses, `n.trees` and `bag.fraction` resemble the shape of the effect for NumberOfInstances, indicating that the estimate for the logarithmic effect has a strong influence here. Function estimates for `interaction.depth` are approximately linear and negative quadratic for `shrinkage`. The effect for MajorityClassSize is a combination of a small positive square root effect and a negative logarithmic effect. This implies, that after an initial drop for very small class sizes, the estimated training time increases until it reaches its peak for a size of about 5,000. For majority classes greater than that, the estimated training time decreases monotonically.

Comparing the results of the glm to glmboost visually (compare figure 4.4 to A.9), demonstrates that the shape of the effects are approximately the same for the modelling approaches for covariates included in both models. However, the estimated coefficients that contribute to that shape differ in their orders of magnitude (for the estimated coefficients refer to B.12), and sometimes even coefficients from different modelling alternatives lead to an approximately similar shape. The boosting process of glmboost excluded all modelling alternatives for MajorityClassSize and `shrinkage`, while the glm attributes a approximately negative, linear effect for MajorityClassSize for values greater than 5,000, and a negative, quadratic effect of `shrinkage` on training time.

The boosting process of glmboost on prediction time also excluded all modelling alternatives for `shrinkage` from the final model. Hence, `shrinkage` has no influence on runtime when using the glmboost modelling approach. Moreover, NumberOfFeatures and `bag.fraction` have no influence on the prediction time of `gbm`. Otherwise, function estimates for the model on training time and the model on prediction time differ only slightly. The only relevant function estimate for MajorityClassSize, which was excluded by glmboost on training time, is a negative square root effect, i.e. for an increasing MajorityClassSize prediction time drops.

The RMSE of random forest on training time is the second lowest. The results of variable importance measurements are shown in figure A.31. With respect to mean decrease in accuracy the most important variable is `n.trees`, followed by NumberOfInstances, `interaction.depth` and `bag.fraction`. Hence, for runtime behaviour of the classifier `gbm` its hyperparameters do play an important role. The corresponding partial dependency plots (figure A.43) show that the relationship between predicted training time and `n.trees`, NumberOfFeatures, `interaction.depth`, `bag.fraction` and NumberOfClasses is approximately linear, while it seems to be logarithmic for NumberOfInstances and constant for MajorityClassSize and `shrinkage`. If we compare these results to the results obtained from the fit of random forest on prediction time, the most important variables are `n.trees`, NumberOfClasses, NumberOfInstances and `interaction.depth`. This indicates, that `interaction.depth` and `n.trees` are important for modelling training time and prediction time.

Single-hidden layer neural network

One of many implementations of neural networks is `nnet`, a single-hidden layer neural network. In this runtime analysis the effect of the characteristics of the dataset and four algorithm specific hyperparameters were investigated. The latter include the number of units in the hidden layer (`size`), whether or not to add a skip-layer connection (`skip`), the regularisation parameter (`decay`) and the maximum number of iterations (`maxit`). Note, that the logarithmic effect for decay needed to be omitted, since there were zero values for this covariate.

Again random forest demonstrates the best fit among all four regression methods,

both on training time and prediction time. The results of variable importance measurements of random forest on training time are presented in A.33. Both importance measures agree on the most important variable: NumberOfFeatures. With respect to mean decrease in accuracy, the other three most important variables are MajorityClassSize, `skip` and NumberOfInstances. Comparing this to the results obtained on prediction time (see left side of figure A.34), shows that NumberOfFeatures is also the most important variable for modelling prediction time. Furthermore, the other characteristics of the dataset have a major impact on prediction time.

The results of the analysis with glmboost are shown in figure A.11 for training time and in figure A.12 for prediction time. The function estimate of the logarithmic modelling alternative for NumberOfInstances, NumberOfFeatures and `maxit` dominates the influence on training time, leading to an approximately logarithmic partial effect of these covariates on training time. Training time increases linearly with an increasing `size`, hence, an increasing number of units in the layer. Time also increases for an increasing NumberOfClasses, but not linearly. In contrast, training time decreases linearly with increasing MajorityClassSize. Allowing for skip-layer connections, changes training time by a factor of 0.963, with all other covariates held constant.

The partial effect plots of NumberOfInstances and NumberOfFeatures, derived from glmboost on prediction time, show a similar shape: after an initial increase of estimated prediction time, training time slowly drops down again for more than 25,000 instances, and more than 200 features respectively. NumberOfClasses and MajorityClassSize overall have a negative effect on estimated prediction time. However, the decrease of prediction time for increasing covariate values is not linear, and the shape demonstrated here, results from the sum of the estimated coefficients. The estimated prediction time has its peak for `size = 1`, which refers to one unit in the hidden layer, and then drops down for two units, increases until a second peak for 6 and 7 units is reached. Then prediction time drops down again for more than 7 units. Function estimates of `decay` and `maxit` result in a similar shape of the effect. After a sharp increase in training time for small values of the respective covariates, the increase slows down, plateaus and drops for values greater than 0.5 for `decay` and 500 for `maxit`. Setting `skip = TRUE` leads to a change by a factor of 0.924, with all other variables held constant.

5 Conclusion

A major goal of every machine learning project is to be as accurate and efficient as possible (Doan and Kalita, 2016). This implies, the selection of one or a couple of appropriate machine learning algorithms in terms of accuracy and computational efficiency for a given set of problems. Hence, prediction of runtime plays a crucial role in various fields of application of machine learning algorithms. The main objective of this thesis was to analyse and model runtime behaviour of different classification algorithms. For this purpose, a meta-learning approach was applied. Since runtime depends on the characteristics of the dataset and the hyperparameters of the classifier, meta-features concerning the dataset characteristics and the hyperparameters were used to create the meta-data for the regression model. Four different regression models (glmboost, gamboost, glm and randomForest) were compared, with a focus on gradient boosting with component-wise linear models.

The experiments included modelling runtime of six classifiers (`naiveBayes`, `rpart`, `ranger`, `glmnet`, `gbm` and `nnet`), which were run on 65 OpenML datasets, with four different regression methods. The goodness-of-fit of the regression models was assessed using two measures: RAE and RMSE.

With RAE, the four regression methods were compared to the average runtime. All regression models had a RAE below one, which indicates, that it is more precise to use the regression method than the average runtime in all cases.

The RMSE is a commonly used evaluation metric for regression problems and was compared between the four regression models. Random fores clearly outperformed the other regression models, having the lowest RMSE in all models on prediction time. Moreover, it performed best in all models on training time, except for models regarding classifiers `glmnet` and `gbm`. Also Hutter et al. (2014) found random forest to be the best prediction model, when comparing model performance of random forest against ridge regression methods, neural networks and gaussian process regression. Hutter et al. (2014) attribute the high performance of random forests on highly heterogeneous data to the fact that tree-based methods model different parts of the data differently. However, Hutter et al. (2014) did not assess the quality of prediction on training data, but evaluated model performance by using 10-fold cross-validation. In order to generalise predictive performance to new cases, strategies like cross-validation and bootstrap need to be implemented (Zou et al., 2012), since an evaluation only on training data easily leads to overoptimistic

and overfitted models (Saed, 2016). Therefore, future work should evaluate model performance of the four regression methods used in this thesis. With packages like `mlr` (Bischl et al., NA) or `caret` (Kuhn et al., 2016) the implementation is greatly supported. However, since models like `gamboost` are only partly or not yet included in these packages, one should consider some more time for setting up the evaluation.

To interpret the results of randomForest, variable importance was measured. The results indicate, which variables are most important, and can be used as a tool to reduce the number of variables for future regression methods on runtime. Regardless of the response variable, NumberOfInstances and NumberOfFeatures are among the four most important variables in all models, except for three cases. Most of the algorithm's hyperparameters are not among the three most important variables. Exceptions include `ranger`'s `respect.unordered.factors` and `gbm`'s `n.trees` for models on training time. With respect to models on prediction time, it is `ranger`'s `num.trees`.

Another approach to identify suitable meta-features – as proposed by Reif et al. (2011) and Priya et al. (2011), is to calculate the Pearson product moment correlation coefficient of the actual runtime and the predicted runtime for different sets of meta-features or single meta-features.

Descriptive analysis of the runtimes demonstrated that the distribution of runtime was extremely right-skewed with wide ranges of values, which makes it more difficult to predict runtime (Priya et al., 2011). Assuming that the dataset characteristics have a major influence on runtime, the reason for the skewness of runtime can be found in the right-skewed distribution of the meta-features relating to the dataset characteristics. Analysis and modelling would benefit from a more balanced selection of datasets.

The interpretation of the parametric models, which include several modelling alternatives for one covariate, was done by describing partial effect plots. The demonstrated effects for glmboost were in most cases the sum of the functional estimates of all modelling alternatives, and for gamboost rough curves, resulting from rough function estimates for the smooth component. This is partly, because the property of the boosting algorithms to select models and variables did in most cases not lead to sparser models. Secondly, this is because the search for an optimal m_{stop} often was not successful on a grid from 1 to 100,000. In this context Mayr et al. (2012) states, that the ability of the boosting algorithm to find sparser and more interpretable models with good prediction accuracy in a fully data driven manner is limited. The problem is selecting the optimal stopping iteration m_{stop}. Commonly, the algorithm is run to a large m_{stop} and the optimal number of boosting iterations is assessed based on a information criteria (e.g. AIC) or resampling strategies, with the latter being applied in the present experiments. This tends to stop the model too late, which results in large models, and can lead to substantial overfitting. To overcome this problem, Mayr et al. (2012) introduced a new sequential stopping rule, which is fully data-driven and frees the user from defining

an upper boundary for the search grid. Another approach for more stable variable and model selection is to apply stability selection, a function that comes with the `mboost` package (Hothorn et al., 2016).

In addition to the dataset characteristics and the algorithm's hyperparameters, future work on runtime analysis, should include statistical, information theoretic, model-based, landmarking and time-based meta-features. Time-bsaed meta-features are calculated in order to take the user's machine into account. Besides time-based meta-features, one could also use measures that are related to the current state of the computational environment, as proposed by Priya et al. (2011). These measures are, for example the current free memory and the current CPU idle in the machine.

In the experimental setup for this thesis each classification algorithm was terminated after a captime of 1,000 seconds. This leads to observations that are terminated before finishing. Hutter et al. (2014) name those observations *capped* runtime observations. In this thesis capped observations were excluded from analysis, while Hutter et al. (2014) treated *capped* observations as if they have finished at *captime*. But there are alternatives to these approaches: one can also consider these observations as right-censored data and build regression models with censored data. Hutter et al. (2014) mentions some methods that could be considered for modelling censored data: gaussian processes, random forest for censored data and non-parametric kaplan meier estimators.

Altogether, future work should analyse runtime by using the meta-learning approach. The influence of traditional and more sophisticated meta-features with respect to dataset characteristics, the algorithm's hyperparameters and the status of the user's machine should be analysed. Moreover, the classification algorithms should be run on a balanced selection of datasets. Before modelling runtime, sophisticated variable selection methods should be applied in order to get sparse and interpretable models. Prediction of runtime with regression methods should be evaluated with cross-validation or bootstrap strategies and appropriate performance metrics. Besides, the already investigated regression methods, new methods could be analysed and compared, especially those that are able to model right censored data. Additionally, interaction terms between the covariates should also be taken into account.

However, the approach of this thesis and the suggestion for future work only refers to modelling runtime. It would be very interesting to model the algorithm's runtime and prediction accuracy at the same time, since in applications one searches for the most accurate and most efficient algorithm to solve a specific problem. Central questions in these investigations might be whether a trade-off between accuracy and efficiency needs to be accepted, or if there are algorithms that provide both: accuracy and efficiency.

Appendices

A Graphs

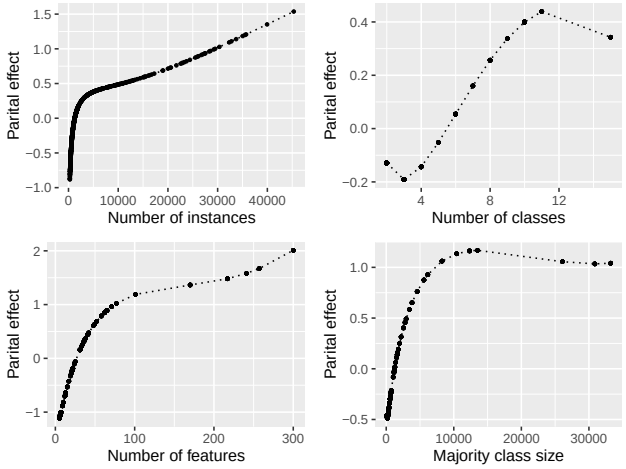


Figure A.1: Partial effects of the covariates obtained from glmboost on training time for classifier `naiveBayes`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

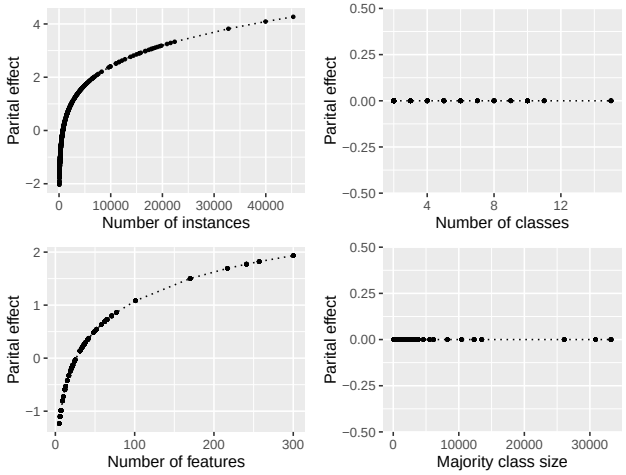


Figure A.2: Partial effects of the covariates obtained from glmboost on prediction time for classifier `naiveBayes`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

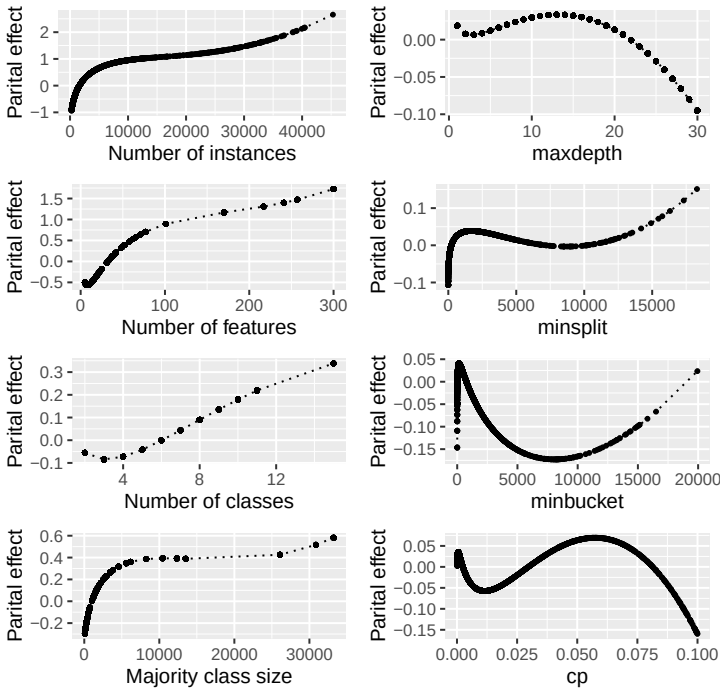


Figure A.3: Partial effects of the covariates obtained from glmboost on training time for classifier `rpart`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

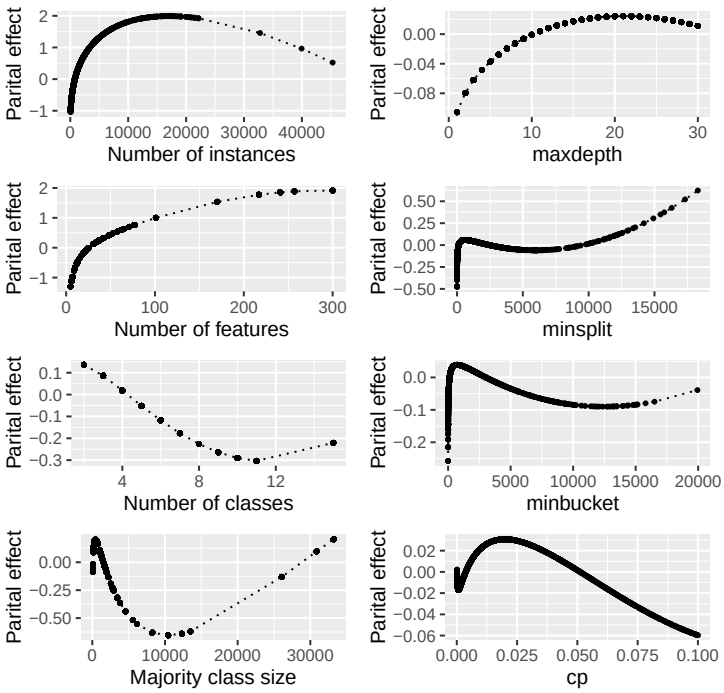


Figure A.4: Partial effects of the covariates obtained from glmboost on prediction time for classifier `rpart`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

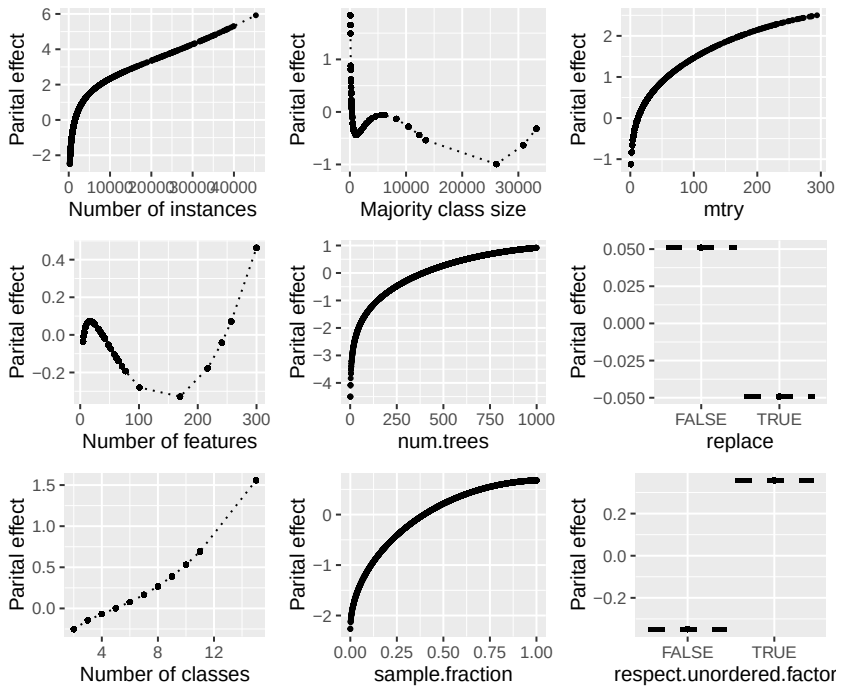


Figure A.5: Partial effects of the covariates obtained from glmboost on training time for classifier `ranger`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

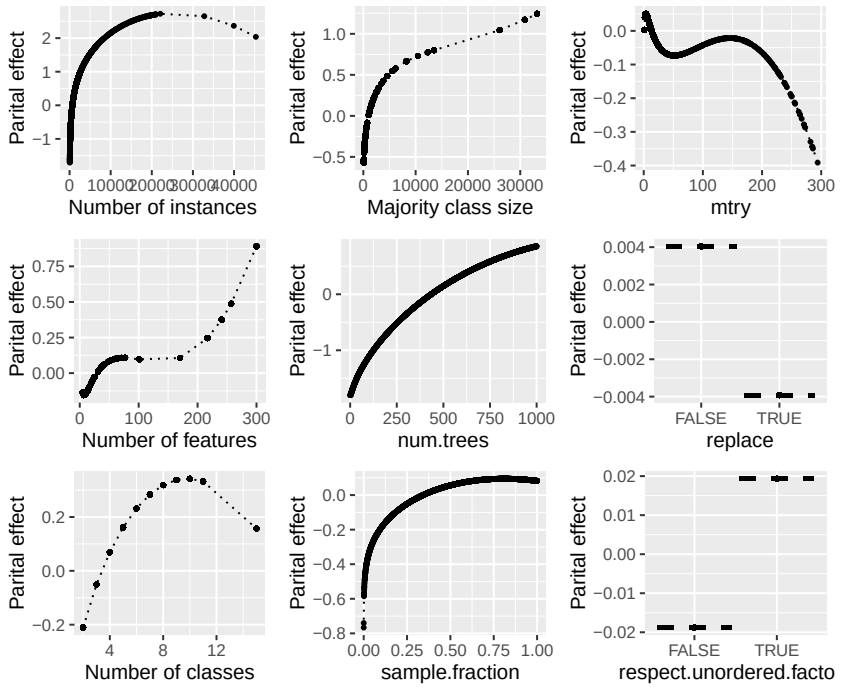


Figure A.6: Partial effects of the covariates obtained from glmboost on prediction time for classifier `ranger`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

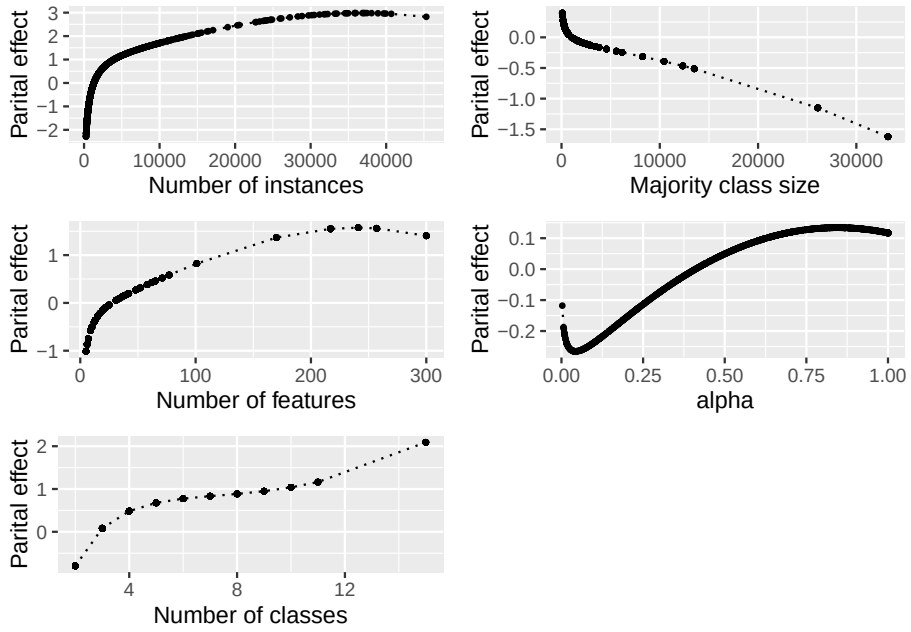


Figure A.7: Partial effects of the covariates obtained from glmboost on training time for classifier `glmnet`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

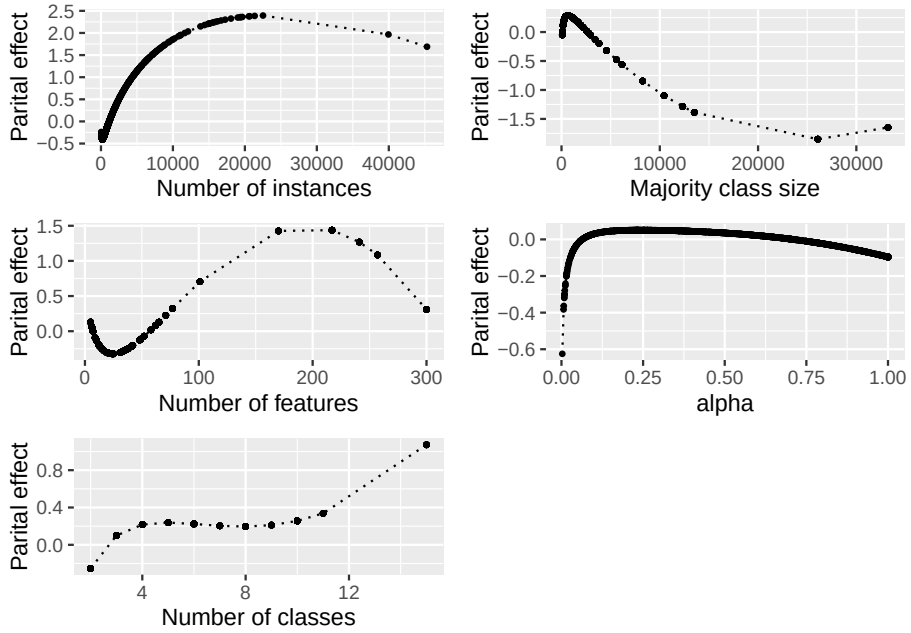


Figure A.8: Partial effects of the covariates obtained from glmboost on prediction time for classifier `glmnet`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

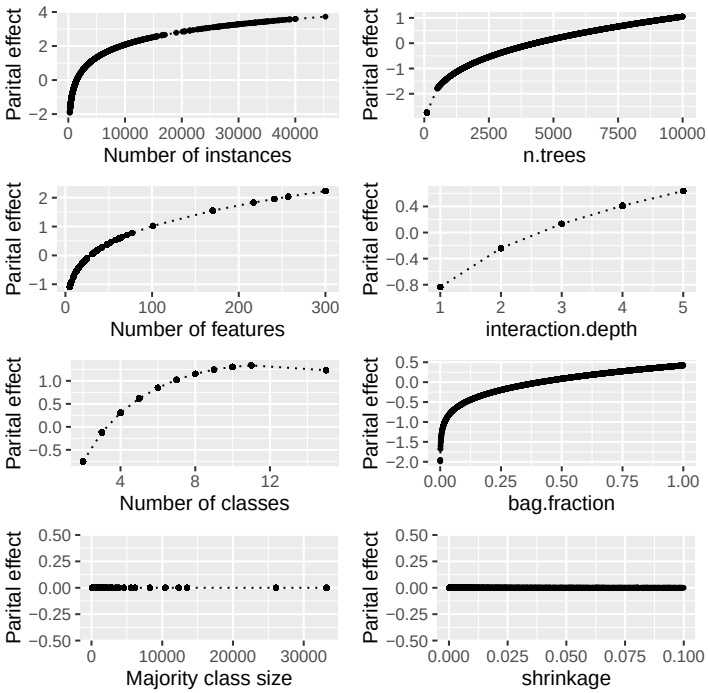


Figure A.9: Partial effects of the covariates obtained from glmboost on training time for classifier `gbm`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

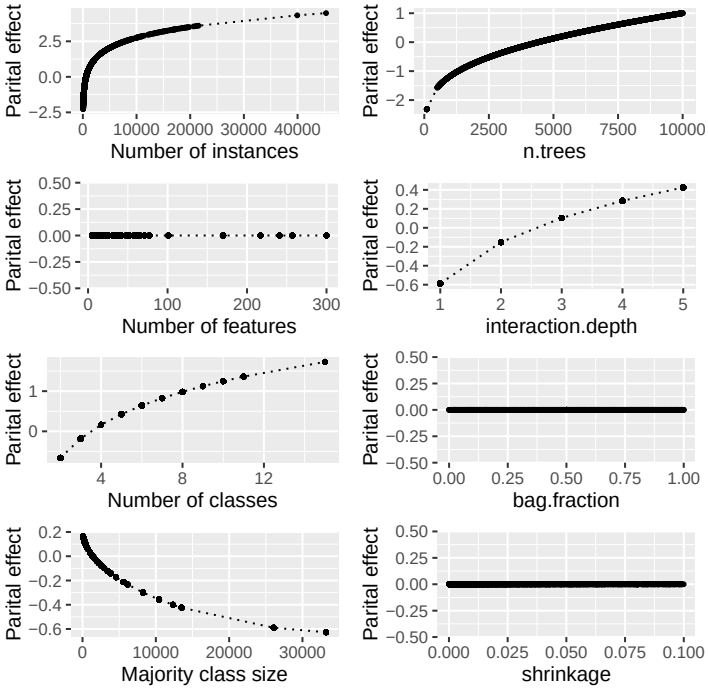


Figure A.10: Partial effects of the covariates obtained from glmboost on prediction time for classifier `gbm`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

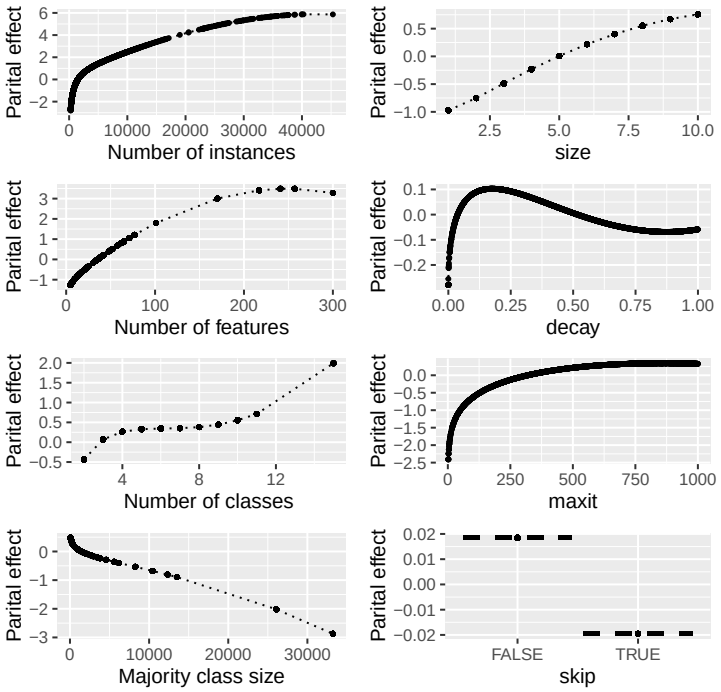


Figure A.11: Partial effects of the covariates obtained from glmboost on training time for classifier `nnet`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

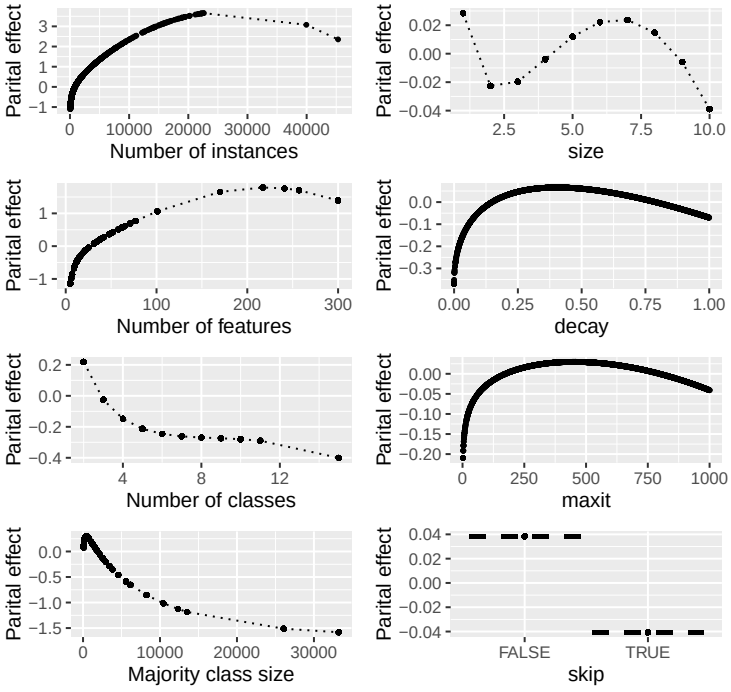


Figure A.12: Partial effects of the covariates obtained from glmboost on prediction time for classifier `nnet`. The partial effects are obtained by summing the functional estimates of the modelling alternatives (linear, quadratic, square root, logarithmic) for each covariate.

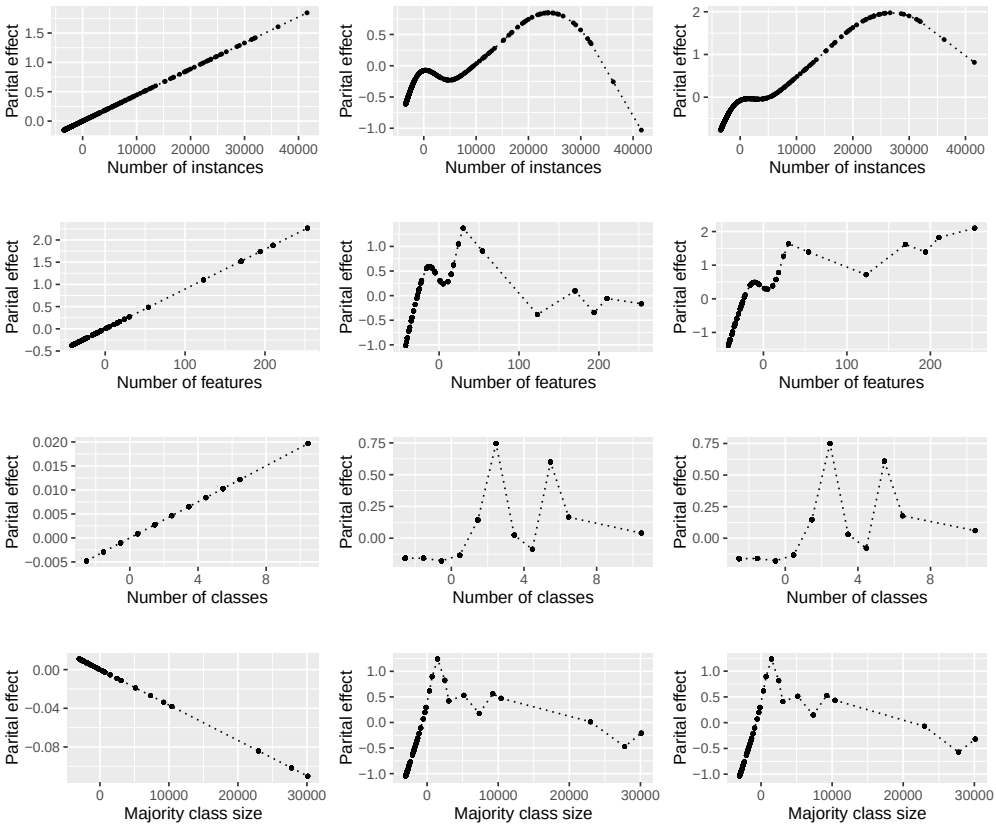


Figure A.13: Linear partial effect (left), smooth partial effect (center) and sum of the linear and smooth partial effect (right) for each covariate of the gamboost model on training time for the `naiveBayes` classifier.

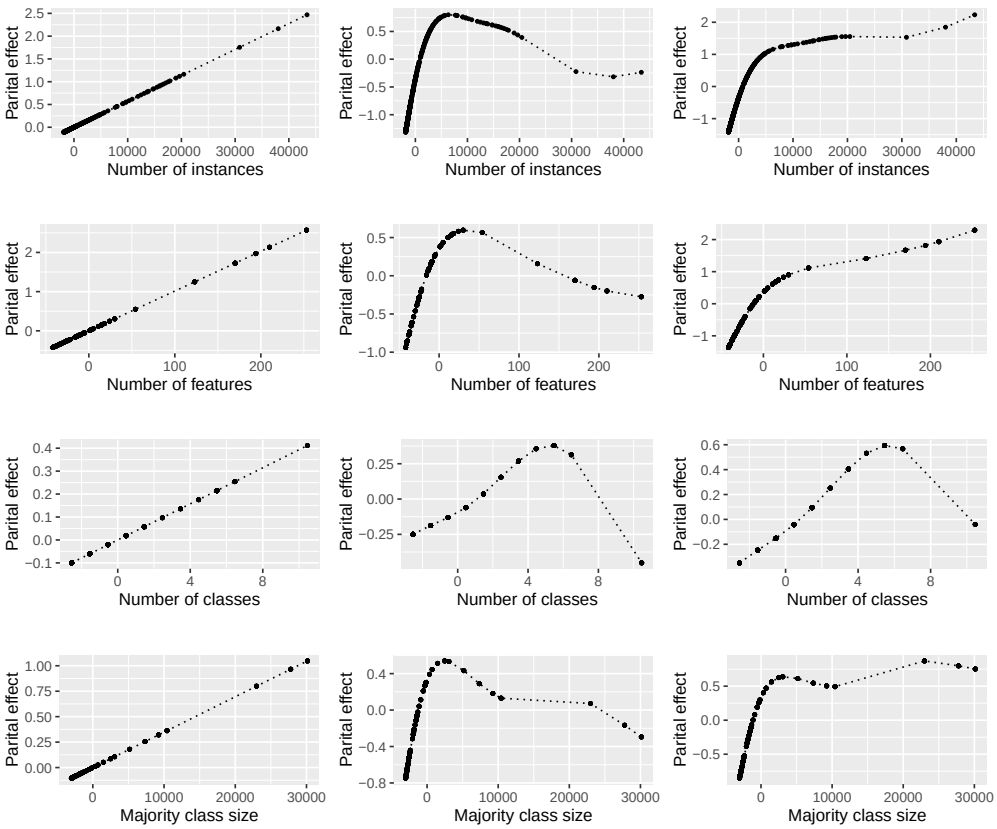


Figure A.14: Linear partial effect (left), smooth partial effect (center) and sum of the linear and smooth partial effect (right) for each covariate of the `gamboost` model on prediction time for the `naiveBayes` classifier.

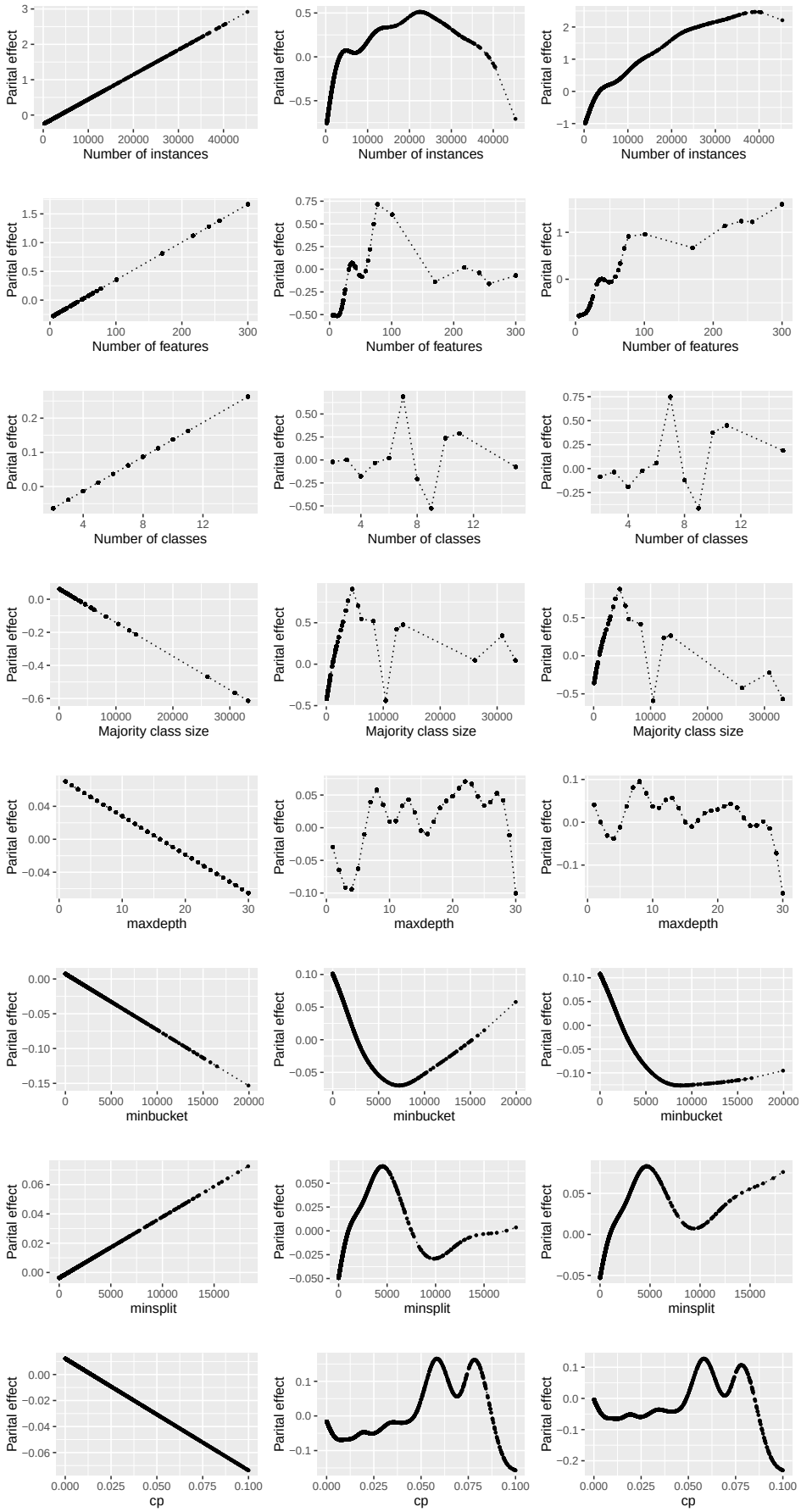


Figure A.15: Linear partial effect (left), smooth partial effect (center) and sum of the linear and smooth partial effect (right) for each covariate of the gamboost model on training time for the `rpart` classifier.

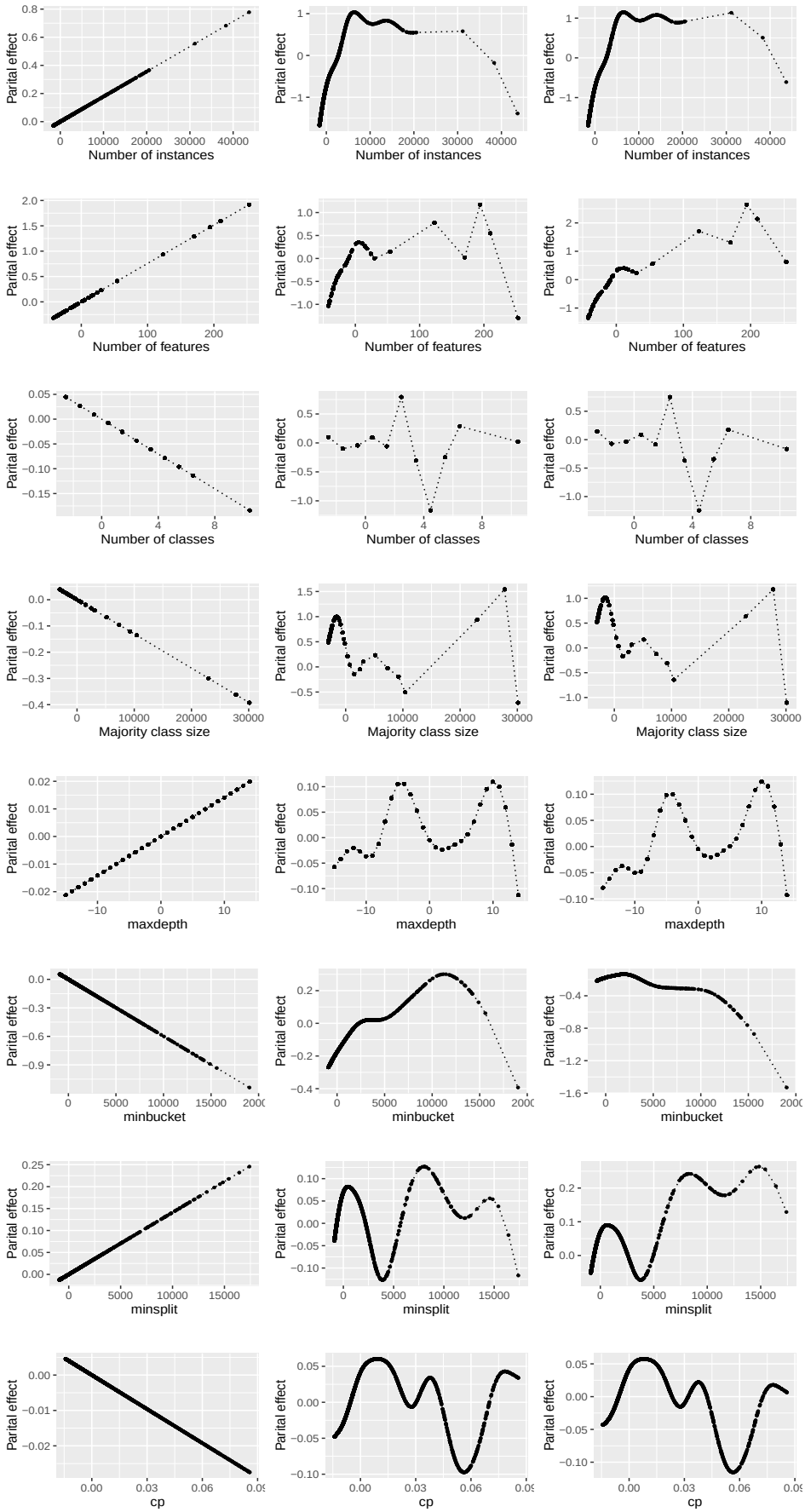


Figure A.16: Linear partial effect (left), smooth partial effect (center) and sum of the linear and smooth partial effect (right) for each covariate of the gamboost model on prediction time for the `rpart` classifier.

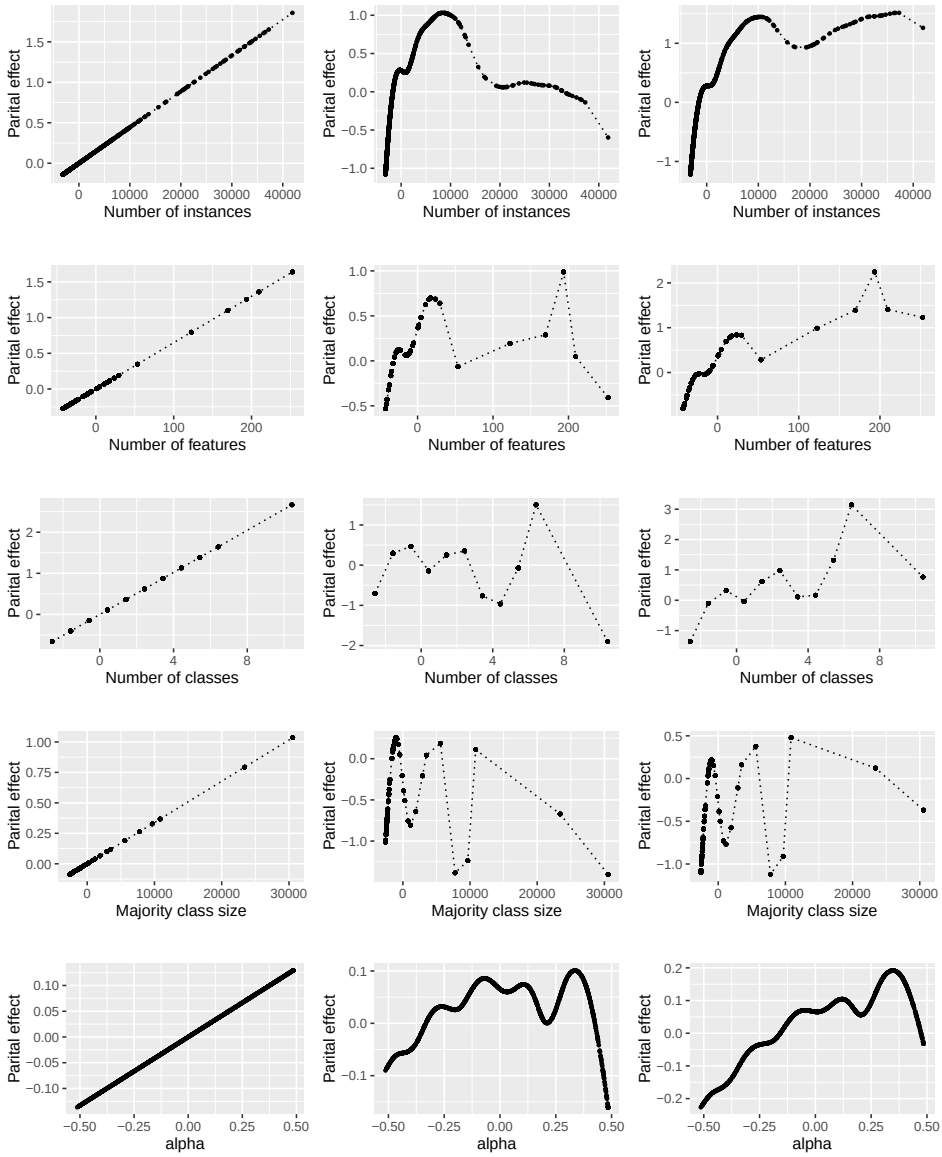


Figure A.17: Linear partial effect (left), smooth partial effect (center) and sum of the linear and smooth partial effect (right) for each covariate of the gamboost model on training time for the `glmnet` classifier.

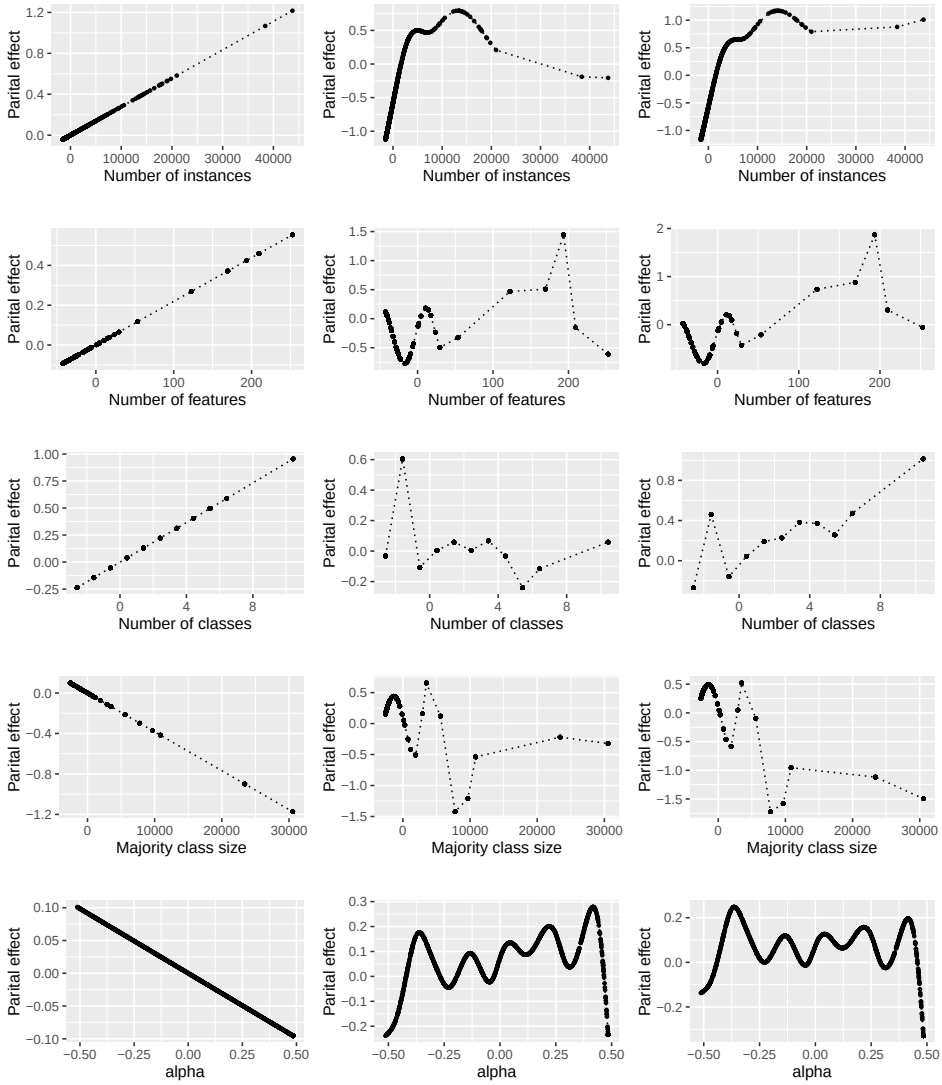


Figure A.18: Linear partial effect (left), smooth partial effect (center) and sum of the linear and smooth partial effect (right) for each covariate of the gamboost model on prediction time for the `glmnet` classifier.

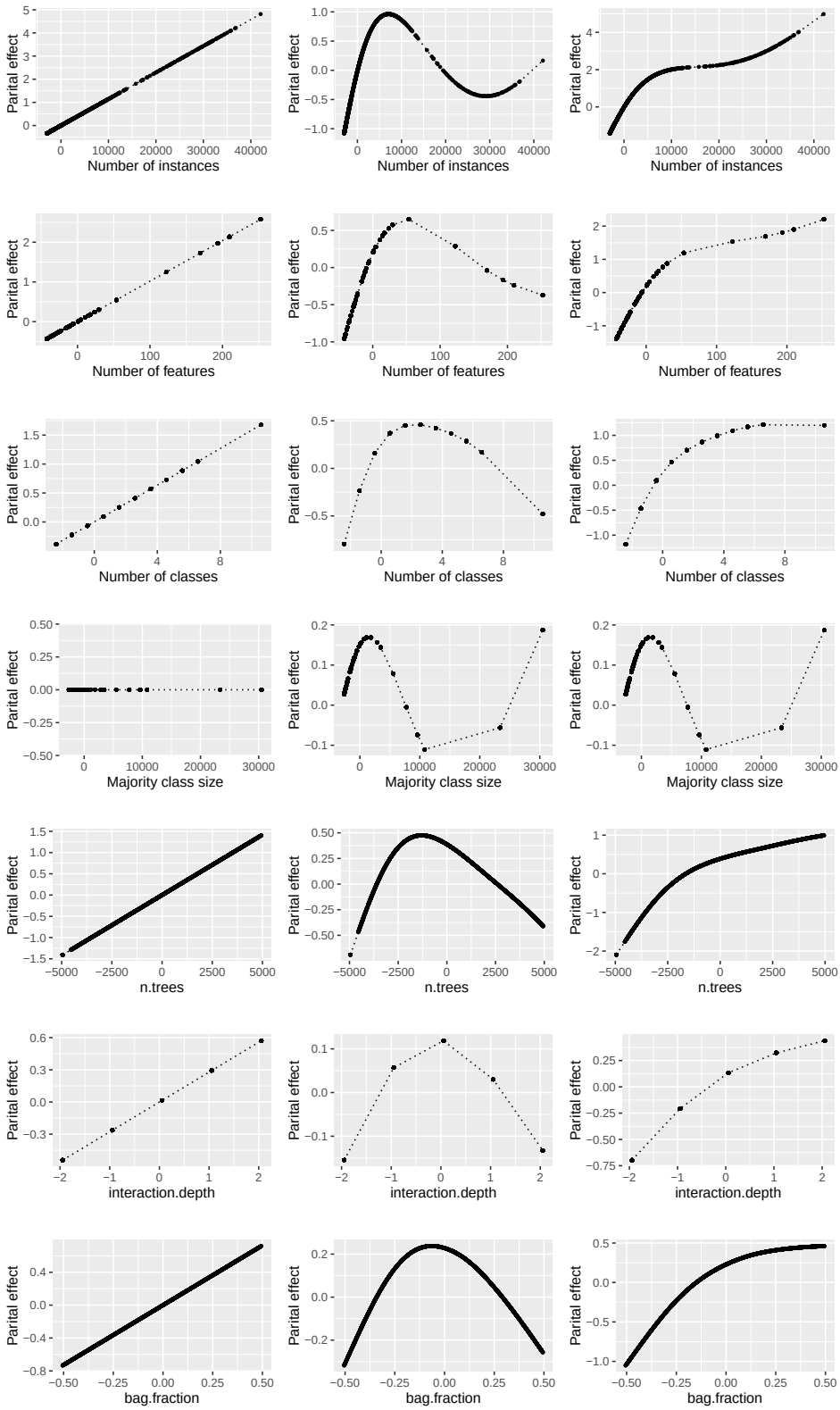


Figure A.19: Linear partial effect (left), smooth partial effect (center) and sum of the linear and smooth partial effect(right) for each covariate of the gamboost model on training time for the `gbm` classifier.

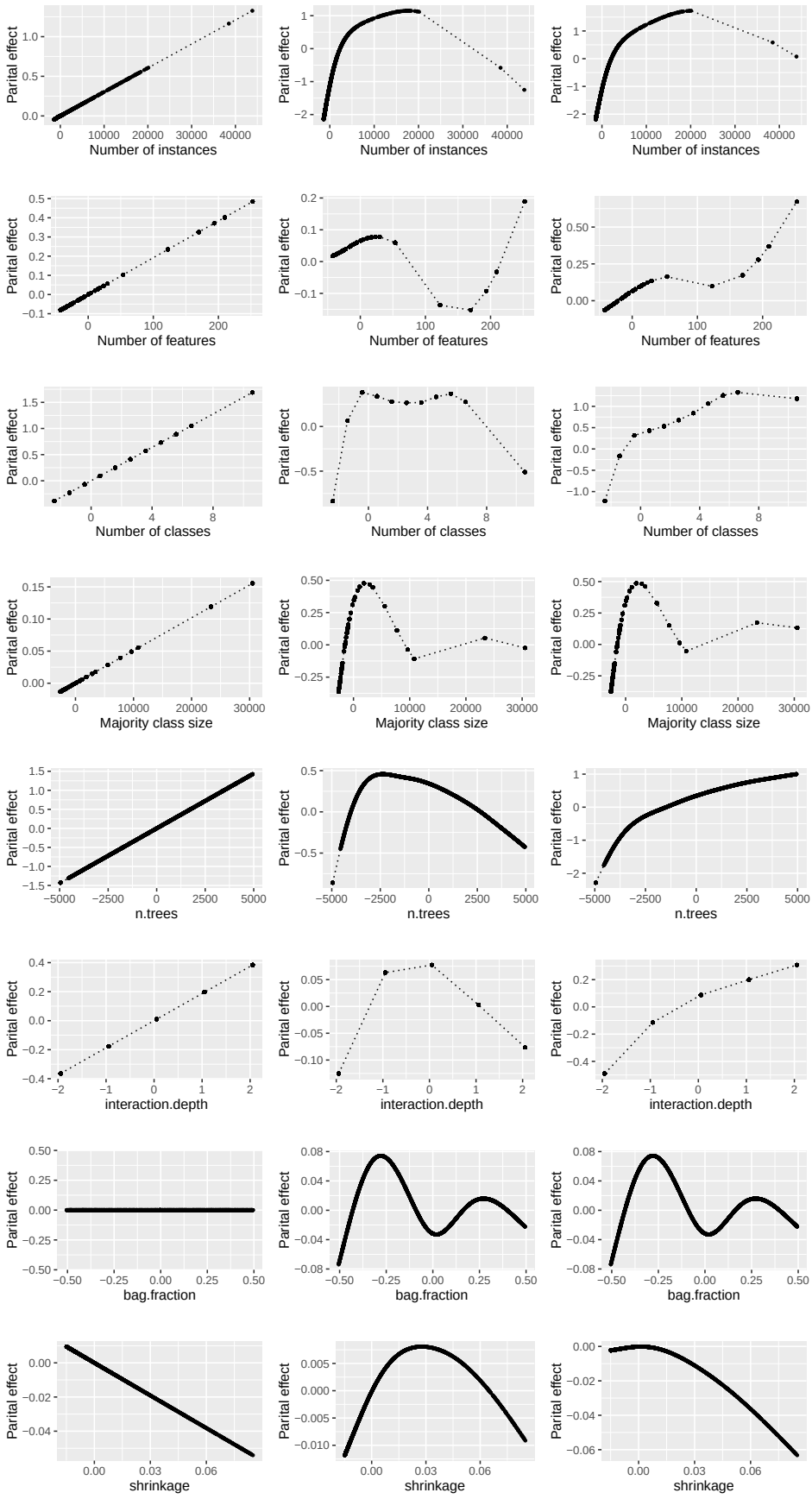


Figure A.20: Linear partial effect (left), smooth partial effect (center) and sum of the linear and smooth partial effect (right) for each covariate of the gamboost model on prediction time for the `gbm` classifier.

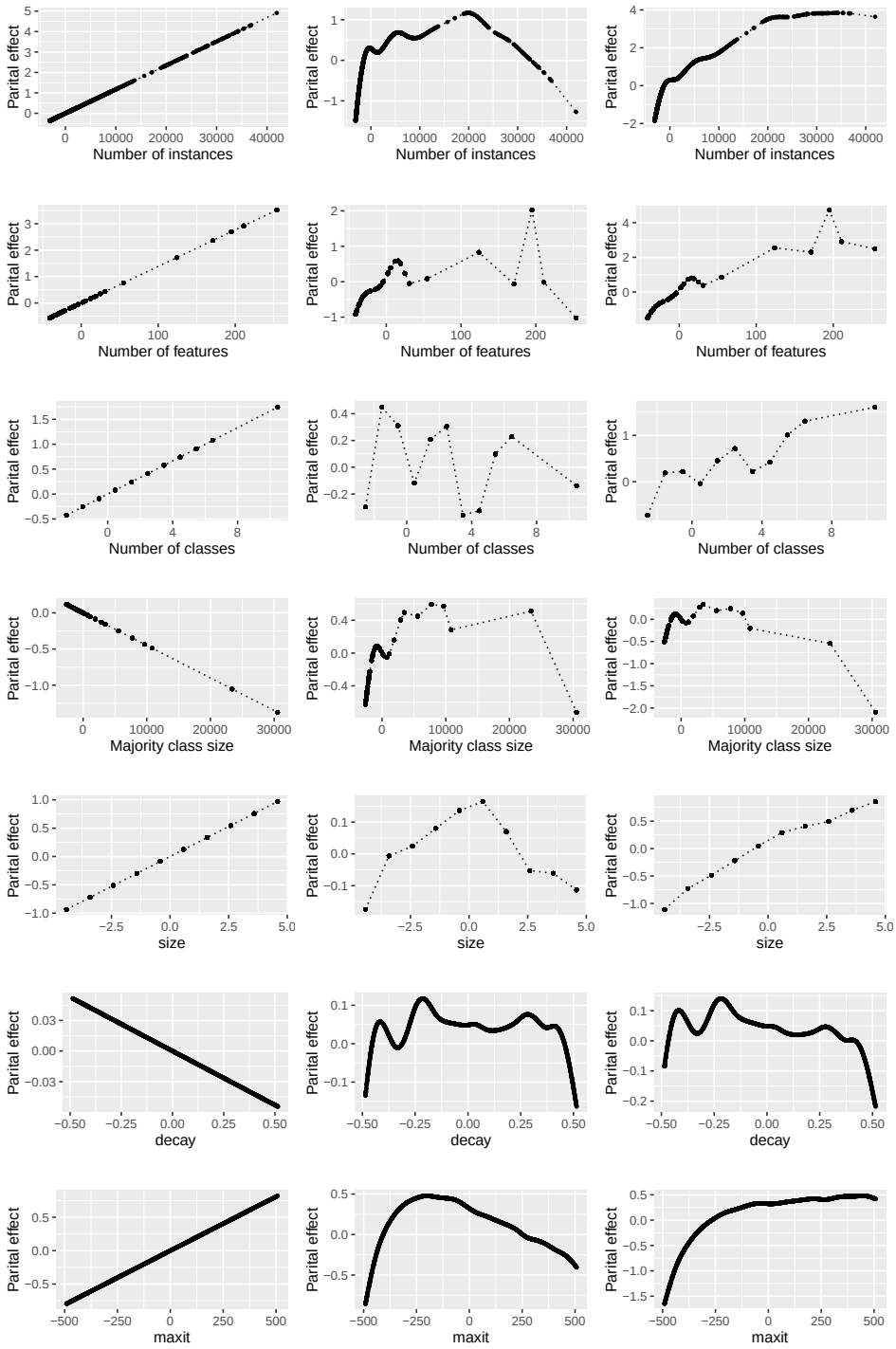


Figure A.21: Linear partial effect (left), smooth partial effect (center) and sum of the linear and smooth partial effect (right) for each covariate of the gamboost model on training time for the `nnet` classifier.

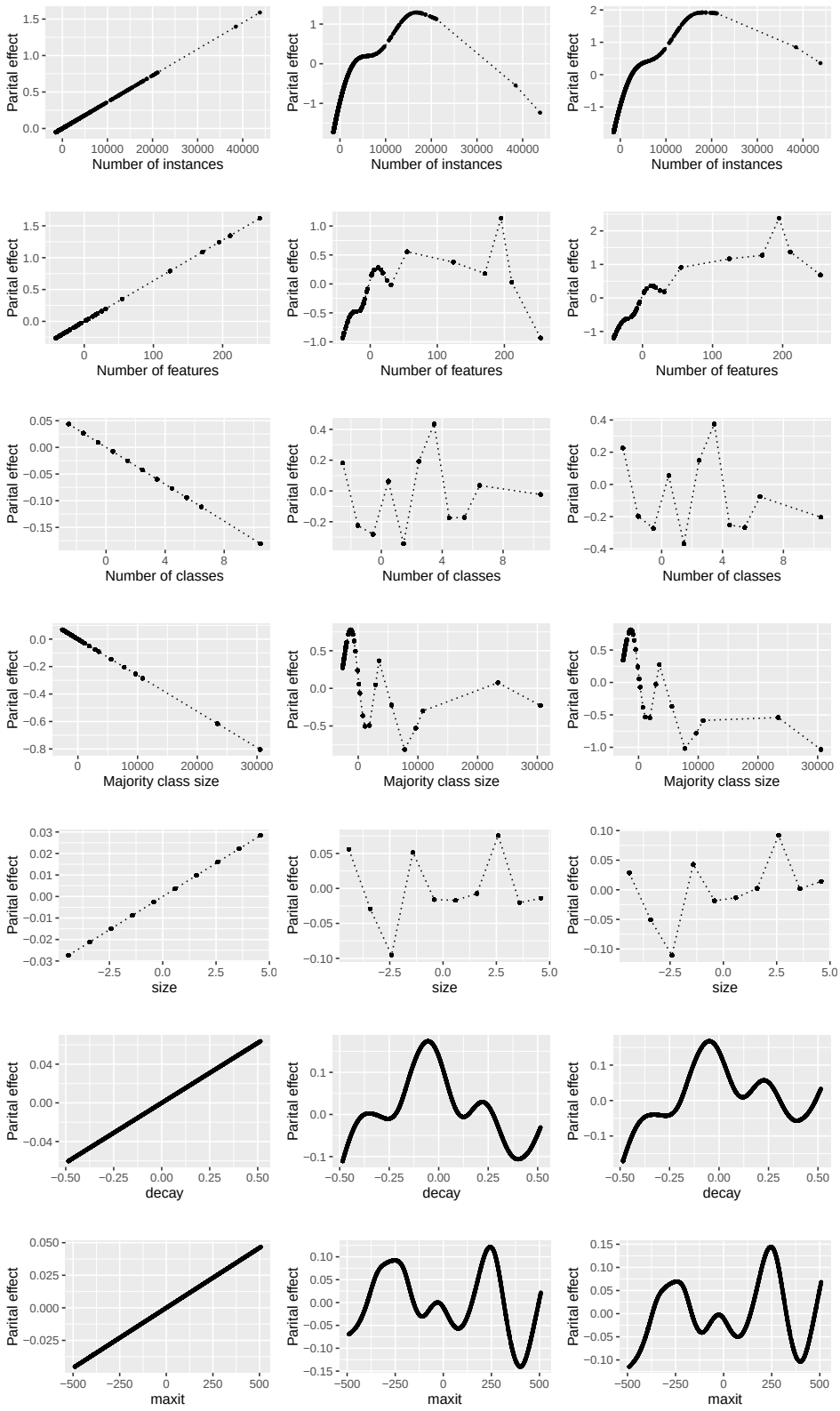


Figure A.22: Linear partial effect (left), smooth partial effect (center) and sum of the linear and smooth partial effect (right) for each covariate of the gamboost model on prediction time for the `nnet` classifier.

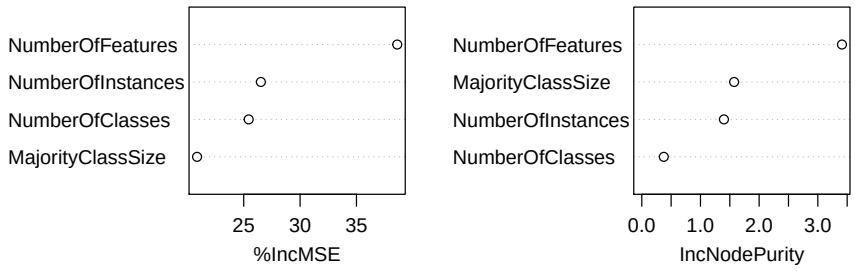


Figure A.23: Variable importance plot of randomForest modelling training time for classifier `naiveBayes`.

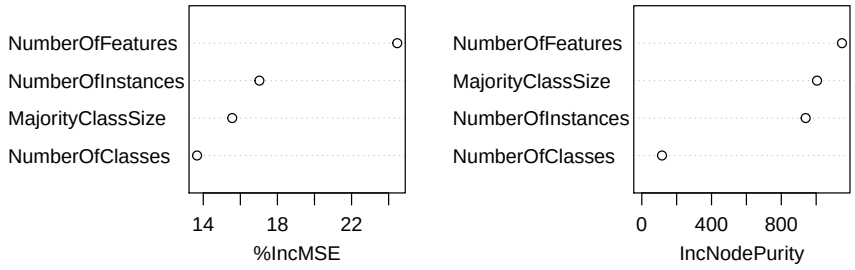


Figure A.24: Variable importance plot of randomForest modelling prediction time for classifier `naiveBayes`.

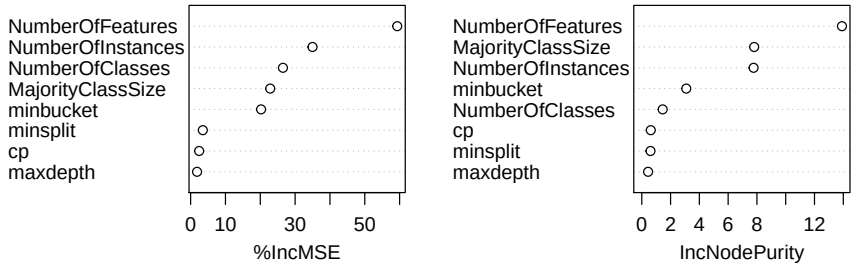


Figure A.25: Variable importance plot of randomForest modelling training time for classifier `rpart`.

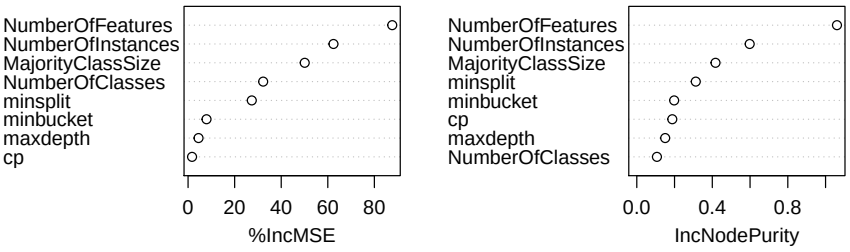


Figure A.26: Variable importance plot of randomForest modelling prediction time for classifier `rpart`.

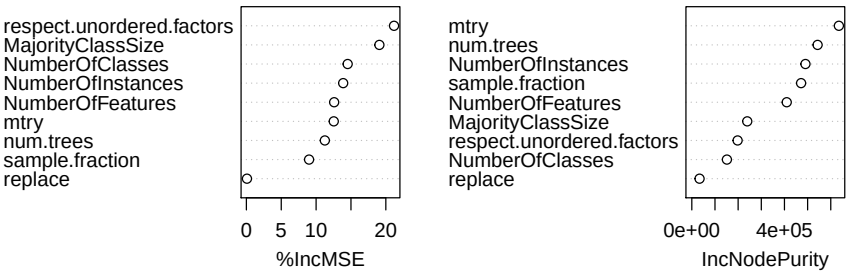


Figure A.27: Variable importance plot of randomForest modelling training time for classifier `ranger`.

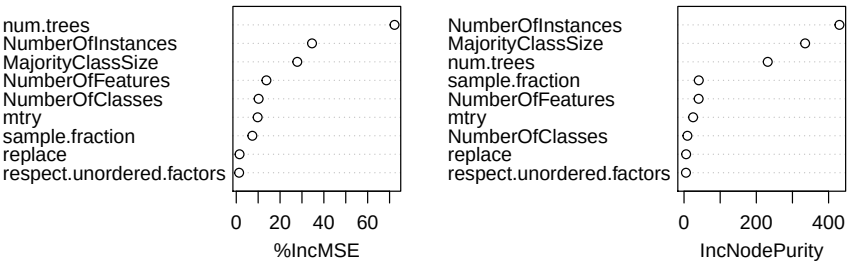


Figure A.28: Variable importance plot of randomForest modelling prediction time for classifier `ranger`.

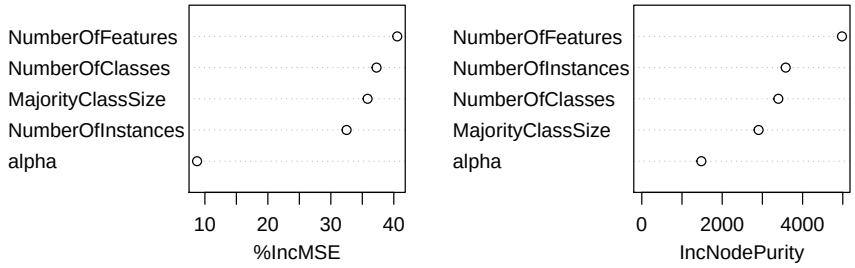


Figure A.29: Variable importance plot of randomForest modelling training time for classifier `glmnet`.

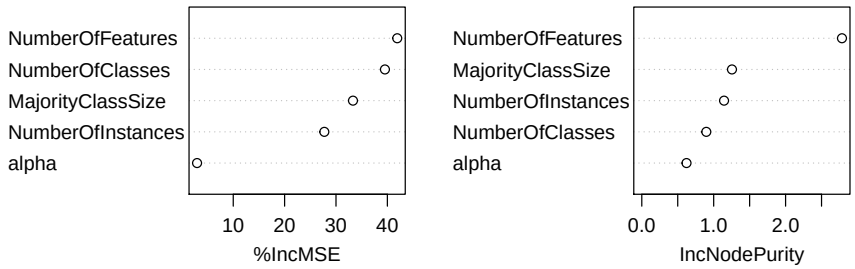


Figure A.30: Variable importance plot of randomForest modelling prediction time for classifier `glmnet`.

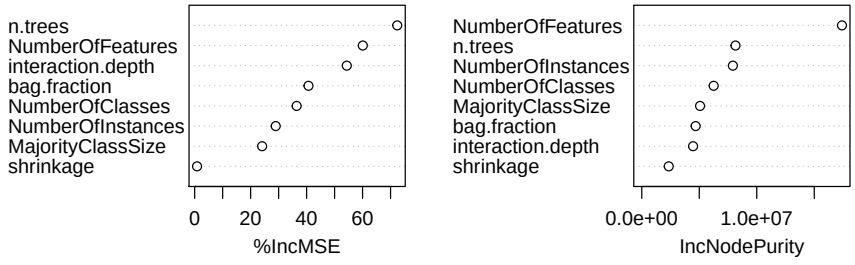


Figure A.31: Variable importance plot of randomForest modelling training time for classifier `gbm`.

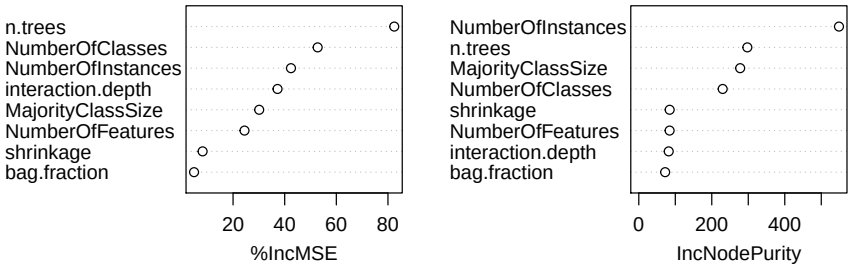


Figure A.32: Variable importance plot of randomForest modelling prediction time for classifier `gbm`.

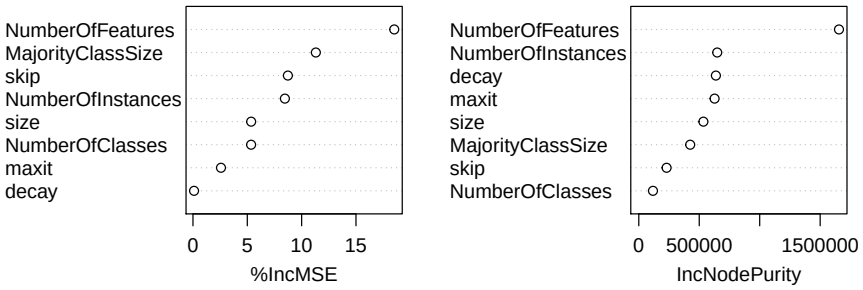


Figure A.33: Variable importance plot of randomForest modelling training time for classifier `nnet`.

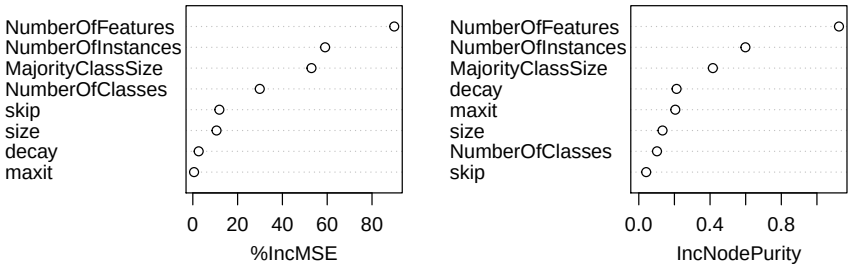


Figure A.34: Variable importance plot of randomForest modelling prediction time for classifier `nnet`.

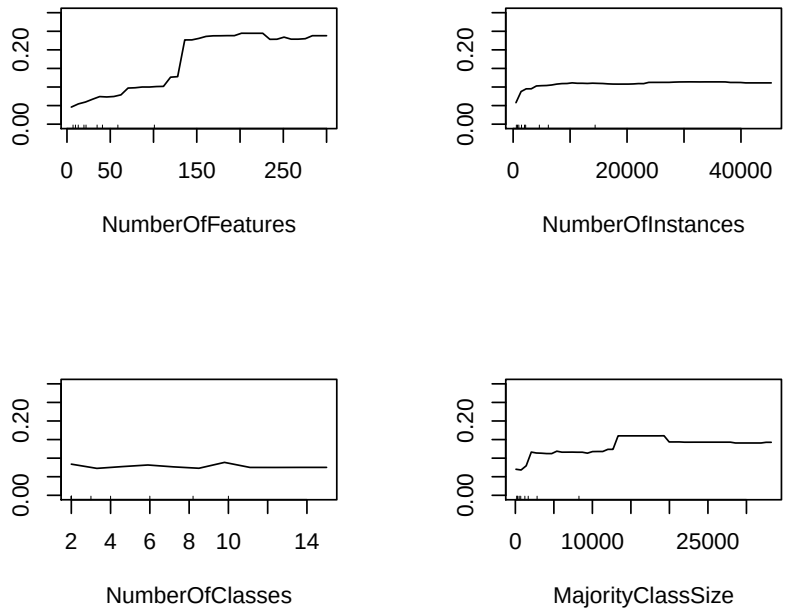


Figure A.35: Partial dependence plot of randomForest modelling training time for classifier `naiveBayes`.

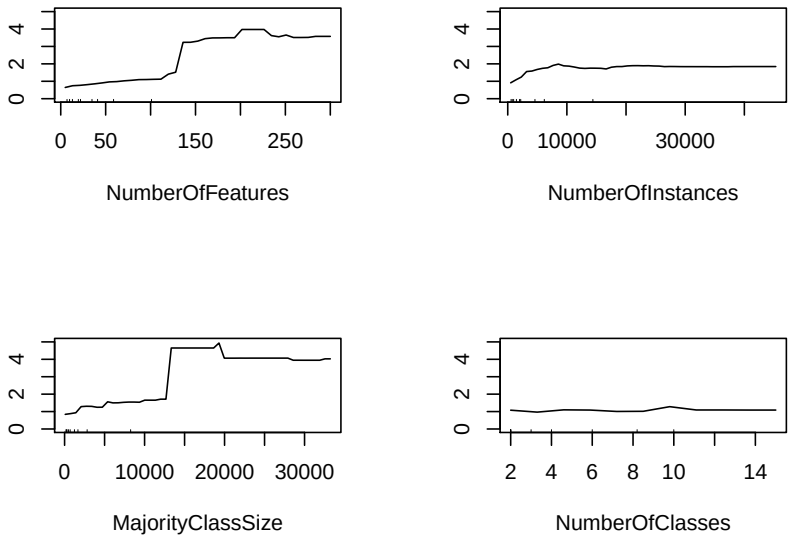


Figure A.36: Partial dependence plot of randomForest prediction time for classifier `naiveBayes`.

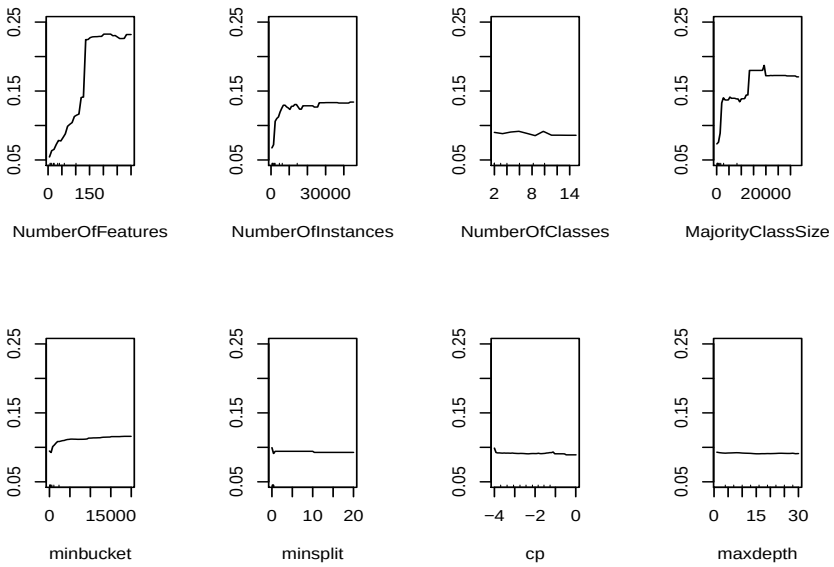


Figure A.37: Partial dependence plot of randomForest modelling training time for classifier `rpart`.

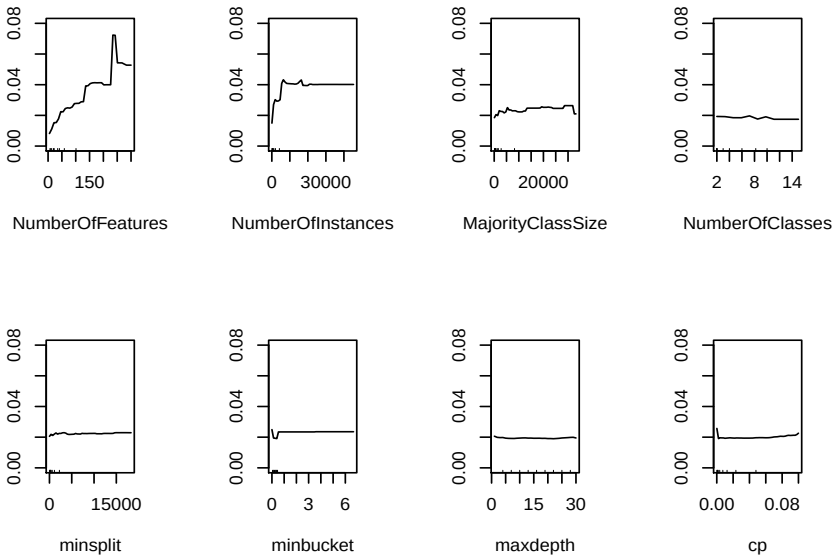


Figure A.38: Partial dependence plot of randomForest modelling prediction time for classifier `rpart`.

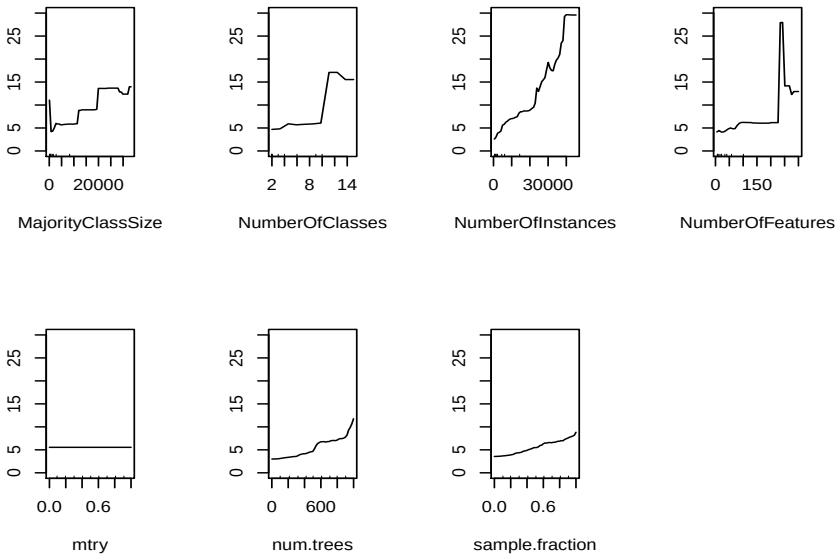


Figure A.39: Partial dependence plot of randomForest modelling training time for classifier `ranger`.

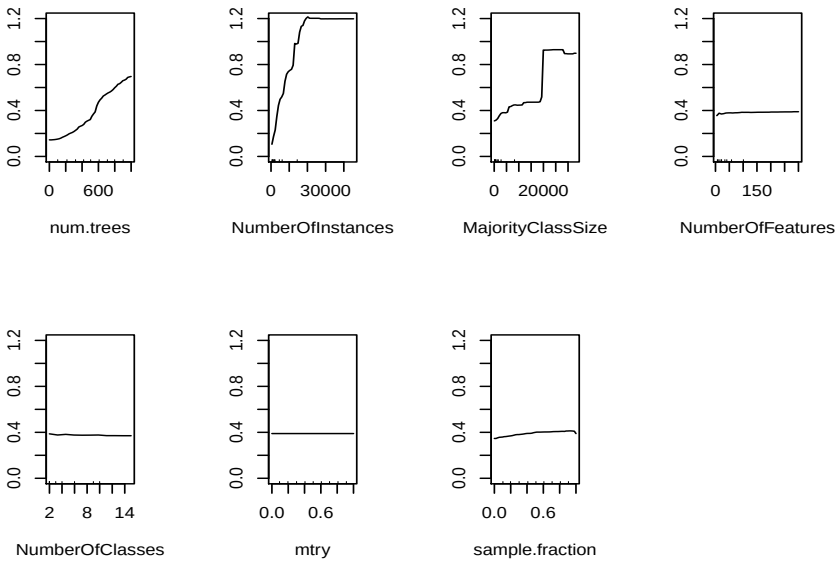


Figure A.40: Partial dependence plot of randomForest modelling prediction time for classifier `ranger`.

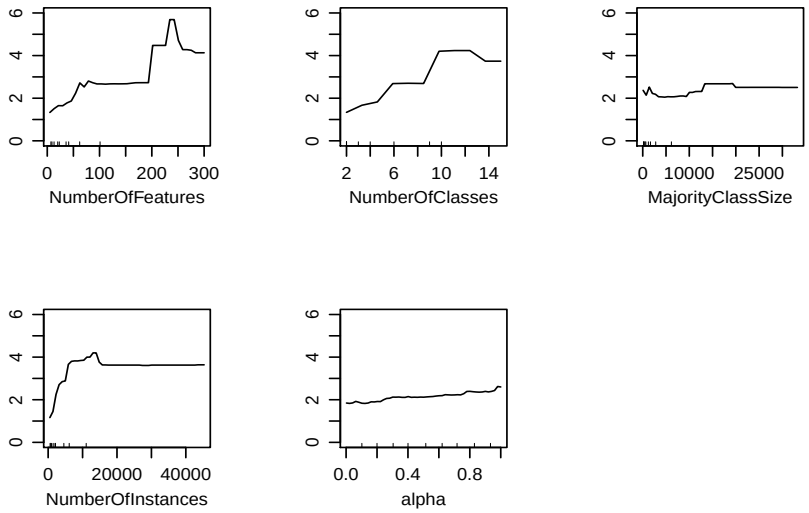


Figure A.41: Partial dependence plot of randomForest modelling training time for classifier `glmnet`.

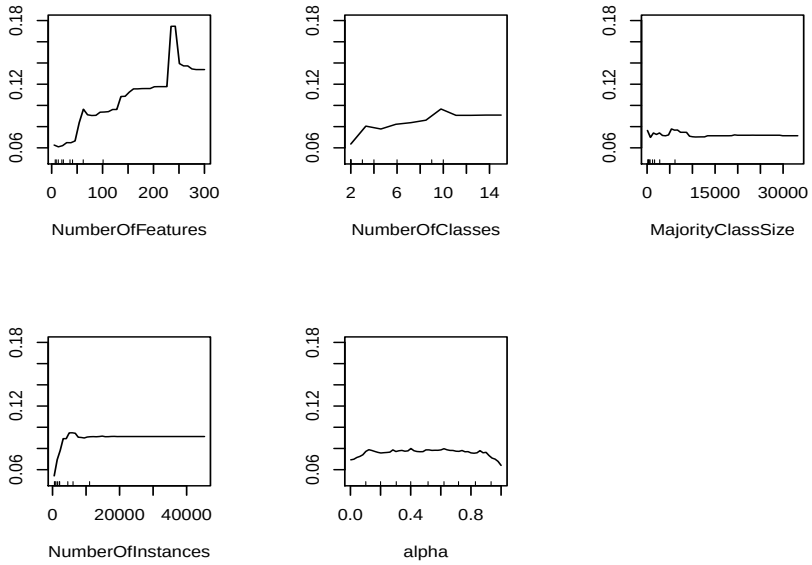


Figure A.42: Partial dependence plot of randomForest modelling prediction time for classifier `glmnet`.

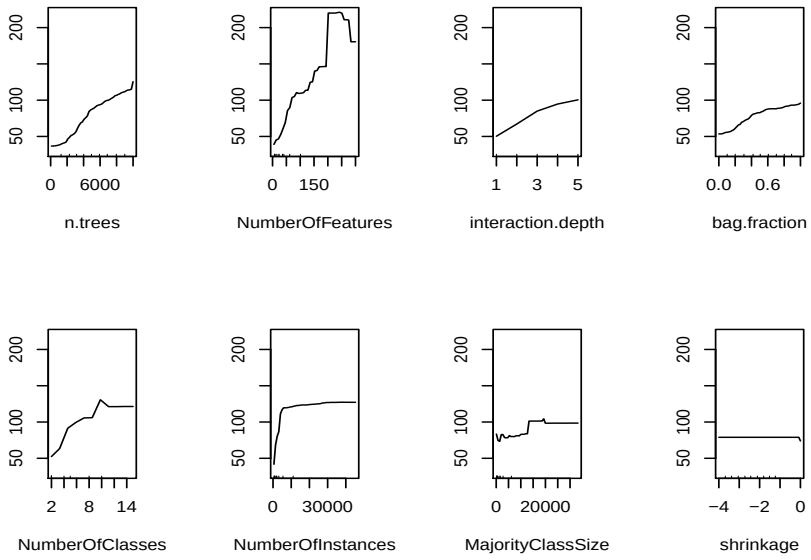


Figure A.43: Partial dependence plot of randomForest modelling training time for classifier `gbm`.

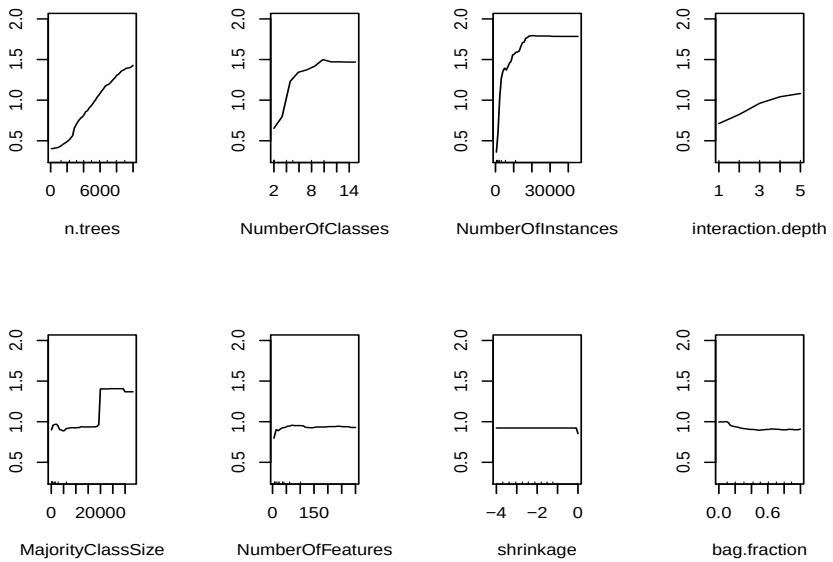


Figure A.44: Partial dependence plot of randomForest modelling prediction time for classifier `gbm`.

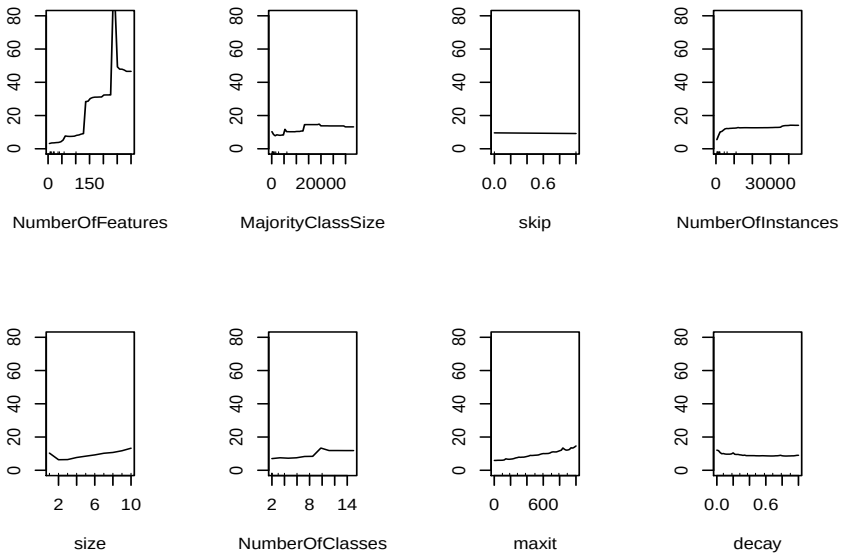


Figure A.45: Partial dependence plot of randomForest modelling training time for classifier `nnet`.

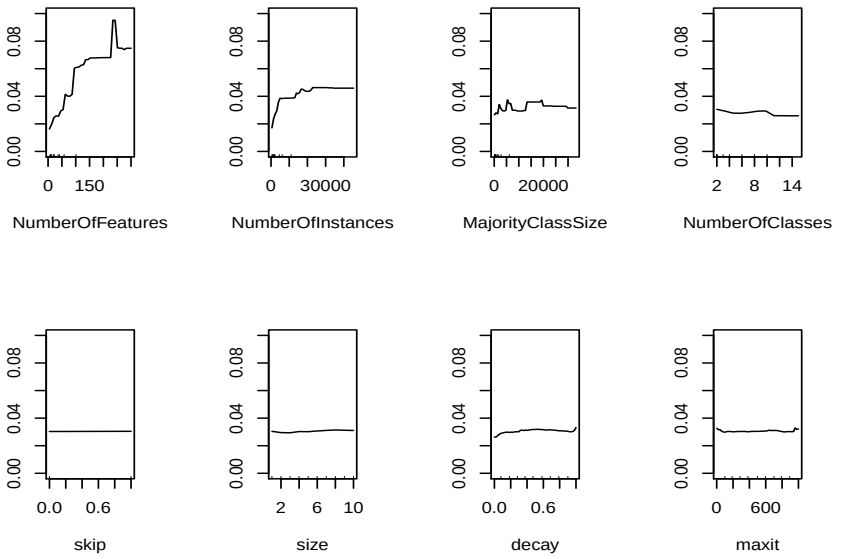


Figure A.46: Partial dependence plot of randomForest modelling prediction time for classifier `nnet`.

B Tables

Table B.1: Collection of the 65 datasets from OpenML. Values are dataset id (did), name of the dataset (name), number of instances (#instances), number of features (#features), number of classes (#classes) and majority class size (Maj.).

did	Name	#Instances	#Features	#classes	Maj.
1554	autoUniv-au7-500	500	13	5	192
478	collins	500	24	15	80
1063	kc2	522	22	2	415
1467	climate-model-simulation-crashes	540	21	2	494
335	monks-problems-3	554	7	2	288
333	monks-problems-1	556	7	2	278
1510	wdbc	569	31	2	357
377	synthetic_control	600	62	6	100
334	monks-problems-2	601	7	2	395
11	balance-scale	625	5	3	288
1553	autoUniv-au7-700	700	13	3	245
1464	blood-transfusion-service-center	748	5	2	570
1549	autoUniv-au6-750	750	41	8	165
37	diabetes	768	9	2	500
469	analcatdata_dmft	797	5	6	155
458	analcatdata_authorship	841	71	4	317
54	vehicle	846	19	4	218
50	tic-tac-toe	958	10	2	626
307	vowel	990	13	11	90
31	credit-g	1000	21	2	700
1547	autoUniv-au1-1000	1000	21	2	741
1555	autoUniv-au6-1000	1000	41	8	240
1494	qsar-biodeg	1055	42	2	699
1552	autoUniv-au7-1100	1100	13	5	305
1068	pc1	1109	22	2	1032
1049	pc4	1458	38	2	1280
23	cmc	1473	10	3	629

did	Name	#Instances	#Features	#classes	Maj.
1050	pc3	1563	38	2	1403
1501	semeion	1593	257	10	162
21	car	1728	7	4	1210
1504	steel-plates-fault	1941	34	2	1268
18	mfeat-morphological	2000	7	10	200
22	mfeat-zernike	2000	48	10	200
16	mfeat-karhunen	2000	65	10	200
14	mfeat-fourier	2000	77	10	200
12	mfeat-factors	2000	217	10	200
20	mfeat-pixel	2000	241	10	200
1067	kc1	2109	22	2	1783
1466	cardiotocography	2126	36	10	579
36	segment	2310	20	7	330
312	scene	2407	300	2	1976
1548	autoUniv-au4-2500	2500	101	3	1173
46	splice	3190	62	3	1655
3	kr-vs-kp	3196	37	2	1669
1043	ada_agnostic	4562	49	2	3430
44	spambase	4601	58	2	2788
1570	wilt	4839	6	2	4578
60	waveform-5000	5000	41	3	1692
1489	phoneme	5404	6	2	3818
1497	wall-robot-navigation	5456	25	4	2205
28	optdigits	5620	65	10	572
1475	first-order-theorem-proving	6118	52	6	2554
182	satimage	6430	37	6	1531
1116	musk	6598	170	2	5581
4538	GesturePhaseSegmentationProcessed	9873	33	5	2950
375	JapaneseVowels	9961	15	9	1614
32	pendigits	10992	17	10	1144
4534	PhishingWebsites	11055	31	2	6157
1036	sylva_agnostic	14395	217	2	13509
1471	eeg-eye-state	14980	15	2	8257
1046	mozilla4	15545	6	2	10437
1120	MagicTelescope	19020	12	2	12332
4135	Amazon_employee_access	32769	10	2	30872
1220	Click_prediction_small	39948	12	2	33220
151	electricity	45312	9	2	26075

Table B.2: Investigated classifiers and their hyperparameters, that were selected randomly, in order to create many different hyperparameter sets. Columns present the R name of the classification algorithm, the name of the hyperparameter, a short description of the hyperparameter (adopted from the manuals of the respective R packages), and the hyperparameter search space. Note, that classifier naiveBayes is not tabulated, since its only hyperparameter `laplace` was set to one for all experiments.

Name	Parameter	Description	Range
rpart	maxdepth	Depth of the final tree (0 = root node).	1 – 30
	minsplit	Minimum number of observations needed in a node in order to split the node.	max(1, ⌈minsplit*sub.sample.frac*n⌉), minsplit ∈ [0, 0.5]
	minbucket	Minimum number of observations in any terminal node.	max(1, ⌈minbucket*sub.sample.frac*n⌉), minbucket ∈ [0, 0.5]
	cp	Complexity parameter. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted.	10^{-4} – 10^{-1}
ranger	num.tress	Number of trees.	1 – 1,000
	mtry	Number of variables to possibly split at in each node.	max(1, ⌈mtry*p⌉), mtry ∈ [0,1]
	sample.fraction	Fraction of observations to sample.	max(sample.fraction, 1/n), sample.fraction ∈ [0,1]
	replace	Whether to sample with replacement.	TRUE/FALSE

⌈ ⌉: floor function, n: Number of instances, p: Number of features

Name	Parameter	Description	Range
glmnet	alpha	Elasticnet mixing parameter.	0 – 1
gbm	n.trees	Number of trees. Equivalent to the number of iterations and number of basis functions in the expansion	500 – 10,000
	interaction.depth	Maximum depth of variable interactions.	1 – 5
	bag.fraction	Fraction of training set observations for the next tree in the expansion.	0 – 1
	shrinkage	Learning rate or step-size reduction. Is applied to each tree in the expansion.	10^{-4} – 10^{-1}
nnet	size	Number of units in the hidden layer.	1 – 10
	decay	Parameter for regularisation.	0.00001 – 1.0
	maxit	Maximum number of iterations.	2 – 1,000
	skip	Whether to add skip-layer connections.	TRUE/FALSE

Table B.3: Number of samples in the meta-datasets used within the evaluation for the different target classifiers. Tabulated are the number of samples when all replications are taken into account and after summarising replications of one experiment. The ten replications were summarised by taking the mean across the ten replications. However, if jobs were erroneous or terminated before finishing, the mean across the remaining non-erroneous replication was taken.

	All replications	After summarising replications
ranger	39037	3842
rpart	33150	3315
gbm	31449	3150
glmnet	13440	1344
naiveBayes	7150	715
nnet	32385	3240

Table B.4: Number and types of errors with the name of the dataset, where the error occurred. The tabulated error message represents an excerpt of the message output by R.

did	name	#cl	p	n	#error	classifier	error message
20	mfeat-pixel	10	241	2000	10	nnet	Too many (4267) weights.
50	tic-tac-toe	2	10	958	10	ranger	User interrupt or internal error.
1116	musk	2	170	6598	320	ranger	Too many levels in unordered categorical variable.
1063	kc2	2	22	522	10	ranger	User interrupt or internal error.
1554	autoUniv-au7-500	5	13	500	10	ranger	User interrupt or internal error.
4135	Amazon_employee	2	10	32769	257	ranger	Too many levels in unordered categorical variable.
	_access				510	gbm	gbm does not handle categorical variables with more than 1024 levels.
					5	ranger	Error in getCacheURI.
					1	ranger	Error in parseHeader. Invalid column specification line found in ARFF reader.

did: dataset id, #: Number of, cl: classes, p: Number of features, n: Number of instances

Table B.5: Number of jobs expiring due to exceeding memory or time limit in total and with respect to the classifiers.

Classifier	Memory	Time
gbm	1027	164
glmnet	210	0
nnet	660	95
Total	1897	259

Table B.6: Minimum (Min.), maximum (Max.), median, mean, 1st and 3rd quartile (1st Qu. and 3rd Qu.) of training time, prediction time, total runtime and mean misclassification error (mmce) with respect to the six classifiers.

Classifier	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Training time						
gbm	0.010	2.696	12.217	63.832	58.815	1033.790
glmnet	0.009	0.181	0.487	2.173	1.932	52.073
naiveBayes	0.002	0.011	0.037	0.080	0.116	0.866
nnet	0.003	0.110	0.639	7.245	3.132	1019.450
ranger	0.004	0.153	0.581	6.759	3.081	733.927
rpart	0.003	0.014	0.040	0.091	0.127	1.054
Prediction time						
gbm	0.001	0.060	0.195	0.472	0.554	7.597
glmnet	0.013	0.018	0.033	0.062	0.055	0.837
naiveBayes	0.011	0.085	0.329	1.125	1.256	26.110
nnet	0.001	0.004	0.007	0.019	0.016	0.293
ranger	0.001	0.018	0.050	0.204	0.148	7.159
rpart	0.001	0.003	0.005	0.019	0.015	0.235
Total runtime						
gbm	0.012	2.786	12.480	64.300	59.430	1035.000
glmnet	0.024	0.211	0.536	2.235	2.001	52.490
naiveBayes	0.015	0.096	0.400	1.205	1.370	26.600
nnet	0.005	0.118	0.656	7.264	3.170	1020.000
ranger	0.005	0.185	0.654	6.964	3.297	734.000
rpart	0.005	0.018	0.083	0.110	0.158	1.131
mmce						
gbm	0.000	0.047	0.135	0.197	0.264	0.909
glmnet	0.000	0.054	0.143	0.210	0.268	0.888
naiveBayes	0.000	0.134	0.247	0.299	0.406	0.937
nnet	0.000	0.062	0.158	0.242	0.344	0.988
ranger	0.000	0.044	0.104	0.176	0.233	0.879
rpart	0.000	0.180	0.321	0.383	0.584	0.900

Table B.7: Minimum (Min.), maximum (Max.), median, mean, 1st and 3rd quartile (1st Qu. and 3rd Qu.) of variability, assessed by the coefficient of variation, for training time, prediction time and the mean misclassification error (mmce) with respect to the six classifiers.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Training time						
gbm	0.128	1.857	2.730	3.065	3.664	58.050
glmnet	0.895	3.705	5.569	12.910	11.370	248.000
naiveBayes	0.000	4.682	6.295	6.770	8.474	21.520
nnet	0.678	9.140	16.560	29.550	32.110	277.500
ranger	0.000	2.578	3.594	5.035	5.118	103.500
rpart	0.000	4.947	6.308	14.400	8.076	264.900
Prediction time						
gbm	0.000	1.947	2.975	3.405	4.028	76.840
glmnet	1.133	3.386	4.438	4.904	5.635	65.330
naiveBayes	0.958	2.637	3.381	3.700	4.479	18.180
nnet	0.000	5.360	8.108	8.886	11.230	111.600
ranger	0.000	4.129	5.458	7.953	7.458	123.400
rpart	0.000	6.908	10.310	11.100	14.640	106.700
mmce						
gbm	0.000	3.689	7.523	15.550	15.310	316.200
glmnet	0.000	3.548	6.383	14.920	12.210	316.200
naiveBayes	0.714	3.769	6.562	9.825	10.470	108.800
nnet	0.000	5.246	11.730	22.650	26.810	316.200
ranger	0.000	4.129	5.458	7.953	7.458	123.400
rpart	0.000	0.000	2.038	4.881	5.465	316.200

Table B.8: Estimated coefficients for each modelling alternative of the covariates for the glmboost and glm model with respect to target classifier `naiveBayes`. Estimates for responses training time and prediction time are presented.

	Training time		Prediction time	
	glmboost	glm	glmboost	glm
(Intercept)	−11.539	−18.604 ***	−8.606	−10.616 ***
NumberOfInstances	0.000	0.000 **		0.000 *
NumberOfInstances^2	0.000			0.000 *
sqrt(NumberOfInstances)	−0.047	−0.041 ***	0.006	−0.014 *
log(NumberOfInstances)	1.064	1.017 ***	0.748	0.822 ***
NumberOfFeatures	−0.083	−0.088 ***	−0.002	−0.028 ***
NumberOfFeatures^2	0.000	0.000 ***		0.000 ***
sqrt(NumberOfFeatures)	1.685	1.793 ***	0.064	0.617 ***
log(NumberOfFeatures)	−1.242	−1.373 **	0.657	
NumberOfClasses	0.483	−0.886 **		0.480 ***
NumberOfClasses^2	−0.017			−0.014 ***
sqrt(NumberOfClasses)	0.211	9.173 **		−0.984 *
log(NumberOfClasses)	−1.302	−5.249 **		
MajorityClassSize	0.000	0.000 **		0.000 **
MajorityClassSize^2	0.000	0.000 .		0.000 **
sqrt(MajorityClassSize)	0.075	0.073 ***		−0.019 *
log(MajorityClassSize)	−0.401	−0.398 *		0.394 ***

Signif. codes: *** < 0.001, **: < 0.01, *: < 0.05, .: < 0.1, ' ': > 0.1

Table B.9: Estimated coefficients for each modelling alternative of the covariates for the glmboost and glm model with respect to target classifier `rpart`. Estimates for responses training time and prediction time are presented.

	Training time		Prediction time	
	glmboost	glm	glmboost	glm
(Intercept)	−5.918	−6.076 ***	−11.645	−52.330 ***
NumberOfInstances	0.000	0.000 ***	0.000	0.000 ***
NumberOfInstances^2	0.000	0.000 ***	0.000	
sqrt(NumberOfInstances)	0.043	0.057 ***	0.041	0.054 ***
log(NumberOfInstances)	0.124		0.110	
NumberOfFeatures	−0.070	−0.075 ***	0.040	0.075 ***
NumberOfFeatures^2	0.000	0.000 ***	0.000	0.000 ***
sqrt(NumberOfFeatures)	1.646	1.748 ***	−0.784	−1.421 ***
log(NumberOfFeatures)	−1.785	−1.905 ***	1.671	2.356 ***
NumberOfClasses	0.150		−0.204	−8.010 **
NumberOfClasses^2	−0.004		0.008	0.110 **
sqrt(NumberOfClasses)	0.022	0.649 ***	−0.004	49.561 **
log(NumberOfClasses)	−0.416	−0.563 ***	0.287	−20.868 .
MajorityClassSize	0.000	0.000 **	0.000	0.001 ***
MajorityClassSize^2	0.000	0.000 *	0.000	0.000 *
sqrt(MajorityClassSize)	0.026	0.018 ***	−0.074	−0.108 ***
log(MajorityClassSize)	−0.062		0.623	0.942 ***
maxdepth	0.016	0.008		
maxdepth^2	−0.001	0.000 *	0.000	
sqrt(maxdepth)			0.036	0.024 0
log(maxdepth)	−0.037		0.016	
minsplit	0.000	0.000	0.000	
minsplit^2	0.000		0.000	0.000 .
sqrt(minsplit)	0.003	0.004 .	−0.010	−0.011 **
log(minsplit)	0.013		0.118	0.134 ***
minbucket	0.000	0.000 *	0.000	
minbucket^2	0.000		0.000	
sqrt(minbucket)	−0.011	−0.013 **	−0.005	−0.005
log(minbucket)	0.060	0.071 **	0.063	0.066 *
cp	35.923	37.641 **	−9.008	
cp^2	−174.398	−182.115 **	20.877	
sqrt(cp)	−8.242	−8.681 **	2.736	
log(cp)	0.075	0.080 *	−0.030	

Signif. codes: *** < 0.001, **: < 0.01, *: < 0.05, .: < 0.1, ' ': > 0.1

Table B.10: Estimated coefficients for each modelling alternative of the covariates for the glmboost and glm model with respect to target classifier `ranger`. Estimates for responses training time and prediction time are presented. The results for the glm model on training time are model results without variable selection and need to be interpreted considering that the algorithm did not converge. The results for glm on prediction time also need to be interpreted considering the algorithm did not converge during variable selection.

	Training time		Prediction time	
	glmboost	glm	glmboost	glm
(Intercept)	−8.711	88.164 ***	−11.000	10.674 .
NumberOfInstances	0.000	0.001 ***	0.000	0.000 ***
NumberOfInstances^2	0.000	0.000 ***	0.000	0.000 ***
sqrt(NumberOfInstances)	−0.015	−0.237 ***	0.003	
log(NumberOfInstances)	1.564	3.193 ***	0.626	0.630 ***
NumberOfFeatures	−0.009	0.041 *	−0.044	−0.057 ***
NumberOfFeatures^2	0.000	0.000	0.000	0.000 ***
sqrt(NumberOfFeatures)	−0.079	−1.010 **	0.783	0.999 ***
log(NumberOfFeatures)	0.289	1.352 **	−0.753	−0.971 ***
NumberOfClasses	−0.154	19.088 ***	0.089	4.272 ***
NumberOfClasses^2	0.013	0.228 ***	−0.006	−0.060 ***
sqrt(NumberOfClasses)		−125.603 ***	−26.677	***
log(NumberOfClasses)	0.489	55.916 ***	0.237	11.692 ***
MajorityClassSize	−0.001	−0.002 ***	0.000	0.000 ***
MajorityClassSize^2	0.000	0.000 ***	0.000	0.000 ***
sqrt(MajorityClassSize)	0.233	0.388 ***	0.019	0.030 ***
log(MajorityClassSize)	−2.548	−3.322 ***	0.094	
num.trees	0.001	−0.003	0.001	
num.trees^2	0.000	0.000	0.000	0.000 ***
sqrt(num.trees)	0.053	0.184	0.100	0.127 ***
log(num.trees)	0.571	0.306	−0.062	−0.113 **
sample.fraction		2.777		−0.771 ***
sample.fraction^2	−0.865	0.086	−0.221	
sqrt(sample.fraction)	3.778	6.670 *	0.503	1.431 ***
log(sample.fraction)	0.009	−0.177	0.064	
mtry	0.003	−0.006	0.017	0.023 ***
mtry^2	0.000	0.000	0.000	0.000 ***
sqrt(mtry)	0.069	0.253	−0.261	−0.334 ***
log(mtry)	0.366	0.173	0.186	0.240 ***
replaceTRUE	−0.100	−0.091 *	−0.008	
respect.unordered.-factorsTRUE	0.707	0.558 ***	0.038	0.038 *

Signif. codes: *** < 0.001, **: < 0.01, *: < 0.05, .: < 0.1, ' ': > 0.1

Table B.11: Estimated coefficients for each modelling alternative of the covariates for the glmboost and glm model with respect to target classifier `glmnet`. Estimates for responses training time and prediction time are presented. The results for the glm model on training time need to be interpreted considering that the algorithm for the model and the variable selection did not converge.

	Training time		Prediction time	
	glmboost	glm	glmboost	glm
(Intercept)	−12.895	166.700 ***	0.698	74.889 ***
NumberOfInstances	0.000	0.000 **	0.000	0.000 ***
NumberOfInstances^2	0.000		0.000	
sqrt(NumberOfInstances)	−0.110	−0.035 ***	0.072	0.076 ***
log(NumberOfInstances)	2.524	1.394 ***	−0.468	−0.560 ***
NumberOfFeatures	0.059	0.075 ***	0.077	0.071 ***
NumberOfFeatures^2	0.000	0.000 **	0.000	0.000 ***
sqrt(NumberOfFeatures)	−1.079	−1.655 ***	−0.722	−0.653 ***
log(NumberOfFeatures)	1.742	2.754 ***	0.050	
NumberOfClasses	−0.196	33.702 ***	−0.457	13.871 ***
NumberOfClasses^2	0.028	−0.416 ***	0.026	−0.159 ***
sqrt(NumberOfClasses)	−6.214	−221.663 ***	−2.117	−93.450 ***
log(NumberOfClasses)	7.177	99.599 ***	3.334	42.486 ***
MajorityClassSize	0.000	0.000 ***	0.000	0.000 ***
MajorityClassSize^2	0.000		0.000	0.000 ***
sqrt(MajorityClassSize)	0.008	0.033 ***	−0.029	
log(MajorityClassSize)	−0.210	−0.262 *	0.402	0.204 **
alpha	0.996	1.042 ***	1.964	
alpha^2	−0.694	−0.635 **	−0.611	
sqrt(alpha)	0.535		−3.064	−0.742 .
log(alpha)	−0.096		0.350	0.188 .

Signif. codes: *** < 0.001, **: < 0.01, *: < 0.05, .: < 0.1, ' ': > 0.1

Table B.12: Estimated coefficients for each modelling alternative of the covariates for the glmboost and glm model with respect to target classifier `gbm`. Estimates for responses training time and prediction time are presented. The results for glm on prediction time need to b interpreted considering that the algorithm did not converge during variable selection.

	Training time		Prediction time	
	glmboost	glm	glmboost	glm
(Intercept)	−15.413	19.599 ***	−13.248	19.108 ***
NumberOfInstances		0.000 ***		0.000 ***
NumberOfInstances^2		0.000 ***		
sqrt(NumberOfInstances)			0.003	0.018 ***
log(NumberOfInstances)	1.087	0.927 ***	0.893	0.746 ***
NumberOfFeatures		−0.036 ***		−0.030 ***
NumberOfFeatures^2		0.000 ***		0.000 ***
sqrt(NumberOfFeatures)	0.099	0.786 ***		0.552 ***
log(NumberOfFeatures)	0.448	−0.265 **		−0.631 ***
NumberOfClasses		7.244 ***		6.281 ***
NumberOfClasses^2	−0.006	−0.079 ***		−0.058 ***
sqrt(NumberOfClasses)		−51.568 ***		−47.114 ***
log(NumberOfClasses)	1.627	26.113 ***	1.186	24.647 ***
MajorityClassSize		0.000 **		0.000 ***
MajorityClassSize^2		0.000	0.000	0.000 ***
sqrt(MajorityClassSize)		0.013 **	−0.006	
log(MajorityClassSize)		−0.103 **		
n.trees		0.000 .	0.000	0.000 **
n.trees^2		0.000		0.000 **
sqrt(n.trees)	0.020	−0.023 *	0.017	−0.066 ***
log(n.trees)	0.440	1.200 ***	0.324	1.446 ***
interaction.depth		−0.303 ***		
interaction.depth^2				
sqrt(interaction.depth)	0.323	1.884 ***	0.000	
log(interaction.depth)	0.666		0.629	0.457 ***
bag.fraction		−2.907 ***		−2.990 ***
bag.fraction^2				1.110 ***
sqrt(bag.fraction)	0.621	5.652 ***		2.292 ***
log(bag.fraction)	0.222	−0.163 ***		
shrinkage		1.434		
shrinkage^2		−14.062		
sqrt(shrinkage)				
log(shrinkage)		−0.011 *		

Signif. codes: *** < 0.001, **: < 0.01, *: < 0.05, .: < 0.1, ' ': > 0.1

Table B.13: Estimated coefficients for each modelling alternative of the covariates for the glmboost and glm model with respect to target classifier `nnet`. Estimates for responses training time and prediction time are presented.

	Training time		Prediction time	
	glmboost	glm	glmboost	glm
(Intercept)	−21.657	49.786 ***	−10.594	−33.784 *
NumberOfInstances	0.001	0.001 ***	0.000	0.000 ***
NumberOfInstances^2	0.000	0.000 ***	0.000	0.000 ***
sqrt(NumberOfInstances)	−0.125	−0.138 ***	−0.006	
log(NumberOfInstances)	2.729	2.701 ***	0.382	0.347 ***
NumberOfFeatures	0.050	0.071 ***	0.063	0.079 ***
NumberOfFeatures^2	0.000	0.000 ***	0.000	0.000 ***
sqrt(NumberOfFeatures)	−0.297	−0.783 *	1.056	−1.328 ***
log(NumberOfFeatures)	0.488	1.136 **	1.739	2.021 ***
NumberOfClasses	−1.005	12.889 ***	0.257	−4.212 .
NumberOfClasses^2	0.043	−0.137 ***	−0.008	0.048
sqrt(NumberOfClasses)	−0.440	−89.178 ***	0.032	28.961 .
log(NumberOfClasses)	3.535	41.725 ***	−1.166	−13.749 *
MajorityClassSize	0.000	0.000 ***	0.000	0.000 ***
MajorityClassSize^2	0.000		0.000	
sqrt(MajorityClassSize)	−0.004	0.032 ***	−0.056	−0.055 ***
log(MajorityClassSize)	−0.127	−0.362 ***	0.487	0.487 ***
size	0.420		0.137	0.012
size^2	−0.018	−0.010 *	−0.008	−0.001
sqrt(size)	0.220	2.437 ***	0.010	0.022 .
log(size)	−0.341	−1.112 *	−0.245	
decay	−2.656	−2.591 **	−1.105	−1.101 ***
decay^2	0.921	0.915 .	0.023	
sqrt(decay)	1.956	1.873 **	1.380	1.412 ***
maxit	−0.001	−0.006 ***		−0.001
maxit^2	0.000	0.000 *	0.000	
sqrt(maxit)	0.050	0.238 ***	0.002	
log(maxit)	0.361		0.043	
skipTRUE	−0.038			−0.079 *

Signif. codes: *** < 0.001, **: < 0.01, *: < 0.05, .: < 0.1, ' ': > 0.1

6 Structure of the repository

The empirical study of this thesis was implemented by using R. All associated R scripts are available from:

https://github.com/MariaErdmann/Bachelor-Thesis-Runtime-Prediction.

A description of this repository on GitHub is given below:

1. MariaErdmann/Bachelor-Thesis-Runtime-Prediction/: contains all scripts and objects for generating the meta-dataset
    - **datasets.R**: creates **class.dsets.RData**, a `data.frame` object with the dataset characteristics of the 65 datasets.
    - **definitions.R**: script for the definition of the learner's hyperparameter set.
    - **experiments.R**: batchtools set up. Creates registries for each dataset and jobs for each learner with a random hyperparameter set.
    - **.batchtools_conf.R**: batchtools configuration file for submitting jobs on the lrz cluster.
    - **lmu_lrz_new.tmpl**: template for submitting jobs on the lrz cluster.
    - **checkSingleResults.R**: checks and collects the results of the registries individually.
    - **createResultDataframe.R**: checks and collects the results of all registries in Results/finalresults.RData.
    - **createExceededData.R**: creates **only_expired_jobs.RData**, a `data.frame` object with jobs that were terminated due to exceeding time or memory limits.
    - **createDataframesForPred.R**: creates meta-datasets for each classifier.

2. MariaErdmann/Bachelor-Thesis-Runtime-Prediction/Results/: contains all scripts and objects for modelling runtime.
    - SingleResults/: contains results from the individual registries, which are `data.frame`s produced by **checkSingleResults.R**.
    - ErrorsForEachTask/: contains erroneous results from the individual registries, which are `data.frame`s produced by **checkSingleResults.R**.
    - DatasetForAnalysis/: contains all meta-datasets for each classifier necessary for analysis. These are produced by **createDataframesForPred.R**.
    - AnalysisOnServer/: contains all scripts for runtime prediction with the gradient boosting models. Since search for the optimal m_{stop} requires a high computational effort, these scripts were run on the server of the statistical department. To run scripts locally in order to test them, the `grid` argument of the `cvrisk` function should be changed to `1:500`.
    - ServerResults/: contains objects of class `cvrisk` of the glmboost and gamboost models, which is a matrix containing estimates of the empirical risk for a varying number of bootstrap iterations. Furthermore, some objects of the class `glmboost` and `gamboost` are uploaded. Model-objects greater than 25 MB cannot be uploaded, which applies for the bigger part of the objects.
    - **formulas.R**: creates the formulas used for modelling with `glmboost`, `gamboost` and `glm`.
    - **helperFunctions.R**: contains several helper functions for creating condensed meta-datasets, creating summary statistics, creating model formulas, generating specific ggplots, calculating RAE and RMSE.
    - **gbmAnalysis.R**, **glmnetAnalysis.R**, **naiveBayesAnalysis.R**, **nnetAnalysis.R**, **rangerAnalysis.R**, **rpartAnalysis.R**: scripts for the analysis of the regression models separately for each classifier.

Bibliography

Al-Jarrah, O. Y., Yoo, P. D., Muhaidat, S., et al. (2015). Efficient machine learning for big data: A review. *Big Data Research*, 2(3):87–93.

Bischl, B. and Bossek, J. (NA). *farff: A faster ARFF file reader and writer*. R package version 1.0.

Bischl, B., Lang, M., Kotthoff, L., et al. (NA). *mlr: Machine Learning in R*. R package version 2.9. https://github.com/mlr-org/mlr.

Bischl, B., Lang, M., Mersmann, O., et al. (2015). Batchjobs and batchexperiments: Abstraction mechanisms for using r in batch environments. *Journal of Statistical Software*, 64(11).

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

Breiman, L., Friedman, J., and Stone, C. J. (1998). *Classification and regression trees*. Chapman & Hall, Boca Raton, repr edition.

Brownlee, J. (2014). Penalized regression in R. Retrieved 18 Aug. 2016 from http://machinelearningmastery.com/penalized-regression-in-r/.

Buehlmann, P. and Hothorn, T. (2007). Boosting algorithms: Regularization, prediction and model fitting (with discussion). *Statistical Science*, 22(4):477–505.

Casalicchio, G., Bischl, B., Kirchhoff, D., et al. (NA). *OpenML: Exploring Machine Learning Better, Together*. R package version 1.0. https://github.com/openml/openml-r.

Chávez, E., Navarro, G., Baeza-Yates, R., et al. (2001). Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321.

Doan, T. and Kalita, J. (2016). Predicting run time of classification algorithms using meta-learning. *International Journal of Machine Learning and Cybernetics*.

Eilers, P. H. C. and Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. *Statistical Science*, 11:89–121.

Elkan, C. (1997). Naive bayesian learning. Retrieved 22 June 2016 from http://www4.cs.umanitoba.ca/~jacky/Teaching/Courses/COMP_4360-MachineLearning/Additional_Information/elkan-naive-bayesian-learning.pdf.

Fahrmeir, L., Kneib, T., and Lang, S. (2009). *Regression*. Springer Berlin Heidelberg, Berlin, Heidelberg.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Friedman, J., Hastie, T., and Tibshirani, R. (2010a). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22.

Friedman, J., Hastie, T., and Tibshirani, R. (2010b). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22.

Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232.

Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, 29(2/3):131–163.

Hastie, T. J., Tibshirani, R. J., and Friedman, J. H. (2016). *The elements of statistical learning: Data mining, inference, and prediction*. Springer series in statistics. Springer, New York, NY, second edition, corrected at 11th printing edition.

Hofner, B., Hothorn, T., Kneib, T., et al. (2011). A framework for unbiased model selection based on boosting. *Journal of Computational and Graphical Statistics*, 20(4):956–971.

Hofner, B., Mayr, A., and Robinzonov, N. (2014). Model-based boosting in R: A hands-on tutorial using the R package mboost. *Computational Statistics*, 29:3–35.

Hothorn, T., Buehlmann, P., Kneib, T., et al. (2010). Model-based boosting 2.0. *Journal of Machine Learning Research*, 11:2109–2113.

Hothorn, T., Buehlmann, P., Kneib, T., et al. (2016). *mboost: Model-Based Boosting*. R package version R package version 2.6-0. http://CRAN.R-project.org/package=mboost.

Hutter, F., Xu, L., Hoos, H. H., et al. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111.

Hyndman, R. J. and Athanasopoulos, G. (April 2014). *Forecasting: Principles and practice*. OTexts, S.l., print edition edition.

James, G., Witten, D., Hastie, T., et al. (2015). *An introduction to statistical learning: With applications in R*. Springer texts in statistics. Springer, New York, NY, corr. at 6. printing edition.

Kneib, T., Hothorn, T., and Tutz, G. (2009). Variable selection and model choice in geoadditive regression models. *Biometrics*, 65(2):626–634.

Kohn, W. (2005). *Statistik: Datenanalyse und Wahrscheinlichkeitsrechnung*. Statistik und ihre Anwendungen. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg.

Kuhn, M. et al. (2016). *caret: Classification and Regression Training*. R package version 6.0-71. https://CRAN.R-project.org/package=caret.

Lang, M. and Surmann, D. (NA). *batchtools: Tools for Computation on Batch Systems*. R package version 0.1. https://github.com/mllg/batchtools.

Leibniz-Rechenzentrum (18.05.2015). Job processing on the lrz clusters. Retrieved 9 Aug. 2016 from http://www.lrz.de/services/compute/linux-cluster/batch_serial/limits/.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R News*, 2(3):18–22.

Louppe, G. (2014). *Understanding random forests. From theory to pratice.* Phd dissertation, University of Liège, Liège.

Mayr, A., Hofner, B., and Schmid, M. (2012). The importance of knowing when to stop. a sequential stopping rule for component-wise gradient boosting. *Methods of information in medicine*, 51(2):178–186.

Meyer, D., Dimitriadou, E., Hornik, K., et al. (2015). *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.6-7. https://CRAN.R-project.org/package=e1071.

Ng, A., Ngiam, J., Foo, C. Y., et al. (NA). Multi-layer neural network. Retreived 20 Aug. 2016 from http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/.

Priya, R., de Souza, B. F., Rossi, A. L. D., et al. (2011). Predicting execution time of machine learning tasks using metalearning. In *2011 World Congress on Information and Communication Technologies (WICT)*, pages 1193–1198.

Priya, R., de Souza, B. F., Rossi, A. L. D., et al. (2012). Using genetic algorithms to improve prediction of execution times of ml tasks. In Hutchison, D., Kanade, T., Kittler, J., et al., editors, *Hybrid Artificial Intelligent Systems*, volume 7208 of *Lecture Notes in Computer Science*, pages 196–207. Springer Berlin Heidelberg, Berlin, Heidelberg.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. Vienna, Austria. https://www.R-project.org/.

Reif, M., Shafait, F., and Dengel, A. (2011). Prediction of classifier training time including parameter optimization. In Hutchison, D., Kanade, T., and Kittler, J., editors, *KI 2011: Advances in Artificial Intelligence*, volume 7006 of *Lecture Notes in Computer Science*, pages 260–271. Springer Berlin Heidelberg, Berlin, Heidelberg.

Ridgeway, G. (2007). *Generalized Boosted Models: A guide to the gbm package*.

Ridgeway, G. (2015). *gbm: Generalized Boosted Regression Models*. R package version 2.1.1. https://CRAN.R-project.org/package=gbm.

Saed, S. (2016). Model evaluation. Retrieved 26 Aug. 2016 from http://www.saedsayad.com/model_evaluation.htm.

Sunil, R. (13.07.2015). 6 easy steps to learn naive bayes algorithm (with code in python). Retrieved on 19 Aug. 2016 from https://www.analyticsvidhya.com/blog/2015/09/naive-bayes-explained/.

Therneau, T., Atkinson, B., and Ripley, B. (2015). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-10. https://CRAN.R-project.org/package=rpart.

Tutz, G. (2012). *Regression for categorical data*. Cambridge series in statistical and probabilistic mathematics. Cambridge University Press, Cambridge.

Vanschoren, J., van Rijn, J. N., Bischl, B., et al. (2014). OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60.

W. N. Venables and B. D. Ripley (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition.

Wright, M. N. (2016). *ranger: A Fast Implementation of Random Forests*. R package version 0.4.0. https://CRAN.R-project.org/package=ranger.

Zou, K. H., Liu, A., and Bandos, A. I. (2012). *Statistical evaluation of diagnostic performance: Topis in ROC analysis*. Chapman & Hall/CRC biostatistics series. CRC Press, Boca Raton, Fla.