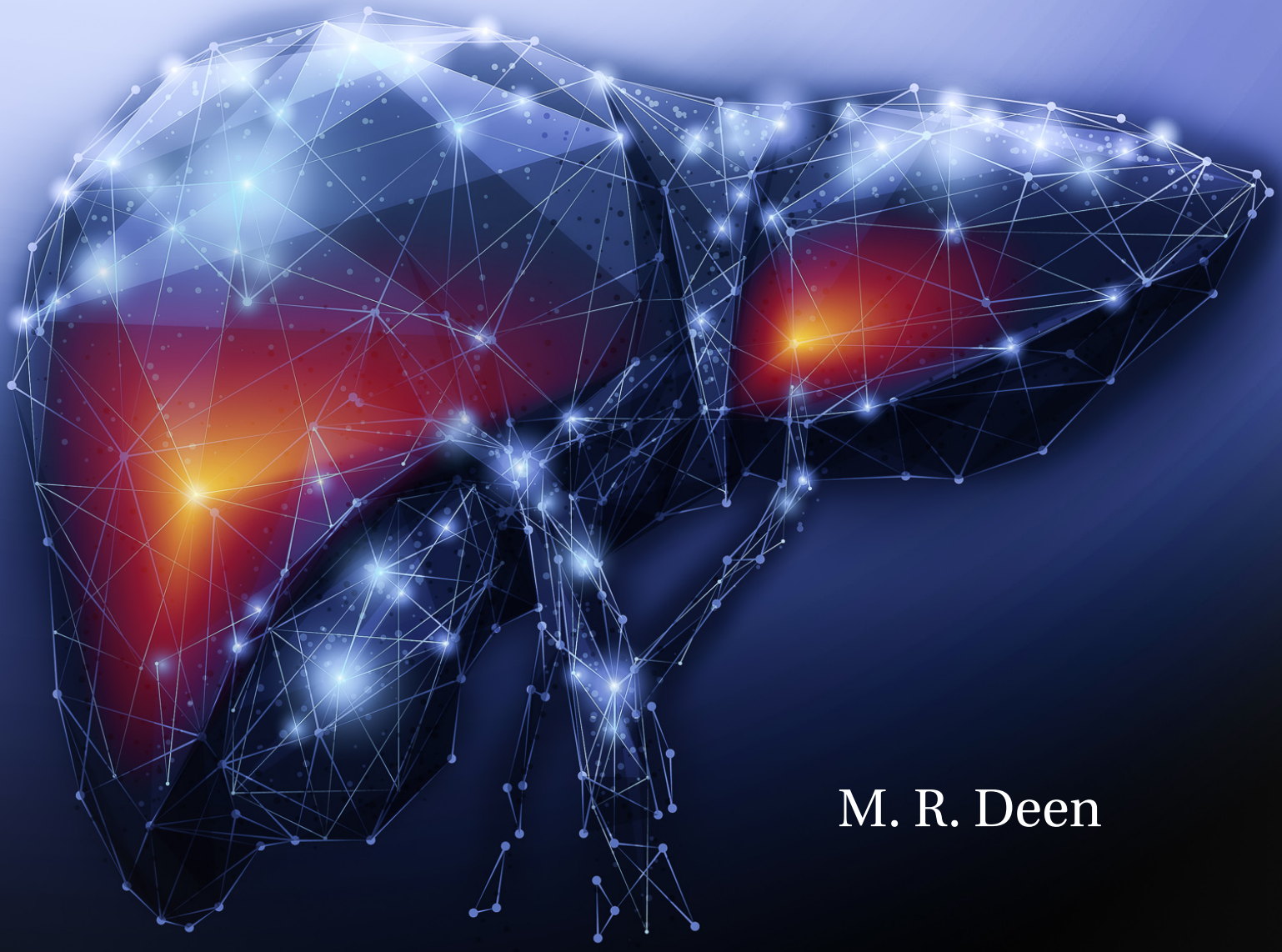
DELFT UNIVERSITY OF TECHNOLOGY

Master's Thesis

MSc Computer Science

Automatic algorithm selection and hyperparameter optimization for medical image classification



M. R. Deen

Automatic algorithm selection and hyperparameter optimization for medical image classification

by

M. R. Deen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on February 26th, 2021.

Student number:	4396340	
Project duration:	April 2020 – February 2021	
Thesis committee:	Ir. M. P. A. Starmans,	Erasmus Medical Center, daily supervisor
	Dr. N. Yorke-Smith,	TU Delft, supervisor
	Dr. Ir. S. Klein,	Erasmus Medical Center, supervisor
	Dr. L. Chen,	TU Delft

TU Delft

Abstract

Recent years have shown a tremendous increase in the application of Artificial Intelligence to the field of radiology, often through the extraction and analysis of large numbers of quantitative features from medical images. These applications increase the demand for machine learning models to extract information from these images. To provide these models, improve their performance and reduce the time that experts have to spend on manually tuning them, the field of Automated Machine Learning (AutoML) aims to automate the design process of machine learning models by optimizing the selection of algorithms and their hyperparameters for each application.

This work applies an AutoML approach to medical image classification, using a Bayesian optimization strategy to automatically optimize the selection of preprocessing and classification algorithms and their hyperparameters. Its performance is compared with the performance of a random search optimization strategy, evaluated on three datasets from three different clinical applications.

The results show that the Bayesian optimization and the random search return models that achieve similar performance on the unseen test sets. We show that a random search with relatively few evaluations and a simple ensemble strategy is sufficient to achieve performance comparable to a more sophisticated and more computationally demanding Bayesian optimization approach, therefore validating the use of a random search optimization strategy in this medical image classification setting.

All found models generalize poorly, with average F1-scores on the validation sets used for optimizing the models being at least 20% lower than the average F1-scores on the unseen test sets. Finally, we further emphasize the difficulty to generalize in this setting, by showing that the differences between subsets of the evaluated datasets are large and that increasing the computation time of the optimization does not benefit the test set performance of the final solution.

Preface

This report documents the research I have conducted for my Master's Thesis Project, as the concluding part of the MSc Computer Science program at Delft University of Technology. It describes the work I have done during a 9-month project — in collaboration with the Radiology and Nuclear Medicine department in the Erasmus Medical Center — on the application of automated machine learning to medical image classification.

First of all, I am very grateful to Martijn for his daily supervision on my project. Your expertise has been impressive to witness, but also on a personal level you have been very nice to work with. Your unfailing willingness to answer my questions has definitely been motivating and invaluable to my work. I really enjoyed our collaboration and I wish you all the best for what no doubt will be a successful academic career.

Second, I would like to thank Neil for being my TU Delft supervisor. It has been a great joy to work with and learn from you, not only during my final project, but throughout the whole Computer Science program. You have given me a lot of freedom and trust to organize my own project, but also encouraged me to keep informing you about my progress, which turned out to be a very pleasant way to work for me.

Finally, thanks to Stefan as well for his supervision on my project. Your comments during our many discussions were always extremely interesting; you really helped to make me see how fun this research actually was! Your positive attitude towards my work and your extensive knowledge on the field has been great to experience.

I feel very privileged to have been able to work with such knowledgeable, friendly and helpful people on this project!

Mitchell Deen
Delft, February 2021

Contents

1

Introduction

Recent years have shown a tremendous increase in the application of Artificial Intelligence (AI) to radiological images. One of the approaches that has gained momentum within this field is *radiomics* [55]: the extraction and subsequent analysis of quantitative features from medical images [39]. Radiomics enables the use of machine learning models to extract information from these images that is not discernible by visual inspection [67]. The main potential of this approach follows from the hypothesis that — through recognition and classification of relevant parts of the images [74] — the constructed models are able to provide valuable diagnostic or predictive information [38], hence facilitating better clinical decision making [26].

The added value of radiomics for clinical decision making increases, when we are able to use machine learning models with better performance on the classification task. However, the performance of many machine learning methods is very sensitive to a great number of design choices. It is therefore essential to find the right set of these choices when designing a machine learning model for a particular dataset. Unfortunately, doing so requires expert knowledge, is hard to reproduce, is time consuming and often of a trial-and-error nature [22]. In order to overcome these problems, the field of *Automated Machine Learning* (AutoML) aims to automate the design process of machine learning models and make decisions in an objective, data-driven way [34].

The Biomedical Imaging Group Rotterdam, at the department of Radiology and Nuclear Medicine in the Erasmus Medical Center, is using an AutoML approach for radiomics: the *Workflow for Optimal Radiomics Classification* (WORC) [58, 61]. WORC is a platform that executes all steps in the radiomics pipeline, from image data to an optimized classification model, automatically. WORC has been used successfully to find machine learning models for a variety of medical image classification problems [59, 60, 68, 69].

For the algorithm selection and hyperparameter optimization, WORC currently employs a random search strategy. Recently, however, the field of AutoML has seen great improvements in the performance of the machine learning models found for a wide range of problems [72, 73]. This development, combined with empirical evidence that guided search methods are able to outperform random search in AutoML problems [5, 6, 21, 52, 66, 71], introduces the hypothesis that WORC's current optimization strategy can be improved.

The first part of this work has been a literature search to determine the most appropriate optimization method to apply to AutoML in radiomics, with Bayesian optimization coming out as the most promising. The goal of this thesis is to compare the performance of Bayesian optimization with the performance of random search on algorithm selection and hyperparameter optimization for medical image classification, evaluated on three datasets from three different clinical applications. Furthermore, we aim to compare the potential of ensembles constructed from

the models returned by these optimization methods. Therefore, we specify the comparison that is central to this work in the following research questions:

1) How do Bayesian optimization and random search compare in terms of performance, when given the same computational resources?

2) What is the effect of different ensemble methods on the final performance of both Bayesian optimization and random search?

3) How do Bayesian optimization and random search compare in terms of consistency in a) their performance and b) the type of models they return?

The main contribution of this work is then twofold:

- From a clinical perspective, the optimization strategy put forward in this work has the potential to improve the performance of the machine learning models that are found using AutoML in radiomics on a large number of datasets, hence directly impacting the value these models can have for clinical practice.

- From a computer science perspective, the main contribution is the application of a Bayesian optimization strategy for AutoML in radiomics and gather empirical data on its performance compared to a random search strategy, as the application of AutoML in radiomics has not been extensively studied.

The remainder of this work is structured as follows. First, Chapter 2 reports on the literature search that was done to determine the optimization strategy. Then, Chapter 3 specifies the methods used in this research. Chapter 4 describes the design of the experiments, the results of which are presented in Chapter 5 and discussed in Chapter 6. Finally, Chapter 7 summarizes the most important conclusions and presents directions for future work.

2

Literature search: determining the optimization strategy

This chapter describes the literature search into the AutoML problem and the ways that have been put forward to solve it. First, it presents the general AutoML optimization problem that we are faced with in Section 2.1. Then, Section 2.2 provides an overview of the optimization methods that have been put forward in literature to solve this problem. Section 2.3 subsequently compares these methods and argues that Bayesian optimization is the most appropriate framework to apply. Finally, Section 2.4 gives some background on the concept of Bayesian optimization and its most important elements.

2.1. The general AutoML problem

The core practice of AutoML is to formulate (parts of) the design process of a machine learning model as a single hyperparameter optimization problem and solve that automatically. Every machine learning system has hyperparameters, so the most basic task in AutoML is to automatically find the set of hyperparameters that optimizes the performance of the chosen model. See Feurer and Hutter [22], Hutter et al. [33] and Luo [43] for overviews of automated hyperparameter optimization.

This problem can be extended to also include the choice of learning algorithm in the automatic optimization. This variant is known as *Combined Algorithm Selection and Hyperparameter optimization* (CASH) [66]. The choice of learning algorithm is usually modeled as an additional, categorical hyperparameter. The complexity of the optimization task is now increased, as we have to deal with conditional variables in the search space, since the choice of algorithm determines which of the hyperparameters are active in a particular configuration.

Finally, the *Full Model Selection* (FMS) [16, 20] problem includes the preprocessing steps — common in any machine learning application — in the search as well. Preprocessing steps for example include feature selection, dimensionality reduction, feature scaling and resampling. Most of the popular AutoML systems extend beyond the CASH problem and solve some version of the FMS problem, but the type and number of additional steps included in the optimization procedure differ between applications.

There are several benefits to applying machine learning in an automated way [22]:

- AutoML can respond to the increasing availability of large amounts of data and the increasing demand for machine learning applications. By greatly reducing the expert knowledge required for applying machine learning, these methods can become more widely accessible.

- AutoML can reduce the valuable time that current experts have to spend on manually tuning models.
- AutoML has the potential to improve the final performance of these models, by more effectively optimizing the selection of algorithms and hyperparameters for the application.
- AutoML can benefit the reproducibility of machine learning studies, by standardizing the approach to designing the models.

2.2. Overview of optimization strategies

This section identifies a number of strategies that can be used to solve the problem of automatic algorithm selection and hyperparameter optimization. We do not know the objective function we are optimizing; we can only observe the output of the function when we evaluate a specific input configuration. This type of optimization is known as blackbox optimization. There are a number of more detailed reviews of the current AutoML practice available with regards to optimization [33, 34, 43, 72, 73].

The first and most basic automated optimization strategy that can replace manual search is *grid search*. The procedure involves specifying a finite set of values for each variable and then evaluating all possible combinations of these values.

Random search instead creates input configurations by sampling randomly from a predefined distribution of values for each variable. This randomly sampled configuration is evaluated, after which the procedure is repeated a fixed number of times to traverse the search space.

Grid search and random search are not influenced by the results of the function evaluations during the optimization, as opposed to guided search methods. A branch of guided optimization strategies that has been successfully applied to AutoML problems is that of *population based methods*, which includes *evolutionary algorithms*. The most prominent examples are: particle swarm optimization (PSO) [20], where solutions move around the search space influenced by the local and global best solutions; and covariance matrix adaptation evolution strategy (CMA-ES) [29] and genetic programming [3], where both methods follow the general principle of evolutionary algorithms that revolves around the repeated application of variation and selection within the population of solutions.

The most commonly applied blackbox optimization method for AutoML in recent years is Sequential Model-Based Optimization (SMBO), specifically applying the *Bayesian optimization* framework [2, 53, 72, 73]. It works by maintaining a probabilistic model of the objective function and updating it with every function evaluation. The utility of evaluating new input configurations is then calculated, prioritizing areas of the search space that show high performance and areas that have not been explored yet. The input with maximum expected utility is evaluated and the procedure is repeated.

2.3. Comparison and choice of strategy: Bayesian optimization

This section compares the optimization methods mentioned in Section 2.2 to find the most appropriate one to be used for the AutoML optimization task in this work.

First of all, while easy to perform, grid search suffers from substantial problems. The required number of evaluations grows exponentially with each added variable, making an informative search of the space quickly computationally infeasible. Simultaneously, discretizing variables with a higher resolution also substantially increases the number of required function evaluations. Finally, grid search assumes that all parameters are of equal importance to the performance, which is often not the case [27].

Next, random search has been used effectively for blackbox optimization and hyperparameter optimization in particular [4]. Additionally, due to the independence of function evaluations,

random search is well-suited for efficient, parallel computation. Despite these advantages, especially within the AutoML literature, many guided search methods have been shown to consistently outperform random search [5, 6, 21, 52, 66, 71]. Random search will be the baseline strategy in this work.

Of the population based methods, PSO and CMA-ES suffer from problems when dealing with non-continuous domains. While CMA-ES is very effective for continuous blackbox optimization, it is not applicable to our search space of mixed-type variables [22, 28]. Similarly, PSO was originally designed for continuous domains - and while attempts have been made to allow for discrete variables [13] and even categorical ones [70] - PSO is rarely used for combined algorithm selection and hyperparameter optimization [72].

While genetic algorithms are much more widely applied to AutoML [72], the conclusions on their effectiveness within AutoML remain varied. Some research points at performance comparable to Bayesian optimization [12, 63], while other research — perhaps most notably that on the *Tree-based Pipeline Optimization Tool* (TPOT) [47], the most widely used application of genetic programming within AutoML [72] — reports performance comparable to random search [47–49].

Bayesian optimization is the most popular method in AutoML [73], and is the optimization strategy of choice for the prominent AutoML systems *Auto-WEKA* [66] and *Auto-Sklearn* [23]. It currently achieves state-of-the-art performance within AutoML, with the winner of the *AutoML challenge 2018* using a sophisticated Bayesian optimization technique [24].

Therefore, compared to the other considered algorithms, we find the evidence of the effectiveness of Bayesian optimization within AutoML to be most convincing, hence we choose to pursue this direction in our goal to improve the optimization of AutoML for radiomics.

2.4. Background on Bayesian optimization

Sequential model-based optimization (SMBO) [31] is a general framework for optimizing blackbox functions that formalizes the use of Bayesian optimization, hence both SMBO and Bayesian optimization usually refer to the same strategy. Bayesian optimization has two key ingredients: a probabilistic surrogate model and an acquisition function. The surrogate model is based on a prior distribution that describes the unknown objective function, which is updated with each observation made through function evaluations. The acquisition function provides some measurement of utility for each input configuration, which determines the next point to query. The basic procedure is shown in Algorithm 1. See Shahriari et al. [53] and Brochu et al. [10] for more details on the Bayesian optimization framework.

2.4.1. Surrogate models

The surrogate model has two important requirements [72]. First, the model should be able to make accurate predictions of what the function value will be, given a certain input configuration. Second, the model should be able to maintain a measure of uncertainty over those predictions. Three such models are most prominent in the AutoML literature [33]:

- Gaussian Process (GP)
- Tree-structured Parzen Estimator (TPE)
- Random forest

The great majority of literature on general Bayesian optimization uses GPs. In the AutoML literature, however, TPEs and random forests have been used more frequently, mainly due to their ability to handle conditional search spaces and their better applicability to high dimensional spaces.

input:
1) objective function $y = f(x)$ that we want to optimize
2) surrogate model S
3) acquisition function $\alpha(S, x)$

---

let $D = \emptyset$
initialize S
for n=1,2,..., **do**
  select new input x_{n+1} by optimizing acquisition function α:
    $x_{n+1} = \text{argmax}_x \, \alpha(S, x)$
  query objective function to obtain output y_{n+1}:
    $y_{n+1} = f(x_{n+1})$
  augment data with the new observation:
    $D_{n+1} = \{ D_n \cup \{ (x_{n+1}, y_{n+1}) \} \}$
  update surrogate model S with data D
end

Algorithm 1: Bayesian Optimization [53].

Gaussian Process. GP [51] is the traditional surrogate model for Bayesian optimization. Its main advantages are that it is fully specified by only a mean and a covariance function, while being able to express many different types of functions with closed-form computable predictions [22]. The choice of the covariance function determines the quality of the GP and is critical for its performance [54]. However, the standard kernel choices for GPs, such as the squared exponential function [10], are not suitable choices for the mixed-type configuration space in most AutoML problems. Distance-based kernels are also not able to distinguish between active and inactive variables, raising confusion on which of the variables are responsible for the model's performance.

One solution to this problem is to construct a separate process for each group of jointly active parameters [6]. More sophisticated kernels have been put forward by [65], [40] and [64], in order to capture the conditional relations in the search space.

Tree-structured Parzen Estimator. Proposed as an alternative to the GP model for high dimensional search spaces, the TPE algorithm does not model the posterior probability $p(y|x)$ of the objective function directly, but it constructs two different distributions $p(x|y < y^*)$ and $p(x|y \geq y^*)$ based on a certain threshold y^* [6]. This effectively divides the observations in good observations and bad observations. The ratio between the two probabilities immediately provides the function that expresses the expected improvement of new observations, making this a relatively simple approach. Using a tree of these estimators, conditional parameters can effectively be represented hierarchically.

Random Forest. The random forest surrogate model enabled *Auto-WEKA* and *Auto-Sklearn* through the Sequantial Model-based Algorithm Configuration (SMAC) method [31] to achieve good performance on large AutoML problems. A random forest is a collection of decision or regression trees [9] that combines their predictions on subsamples of the data. The random forest surrogate model works by learning a predictive function (using regression trees) that maps candidate input configurations to a performance value, with an uncertainty measure. Through evaluating different inputs and observing their performance, the random forest model is updated. Random forests scale well and can natively handle categorical and conditional search spaces [22, 33].

Comparison. As Yao et al. [73] conclude, more work needs to be done before GPs can work as well as tree-based models - like TPE and random forest - in conditional spaces. Additionally, tree-based models are a better choice when there are many data points, because GPs become computationally inefficient [64], or when the input space is high dimensional [19, 23, 32, 33]. In line with these observations, the two most popular AutoML systems, *Auto-WEKA* and *Auto-Sklearn* (and their variations), indeed use Bayesian optimization with tree-based models. While Thornton et al. [66] have demonstrated that random forest models outperform TPE models, the latter are still widely used in AutoML and remain a suitable choice [6, 19, 36].

2.4.2. Acquisition functions

The information that the surrogate model contains is used to determine the sequence of points to evaluate during the optimization. The utility of evaluating a new point is modeled through an acquisition function, which is optimized in each iteration to find the next point to evaluate. Generally, the algorithm will try new inputs that are either close to inputs that resulted in high performance, or that are in unexplored areas of the search space. To do this, the acquisition function calculates the utility by combining the performance predictions and the uncertainty predictions of the surrogate model. The function returns high utility values in those parts of the search space where the performance predictions of the surrogate model are high (exploitation) and where the uncertainty on those predictions is high (exploration). Common acquisition functions are *probability of improvement* (PI), *expected improvement* (EI) and *upper confidence bound* (UCB). See Shahriari et al. [53] and Brochu et al. [10] for an overview of these functions, including empirical evidence of their performance.

3

Methods

The first part of this chapter, more specifically Section 3.1, describes WORC [58, 61], the AutoML tool for radiomics that is used in this work. The second part of this chapter focuses on Bayesian optimization for WORC. Section 3.2 first formalizes the problem that we aim to solve. Section 3.3 then describes the optimization methods that are compared in this work. Finally, Section 3.4 covers the creation of ensembles of models after the optimization.

3.1. WORC: Workflow Optimal Radiomics Classification

The WORC tool has been used throughout this work for the automated classification of image data, enabling us to compare the algorithms used for the optimization of the machine learning methods and their hyperparameters. This section first provides a high-level overview of radiomics and therefore — since WORC automates the typical steps in a radiomics study — also of the functionality of WORC. We then limit our focus to the optimization part of WORC, defining the methods and hyperparameters that are included and how they are optimized and evaluated.

3.1.1. The radiomics workflow

The goal of radiomics is to combine quantitative features with machine learning methods to find relationships between medical images and clinical factors of interest [62]. WORC automates this process, visualized in Figure 3.1, by accepting labeled image data and a segmentation as input and returning an optimized machine learning model as output, that can potentially be used to predict relevant clinical outcomes on new data. The following steps are part of the automated workflow:

1) Image preprocessing.

2) Segmentation preprocessing.

3) Feature extraction.

4) Feature preprocessing.

5) Data mining.

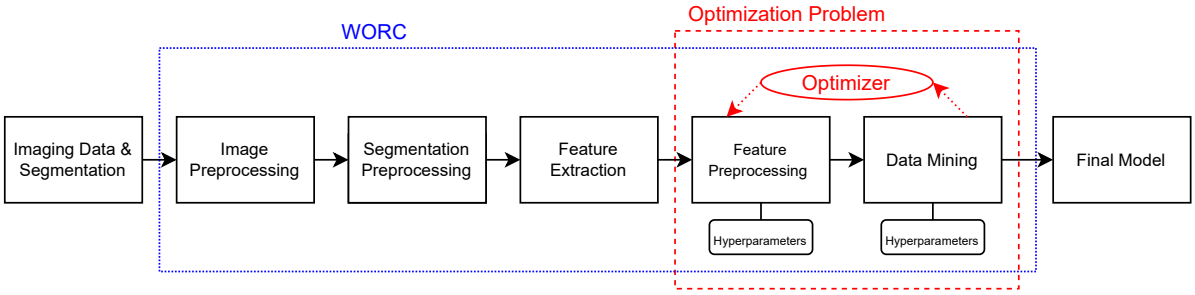


Figure 3.1: Radiomics workflow that is automated in WORC. The set of extracted features is fixed and is not included in the optimization.

The first step in the workflow is *image preprocessing*. It prepares the imaging data for further analysis, by using techniques such as discretization, normalization or resampling. The goal of these operations is to reduce the unwanted variation in the data, caused by differences in the condition in which the images were taken.

The second step in the workflow is *segmentation preprocessing*. The segmentation of the image determines the region of interest (ROI) that is used throughout the process. In most cases, the ROI indicates a specific part of the image where we want to extract the features, for example a tumor within an organ. The segmentation itself is provided to WORC as input along with the image data; in WORC there are methods available to process the segmentation further.

The third step is to *extract* the features from the ROI. These features form a quantitative representation of the characteristics of that part of the image. We will not cover the details of feature extraction methods here; see Starmans et al. [57] for details on the feature extraction methods that WORC applies. WORC extracts a standard set of 572 features that is used as the input to the optimization.

The fourth step, *feature preprocessing*, is the method used to go from the standard set of extracted features, to the set of features that is finally used as input for the data mining step. These methods may include feature imputation, feature scaling, feature selection, dimensionality reduction and resampling.

The final step in the workflow is *data mining*. While the data mining step in a radiomics workflow can include many techniques for data analysis, the scope of the data mining step in this work is defined to only include machine learning methods for classification. Therefore, to be more precise, we will refer to this step as the classification step.

The first three steps of the workflow are not part of the optimization; they are too computationally expensive to repeatedly evaluate as part of an optimization procedure. Therefore, the part of the workflow that is automatically optimized — and therefore also the part that is relevant to this work — includes the feature preprocessing and the classification. From here, we will limit our focus to these steps.

3.1.2. Search space

The feature preprocessing and classification steps include a number of different methods, each with their associated hyperparameters. Their combined setting defines the search space of the optimization, as the goal of the optimizer is to search for the best configuration of these methods.

Table 3.1 summarizes the methods that are included and their hyperparameters. This part of the workflow consists of eight steps that are executed in order. The first seven steps are feature processing methods; for each of these steps there is the option to execute one algorithm or none of the algorithms. For the final classification step, always one algorithm is used.

The workflow starts with feature imputation, where there is the option to impute feature values using one of five different algorithms. These algorithms calculate the value to impute based either

Table 3.1: Overview of feature preprocessing and classification algorithms included in WORC. For each numbered step in the workflow, zero or more algorithms are executed, except for the final classification step, where exactly one algorithm is used. The total number of variables that are part of the optimization, which includes the variables for selection of the methods, amounts to 55.

Method	**Step in workflow**	**Algorithm**	**Hyperparameters**			
			Bool.	Cat.	Disc.	Cont.
Feature imputation	1	mean				
	1	median				
	1	most frequent				
	1	constant				
	1	knn			1	
Feature scaling	2	robust z-score				
Feature selection	3	groupwise selection	25			
	4	variance selection				
	5	Relief			3	1
Dimensionality reduction	6	Principal Component Analysis		1	1	
Feature selection	7	statistical test		1		1
Classification	8	Support Vector Machine		1	3	1
	8	Random Forest			3	
	8	Logistic Regression		1		1
	8	Linear Discriminant Analysis		1		1
	8	Quadratic Discriminant Analysis				1
	8	Gaussian Naive Bayes				

on a descriptive statistic, a constant, or based on the value of a number of nearest neighbors.

The next step is feature scaling, which is always applied, where the feature values are standardized using the robust z-score. The robust z-score is a variation on the z-score, which works by first removing the outliers (< 5th and > 95th percentile) and then scaling using the z-score, thus reducing the influence of outliers on the scaling of the features.

Then follow three feature selection methods. First, groupwise selection, which simply activates and deactivates groups of features. There is a binary parameter for every group, determining whether those features are included. Second, variance selection, which removes all features that have a variance below a set threshold. Third, there is the option to apply the statistical feature selection method *Relief* [35].

The sixth step in the workflow is dimensionality reduction, using the Principal Component Analysis (PCA) algorithm [1]. The goal of PCA is to reduce the number of dimensions, while maximizing the variance of the data within the remaining dimensions.

The seventh and final step of the feature preprocessing is another feature selection method that selects features using a statistical significance test with a specified method and threshold, testing for predictive value of the feature with respect to the labels in the data. In WORC there is an additional

resampling step performed here, but it is not considered in this work, as its use has undesired implications for the size of the input at different points in the workflow.

Finally, in the classification step, one of six classifiers is used. The classifiers included are Support Vector Machine [15], Random Forest [9], Logistic Regression [30], Linear Discriminant Analysis [14], Quadratic Discriminant Analysis [25] and Gaussian Naive Bayes [41].

The search space contains a total of 55 variables. There are 8 variables to determine which methods are used, one for each step in this part of the workflow. Then there are a combined total of 47 hyperparameters of various data types. Not all of these hyperparameters are active at the same time; their activation depends on which algorithms are used. This complicates the optimization task, as the size of the (relevant) input to the objective function is not constant.

3.1.3. Optimization and evaluation

WORC enables the optimization of the feature preprocessing and the classification to find the best performing model. The optimization requires a performance metric to optimize for and a way to validate it during optimization time. After the optimization terminates, we require a way to evaluate it.

WORC optimizes for the *weighted F1-score* (or weighted F-measure) [50]. The F1-score balances two performance metrics: *precision* and *recall*. Precision is defined as the fraction of correct positive predictions over all positive predictions. Recall is defined as the fraction of correct positive predictions over all positive samples. The F1-score is the harmonic mean of these two measures:

$$2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{3.1}$$

The *weighted* F1-score is then a combination of the F1-scores calculated on both classes separately, adjusted for the weights of the classes based on their distribution in the dataset. In the remainder of this work, when we report the F1-score, we do always refer to the weighted F1-score.

Since the performance of a specific machine learning model is sensitive to which data it was trained and evaluated on, it is common practice to average the performance evaluation over a number of different splits of the data through cross-validation. Figure 3.2 summarizes the cross-validation setup used in WORC.
The performance that a specific model achieves during the optimization is computed using the average F1-score over a 5-times random-split cross-validation. Each random-split cross-validation uses 85% of the data for training and 15% of the data for validation.

When the optimization is completed and the model with the best found average F1-score over the splits on the validation set is returned, we evaluate the performance of the final model on a separate test set. To this end, we also split the initial dataset into a part used for training with 80% of the data and a part used for testing with 20% of the data. This test set remains separate from the optimization procedure and is only used for evaluation afterwards. This process of optimization and subsequent evaluation is then repeated 20 times, to gain a more accurate estimate of what we can achieve on unseen data and reduce our dependence on the specific split of the data.

Summarizing, this means that three different parts of the data are relevant within WORC; these will be referred to as the *training set, validation set* and *test set*:

- The 'outer' cross-validation splits the full dataset into a part used for training (80%) and a part used as the *test set* (20%).
- The 'inner' cross-validation splits the part of the data used for training into a *training set* (85%), that candidate models are trained on;
- and a *validation set* (15%), that the candidate models are evaluated on.

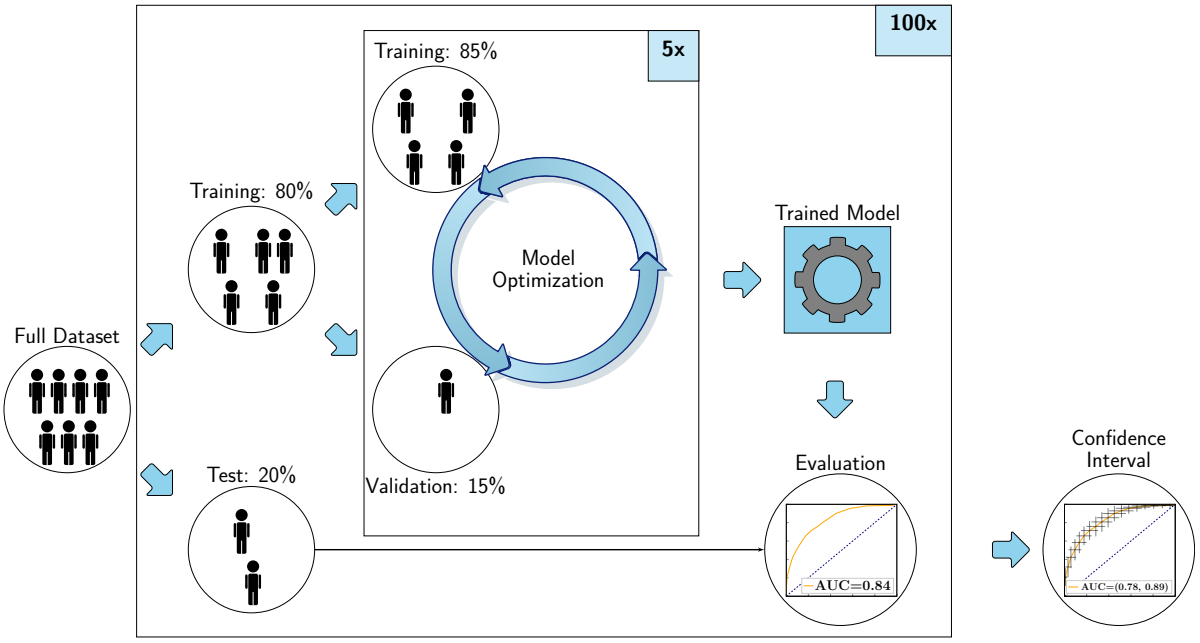


Figure 3.2: Cross-validation setup in WORC [68].

3.2. Problem specification

Before we describe the optimization methods used in this work, we define the optimization problem that we consider. We formulate the general problem as follows:

- **Data:**
  - Let $D = \{(x_1, y_1), ..., (x_N, y_N)\}$ be the training data used in the optimization, where, for each $i = 1, ..., N$, x_i represents the feature values and y_i the corresponding label.
  - Let D be split into R random-split cross-validations $\{D^1_{\text{train}}, ..., D^R_{\text{train}}\}$ and $\{D^1_{\text{valid}}, ..., D^R_{\text{valid}}\}$
- **Model parameters:**
  - Let $\mathscr{P} = \{p_1, ..., p_n\}$ be the set of steps in the workflow, where $p_1, ..., p_{n-1}$ are the feature preprocessing steps and p_n is the final classification step.
  - Each step $p_j \in \mathscr{P}$ has an associated hyperparameter setting $\gamma_j \in \Gamma_j$ that includes the algorithm selection for that step and its associated hyperparameters.
  - Then $c \in \Gamma$ is the full model configuration, where $\Gamma = \Gamma_1 \times ... \times \Gamma_n$.
- **Objective**
  - Let $\mathscr{L}(c, D^k_{\text{train}}, D^k_{\text{valid}})$ be the loss of model configuration c on D^k_{valid}, when trained on D^k_{train}.
  - The goal is then to find the optimal model configuration $\hat{c}$:

$$\hat{c} \in \underset{c \in \Gamma}{\operatorname{argmin}} \frac{1}{R} \sum_{k=1}^{R} \mathscr{L}(c, D^k_{train}, D^k_{test}) \tag{3.2}$$

We use the weighted F1-score as the loss function $\mathscr{L}$, we choose $R = 5$ and the current workflow contains $n = 8$ steps. The value of N depends on the size of the dataset.

3.3. Optimization methods

We compare two different optimization methods in their ability to find optimal model configurations for image classification. The first method, which is currently implemented in WORC and serves as the baseline, is random search (RS). The second method, which is introduced in this work, uses the Sequential Model-based Algorithm Configuration (SMAC) method [31, 42]. The SMAC algorithm was introduced in 2011 [31] as a sophisticated instantiation of the SMBO framework using Bayesian optimization. This algorithm was chosen for its validated ability to solve AutoML problems that include categorical and conditional parameters, through its use as the optimization algorithm in the prominent AutoML systems Auto-WEKA [37, 66] and Auto-Sklearn [23], the latter often considered the state-of-the-art for generally applicable AutoML systems [72].

3.3.1. Random search

The RS algorithm follows the following procedure:

1) Generate a fixed number of model configurations. For each model, the parameter values are drawn from pre-specified distributions.

2) Fit and evaluate all models.

3) Rank the models and return (a subset of) them.

The RS implementation allows for parallel execution of the second step. Each parallel process then computes the performance of a subset of the total number of generated configurations.

3.3.2. SMAC

The optimization method using the SMAC algorithm follows the following procedure:

1) Start a fixed number of independent SMAC instances.

2) For each instance:

    a) Initialize the surrogate model with a set of random function evaluations.
    b) Repeat until the evaluation limit is reached or the time budget has run out:
        - Optimize the acquisition function to find the next model configuration to evaluate.
        - Fit and evaluate the model.
        - Store the result and update the surrogate model.
    c) Return all evaluated models.

3) Combine the results of all instances, rank the models and output (a subset of) them.

Initialization SMAC's surrogate model requires an initial set of points before the acquisition function can be effectively optimized. Our algorithm uses a random initialization, to ensure variation across the starting points of each of the independent SMAC instances.

Surrogate model. SMAC uses a random forest as the surrogate model of the objective function. The random forest maps candidate parameter settings to predicted performance values, which are used by the acquisition function. In SMAC, the random forest is constructed from $B = 10$ regression trees, where each tree is built on N data points, sampled randomly with repetitions from the full training set of size N. At each node in the regression tree, a random selection of $\lceil d \cdot p \rceil$ of the d input parameters is available to use as a splitting criterion. The standard setting for p is 5/6. The trees are split until less than $n_{min} = 10$ data points are left. These three hyperparameters of SMAC, B, p and n_{min}, were left at their default value [31]. The random forest's final prediction for a new input parameter setting is the empirical mean and variance of the individual trees' predictions.

Acquisition function. SMAC implements the common *Expected Improvement* (EI) function to quantify the utility of evaluating a new input configuration. In order to find the next point to query, this function needs to be optimized. SMAC computes the EI for all previously evaluated input configurations and employs a multi-start local search from each of the ten inputs with highest EI. Since computing the EI is not expensive, SMAC finds the EI for an additional 10000 randomly sampled input configurations. Then, during optimization, both randomly sampled configurations and those found by the acquisition function are evaluated and compared to the current best solution.

Parallel computation To increase the number of SMBO iterations that can be executed per unit of time and to compete with the highly parallel implementation of random search, the optimization was designed to run in independent parallel instances. Combining a search history over different instances of the algorithm resulted in too much computational overhead. Therefore, each instance of SMAC does its own search and the best solutions over all instances are identified and stored afterwards.

3.4. Ensemble methods

Both optimization strategies evaluate many different models during their search. Instead of only using the single best model, we can create an ensemble of models and use that as our final predictive model, with the potential to improve performance and robustness [17].

To have access to a set of good models after the optimization, we save a subset of the models that are returned by the optimization algorithms. To limit storage requirements, we save the best 100 models. This set of 100 models then forms the input to a number of methods that combine them into an ensemble.

All constructed ensembles work in the same way, by taking the average posterior probability of the class predictions of all individual models in the ensemble as its final prediction. The difference between the methods lies in the selection of models for the ensemble. Two ensemble methods are currently used in WORC:

- **Top 50**. The default method, which selects the best 50 individual models.
- **Top n**. Similar to the default method, this method selects the first n of the individual best models, choosing $1 \leq n \leq 100$ that gives the highest performance on the validation set.

We introduce three new methods in WORC:

- **Forward selection**. Adds the model to the ensemble, out of the 100 available ones, that results in the best performance on the validation set. It then repeats this process, with replacement, until adding any of the models results in no more improvement.
- **Caruana**. Based on Caruana et al. [11]. Same as the forward selection method, but repeats the process for a fixed number of 20 iterations.
- **Bagging**. Applies the Caruana method with bagging. In each of the 20 bags, only a random selection of half of the models is available for the ensemble creation. The final ensemble prediction is again the average over the individual ensembles' posterior class probabilities from each bag.

4

Experimental Design

This chapter describes the experiments designed to compare the performance of WORC when using either the RS algorithm or the SMAC algorithm for its optimization. Section 4.1 specifies the goals of the experiments, after which Section 4.2 defines the setup of the experiments in order to achieve those goals. The algorithms have been compared on three different datasets; Section 4.3 provides a brief overview of these. Finally, Section 4.4 contains details on the hardware used to run the experiments.

4.1. Objective of the experiments

The experiments were designed to address the following three questions:

1) **Performance versus runtime:** How do RS and SMAC compare in terms of performance, when given the same computational resources?

2) **Ensemble methods:** What is the effect of different ensemble methods on the final performance of both RS and SMAC?

3) **Solution consistency:** How do RS and SMAC compare in terms of consistency in a) their performance and b) the type of models they return?

By including the results on the validation set, as well as the independent test set and comparing them, we aim to gain insight into the generalization error.

4.2. Experimental setup

This section covers the design of our experiments, each targeting one of the formulated questions.

4.2.1. Performance versus runtime experiment

The main experiment is to compare the performance between RS and SMAC, when given the same computation time budget. The performance is defined in terms of the weighted F1-score. The performances of the single best found solutions by RS and SMAC, optimized on the validation set, are compared for a range of computation times on both the validation set and the test set. Since the RS implementation does not support a computation time limit on the optimization, but only accepts as input the number of configurations to randomly generate and evaluate, a range of 9 different evaluation limits were chosen first. These were both smaller and larger than the default used for WORC in clinical studies (ranging from 1,000 to 150,000 evaluations; the default value is 25,000). The runtime values returned during these experiments were then given as computation time budgets to the SMAC algorithm, resulting in equal computational resources for both methods.

Runtime. The RS runtime is defined as the combined wallclock time used for fitting and evaluating the models, calculated separately for and summed over each of the independent processes that might run in parallel on different cores. Similarly, the SMAC runtime is the wallclock time spent on fitting and evaluating the models, as well as on selecting new points to query. This implementation ensures that scheduling overhead on the computer cluster used for experimentation does not influence the runtime calculations. The implementation is not independent of the cluster file write speeds, however; this effect is not taken into account during the experiments.

Cross-validations. The experiments use a random-split 5-fold 'inner' cross-validation to evaluate model performance during optimization. These validation set scores are reported to see the pure performance of the optimization strategy, since these are the values that the algorithms are actually optimizing for. Then, the 'outer' train-test cross-validation is repeated 20 times, to balance the effect of randomness in the data split with the computational cost of the experiment. From previous experience, we know that increasing the number of train-test cross-validations has a limited effect on the confidence intervals of the final results, while having a great impact on overall runtime. For both the scores on the validation set and on the test set, the final result is the average over all train-test split scores of the best found models for each split. For all experiments and for both types of cross-validation, exactly the same splits are used in each step by using a fixed random seed.

Statistical comparison. To compare the final average performance of both algorithms on the test set, we use the corrected resampled t-test [8, 45] with significance level $\alpha = 0.05$. This statistic corrects for the lack of independence between the randomly sampled train-test splits. Additionally, we use this corrected measure to construct the 95% confidence intervals around the mean values over the train-test splits.

4.2.2. Ensemble methods experiment

To compare the different ensemble methods, the optimizations from the performance versus runtime experiments are reused. The ensembles are then constructed from the best 100 stored configurations for each algorithm run. The ensemble experiments show the effect of the different ensemble construction procedures on the validation and test performance, for different runtimes and for both SMAC and RS.

4.2.3. Solution consistency experiment

While the performance experiments are insightful to the relation between performance and runtime and whether this differs between SMAC and RS, the sample size remains small. The goal of this experiment is to repeat the same algorithm run many times, to discover whether the algorithms find consistently good models and what these models generally look like. To this end, the default setting of WORC (25,000 configurations) and its SMAC equivalent in terms of runtime are run 50 times on a fixed set of 10 train-test splits, with fixed 5-fold random-split cross-validation. Based on these results, the variance in performance can be analyzed for each of these splits individually, while also allowing a more in-depth look at the specific models that were found.

4.3. Datasets

The three Magnetic Resonance Imaging (MRI) datasets included in this work originate from previous radiomics studies where WORC was used to find the predictive models. All datasets contain heterogeneous data, as a result of the scans often originating from different medical institutes with different scanning hardware or protocols. For more details on the data and background on the

classification problems, see the papers that use the DM dataset [68], Lipo dataset [69] and BLT dataset [59]. For a summary of the most basic characteristics of the three datasets, see Table 4.1.

Table 4.1: Summary of the three datasets.

	DM	**Lipo**	**BLT**
Positive class patients	79	57	94
Negative class patients	142	58	94
Total patients	221	115	188
Classification	Desmoid-type fibromatosis vs. non-desmoid-type fibromatosis	Lipomas vs. well-differentiated liposarcomas	Benign vs. malignant liver tumors
Tumor Location	Muscular tissue	Soft tissue	Liver
Image type	T1-weighted MRI	T1-weighted MRI	T2-weighted MRI

4.3.1. 'DM': Desmoid-type fibromatosis vs. non-desmoid-type fibromatosis

Desmoid-type fibromatosis (DTF) is a rare form of soft tissue tumor that can arise in muscle tissue. In order to distinguish DTF from several malignant, non-DTF soft tissue tumors, an invasive tissue biopsy is required. To improve this practice, this dataset has been used to study the possibility of automatically distinguishing DTF from non-DTF [68].

With 221 patients (79 DTF, 142 non-DTF), this is the largest of the three datasets. It is, however, heterogeneous, with the original data coming from 68 scanners [68]. Additionally, the tumors can arise in different parts of the body, introducing more variety in the background of the images. As opposed to the other datasets, the classes are not exactly balanced.

4.3.2. 'Lipo': Lipomas vs. well-differentiated liposarcomas

This dataset contains MRI scans of benign lipomas and malignant, well-differentiated liposarcomas (WDLPS). Again, a predictive model was developed to automatically predict the class [69].

This dataset is the smallest and contains 115 patients (58 benign, 57 malignant), all from the Erasmus Medical Center, but with balanced classes. Again, the scans were produced with a wide range of different imaging hardware and acquisition protocols, originating from 41 different scanners [69]. The heterogeneity of this dataset is reinforced by the fact that lipomatous tumors are not located in a specific part of the body.

4.3.3. 'BLT': Benign liver tumors vs malignant liver tumors

This dataset contains MR images of patients with benign and malignant liver tumors. This dataset has been used to distinguish both types of tumors in a non-invasive manner [59].

The dataset includes 188 patients (94 benign, 94 malignant), with the images originating from 42 different institutes [59], again introducing heterogeneity. The classes are balanced and the image backgrounds are relatively consistent, due to all scans being located in the liver.

4.4. Hardware

To run the experiments we made use of two different computer clusters: The BIGR cluster, the high-performance computing cluster of our research group, and the Cartesius cluster, a Dutch supercomputer available for research purposes. Specifications of the systems are summarized in

Table 4.2. Since both systems differ in specifications, runtime comparisons could not be made between the two. Hence, all experiments that compare the performance between RS and SMAC for fixed runtimes have been run on the BIGR cluster. The remaining experiments were run on the Cartesius cluster, using the Broadwell nodes.

Table 4.2: Hardware specifications of both computer clusters.

	BIGR	Cartesius
Cores per node	48	32
RAM per node	256 GB	64 GB
Core limit	140	32
Processor	AMD Opteron 6172 2.1 GHz	Intel Xeon E5-2697A v4 2.6 GHz

5
Results

This chapter presents the results of the comparison between RS and SMAC, as well as the performance of different ensemble methods applied after performing the model search. Following the structure of the experiments outlined in Chapter 4, Section 5.1 covers the first experiment: comparing the performance of RS and SMAC given a fixed runtime budget. Section 5.2 then shows the effect on the performance when applying a selection of ensemble methods to the same problems. Finally, Section 5.3 provides a more in-depth look at the consistency of SMAC's output compared to that of RS, both in terms of performance and in terms of the type of solutions found.

5.1. Performance versus runtime

This section covers the first experiment, comparing the performance of RS and SMAC when they are given equal computation time budgets. The first part of this section focuses on the validation scores of the single best solutions returned by the optimization algorithms and the second part focuses on the test scores achieved by these solutions.

5.1.1. Validation set performance versus runtime

Figure 5.1 shows the comparison between RS and SMAC on the three datasets. First, the focus is on the validation scores. For each of the three datasets, the algorithms were run with equal, fixed limits on the allowed computation time. The reported value is the average F1-score over the 20 train-test splits, of the average 5 times random-split cross-validation score using the training set. The errorbars for the results on the validation set indicate one standard deviation above and below the mean values.

A similar pattern is apparent for each of the datasets. Given more computation time, the performance of the single best found model on the validation set increases, the curve flattening when the algorithms are given more time. The diminishing returns of more computation time are slightly more apparent for RS compared to SMAC. In terms of absolute values, the average F1-scores are highest for the Lipo dataset, with values ranging between 0.83 and 0.90, followed by the DM dataset (0.79 - 0.87) and the BLT dataset (0.77 - 0.85).

RS performs somewhat better for low runtimes, with improvements of up to 2%. Starting from 250 computation hours, depending on the dataset, SMAC starts — on average — to outperform RS, but the mean values lie within one standard deviation. At the largest runtime that was part of this experiment, with RS running for 150,000 evaluations, the difference is more pronounced: SMAC achieves a 2.6%, 3.5% and 2.9% increase in the average F1-score compared to RS, for the three datasets respectively.

On average, given the same computational budget, SMAC performs around 40% of the number

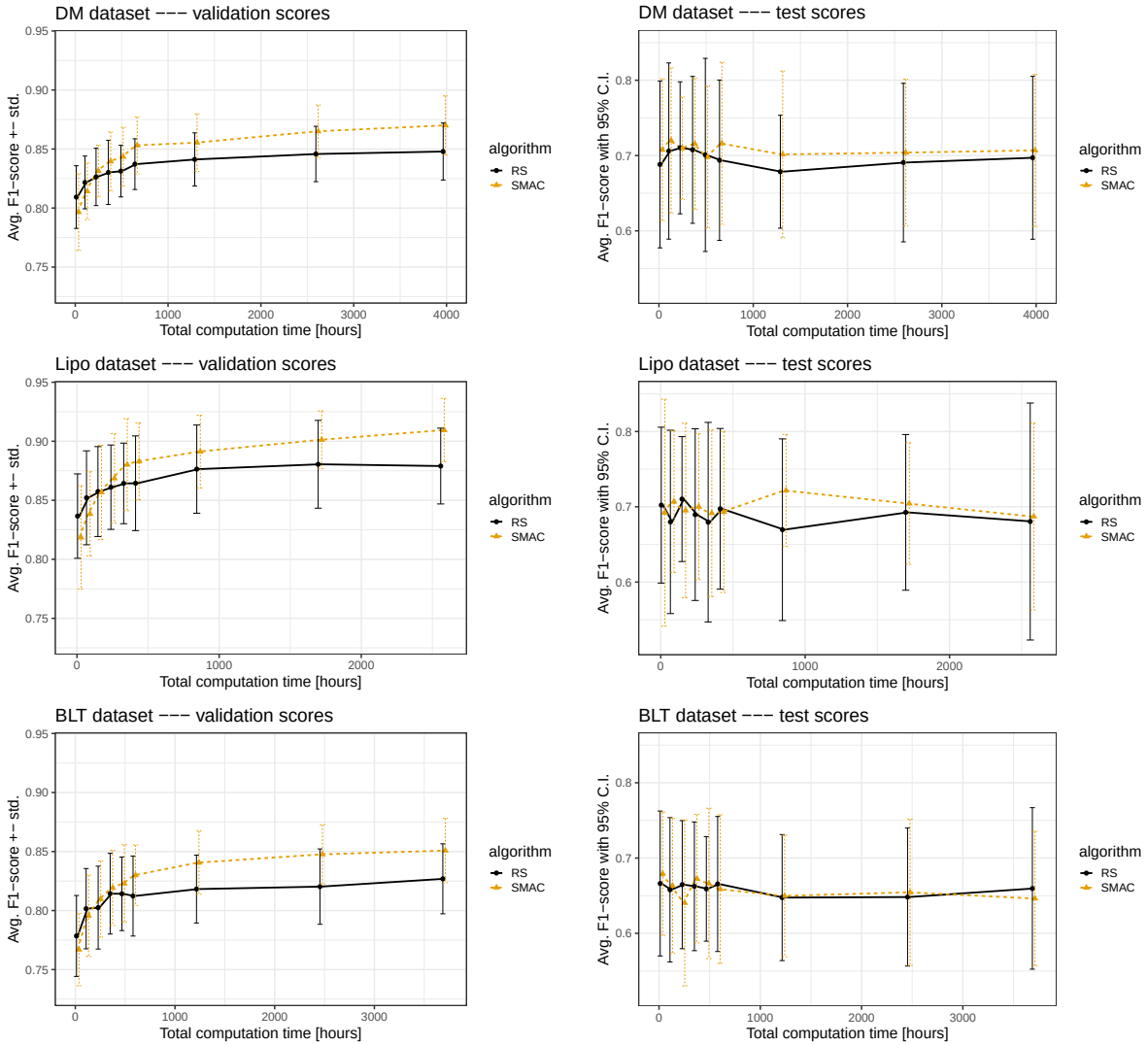


Figure 5.1: Average F1-score on the validation set (left, with standard deviation) and the test set (right, with 95% confidence interval) over the 20 train-test splits of the single best solution for each of the three datasets, for varying computation times. The positions of the points with equal computation times have been shifted slightly to improve readability.

of function evaluations compared to RS. This means that SMAC spends the other 60% of the available time on finding the next point to query.

5.1.2. Generalization: Test set performance versus runtime

Using the average value of the performance on the test set of the 20 train-test splits, we estimated the performance on unseen data of the best models found during the optimization. The result, as displayed in Figure 5.1, is clearly different from the performance on the validation set. No longer does increasing the computation time lead to better average F1-scores. RS with 1,000 or 150,000 random evaluations now leads to similar average F1-scores on the test set. The 95% confidence intervals, represented by the errorbars around the mean test scores, range from 0.15 to 0.20 in average width (± 10 to 15% around the mean) and are large compared to the differences between the methods. Longer computation time does not reduce the size of the confidence intervals.

Overall, RS and SMAC show comparable performance on the test set. None of the compared runs resulted in a difference in performance statistically significant from zero; all corrected

resampled t-tests returned a p-value greater than 0.05.

Finally, the ability of the found models to generalize is evaluated by comparing their performance on the validation set with their performance on the test set. For all three datasets, the average F1-scores on the test set are up to 20% lower than those found during training time using random-split cross-validation. The generalization error increases for both algorithms when given additional computation time. For the DM dataset, the average F1-scores on the test set are 0.09 to 0.17 lower than on the validation set, with an average test performance of 0.70. For the Lipo dataset, this is 0.14 to 0.21 lower, with an average test performance of 0.69. Finally, the BLT dataset sees a 0.11 to 0.19 decrease in performance with an average test score of 0.66.

5.2. Ensemble performance comparison

In this experiment, we apply each of the ensemble methods to the set of 100 best models that were found by the two algorithms, again for increasing computation time budgets.

5.2.1. Performance on the validation set

Figure 5.2 shows the F1-score of the different ensemble methods evaluated on the validation set. To ensure readability of the plots, the standard deviations around the mean values have been omitted.

The three datasets show similar results; the ranking of the ensemble methods is consistent throughout the experiment. Two methods, namely bagging and top 50, create an ensemble that, on average, performs worse than the single best solution on the validation set. In most scenarios, the other methods do increase the performance of the final model by using their ensemble.

For RS, there is a consistent, seemingly fixed increase in the performance value when using these methods, regardless of total computation time. Caruana provides the largest increase in performance of around 6% compared to the best individual model.

For SMAC, the benefit of the ensemble decreases for longer computation times, with the final performance of all methods appearing to converge to that of the single best solution. For shorter computation times, the ensemble methods show a pattern similar to the results of RS.

5.2.2. Performance on the test set

Figure 5.3 presents the performances of the ensembles on the test set. The 95% confidence intervals are omitted for readability; Table 5.1 instead contains the average width of the intervals for the performance on the test set. These aggregated results lose information on the effect of the computation time on the confidence interval. However, similar to the results shown in Figure 5.1, this effect is minimal, hence the average value is a sufficient indicator. While some of the ensemble methods are able to reduce the average width of the confidence intervals, they are still large compared to the individual differences between the methods, hence the intervals do overlap for all data points.

First, we observe the effect of the ensemble methods on the performance. Compared to the ranking of the methods on the validation set, the effectiveness of Bagging and top 50 on the test set stands out; these were the methods with the lowest performance on the validation set.

For RS, Bagging and top 50 now provide the largest performance increase over the single best solution, which has the lowest performance on the test set. For Bagging, this increase in the average F1-score ranges from 0.03 to 0.05, which is up to 7% for the Lipo dataset. The other ensemble methods achieve test scores somewhere in between. The computation time of the optimization has little effect on the performance of the RS ensembles.

For SMAC, the differences between the ensemble methods are smaller compared to RS, resulting in a smaller performance increase from the ensemble (rarely exceeding 0.02) compared to the single best solution. In general, Bagging and top 50 again perform the best on the test set, but the difference with Caruana, FS and top N is less pronounced compared to RS. Additionally, the average

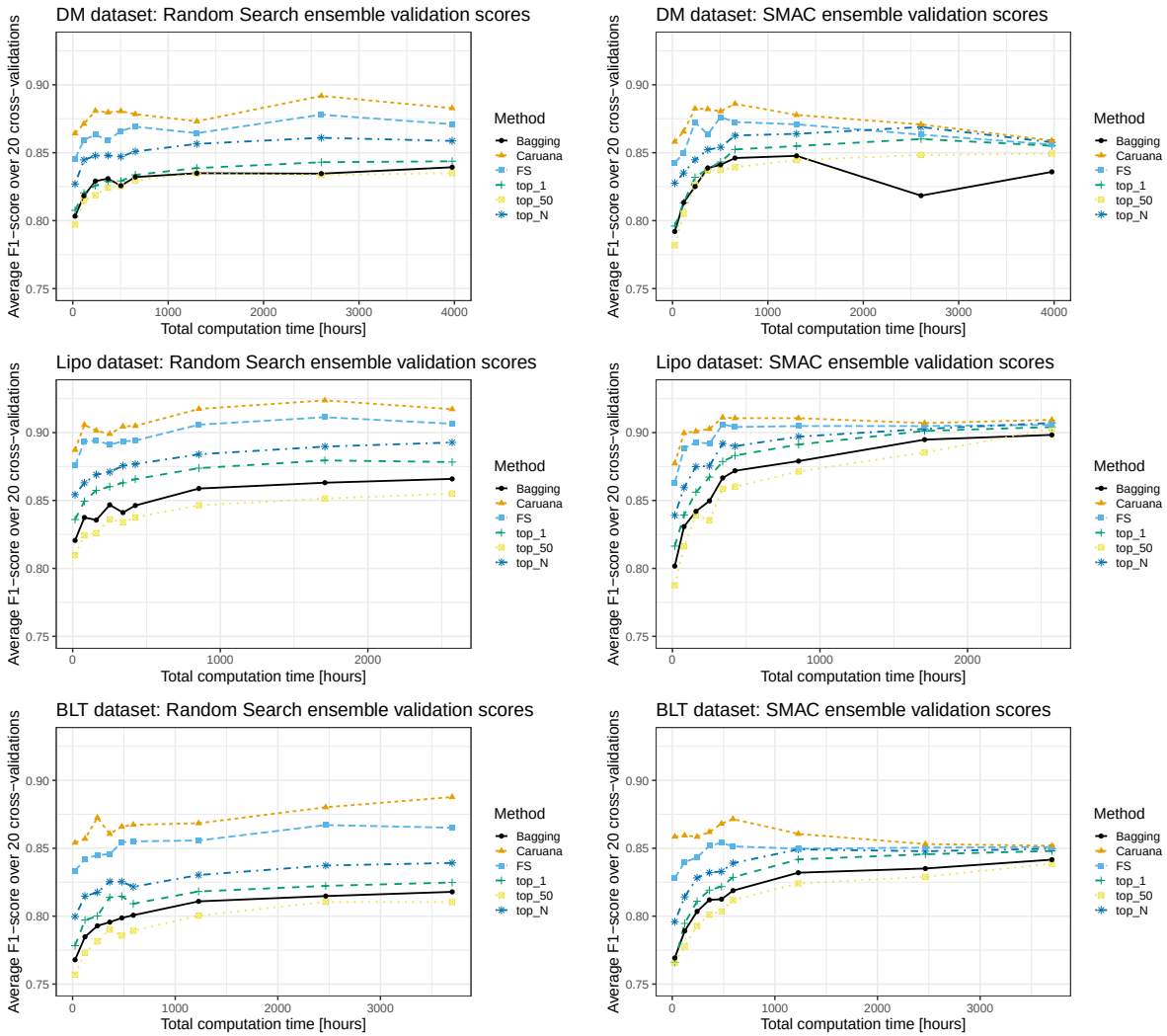


Figure 5.2: Average F1-score on the validation set over 20 train-test splits for RS (left) and SMAC (right) versus the computation time of the optimization that resulted in the set of models available for the ensemble, for a selection of different ensemble methods. The visualized computation time is therefore unrelated to the computation time of the ensemble methods themselves.

Table 5.1: Average width of the 95% confidence interval around the mean F1-score over the 9 instances for each algorithm.

		Top 1	Top 50	Top N	FS	Caruana	Bagging
DM	RS	0.151	0.136	**0.135**	0.147	0.137	**0.135**
	SMAC	0.157	**0.147**	0.153	0.149	0.150	0.148
Lipo	RS	0.209	0.175	0.187	0.182	0.187	**0.168**
	SMAC	0.192	**0.162**	0.184	0.179	0.175	0.169
BLT	RS	0.181	**0.142**	0.158	0.180	0.173	0.154
	SMAC	0.173	**0.161**	0.171	0.162	0.169	0.168

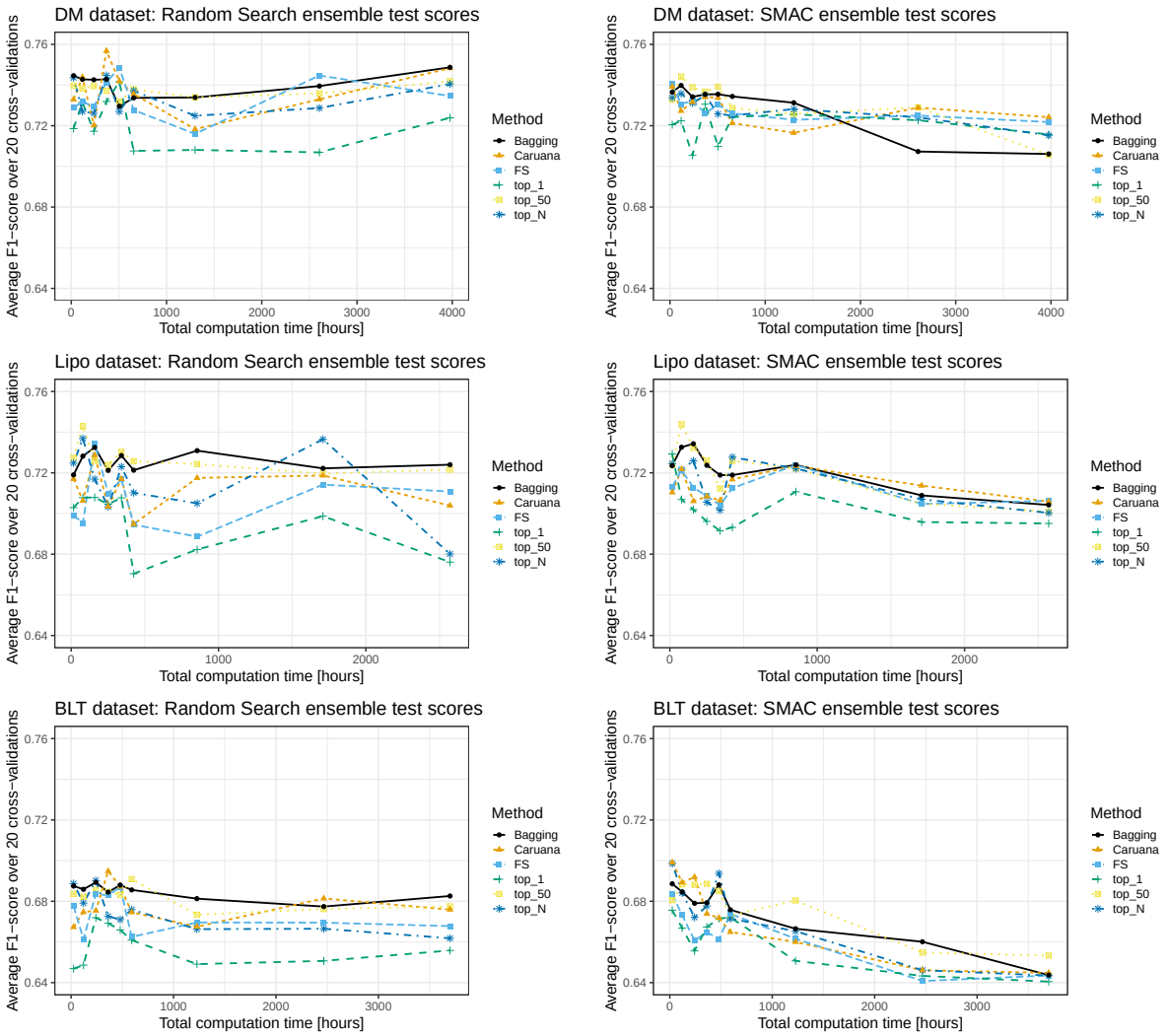


Figure 5.3: Average F1-score on the test set over 20 train-test splits for RS (left) and SMAC (right) versus the computation time of the optimization that resulted in the set of models available for the ensemble, for a selection of different ensemble methods. The visualized computation time is therefore unrelated to the computation time of the ensemble methods themselves.

F1-scores seem to be either equal or lower for increasing computation time of SMAC.

Second, we observe the effect of the ensemble methods on the width of the confidence intervals. All of the tested methods manage to reduce the average width of the confidence intervals compared to the single best solution. Again, Bagging and top 50 show the best results. In the most extreme case, the top 50 ensemble using RS on the BLT dataset, this is a reduction in the average width of 20%. Comparing RS to SMAC, we see that the RS ensembles reduce the average width of the confidence intervals more on the DM and BLT datasets, while the SMAC ensembles achieve a larger reduction on the Lipo dataset.

5.3. Consistency comparison

For the previous experiments, which explored the effect of increased runtime on the performance of both algorithms, each performance value was only calculated once, using a 20 split cross-validation. In this experiment, the results are displayed in Figure 5.4, we choose a single setup for RS and SMAC and repeat it 50 times, to investigate how consistent the performance of these algorithms is when

given the same problem multiple times. Unfortunately, due to some errors on the Cartesius cluster, a handful of repetitions for both RS and SMAC failed (seemingly randomly). The impact on the overall results of the experiment is, however, believed to be minimal; hence our decision to keep the results as they are. For this experiment, we used the baseline version of RS, i.e. 25,000 iterations, and SMAC with an equal runtime budget on 10 fixed train-test splits.

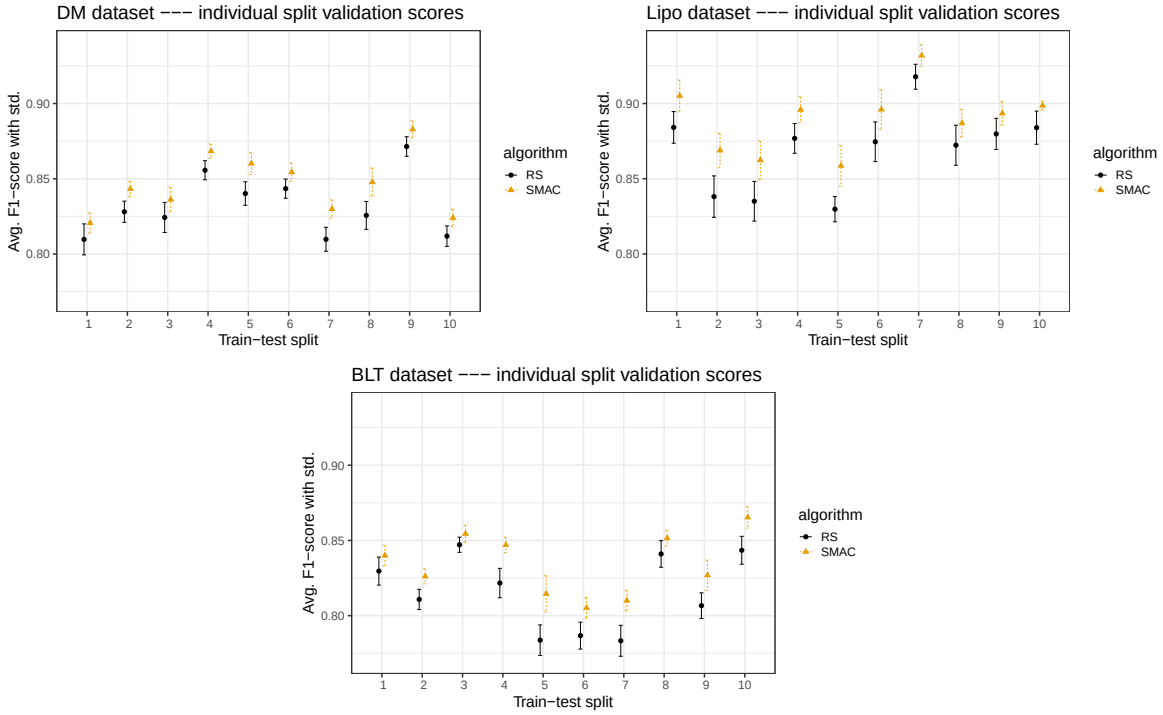


Figure 5.4: Average F1-score (plus/minus one standard deviation) of the single best solution on the validation set over 50 instances of 10 train-test splits. The train-test splits were identical for each of the 50 runs.

5.3.1. Performance consistency

The first observation that can be made regarding the validation scores on the individual splits is the consistent higher average of SMAC compared to RS. For all three datasets and each of the ten train-test splits, SMAC is able to find models that on average perform better on the validation set than the best models found by RS. This reinforces the earlier observation that SMAC achieves slightly higher scores on the validation set compared to RS, when it was averaged over the 20 train-test splits.

The second observation is that within each split the variance in performance is small compared to the variance between splits, which holds for both RS and SMAC. Naturally, this large variation between random splits of the dataset results in larger confidence intervals for the overall average performance estimate.

Finally, the standard deviation of SMAC is, on average, smaller than that of RS. In these experiments SMAC managed to reduce the average standard deviation of the F1-scores per split by 17%, 13% and 18% for the DM, Lipo and BLT datasets respectively.

5.3.2. Solution consistency

Using the repeated runs on the fixed splits, we now turn to a more qualitative analysis of the returned models. We first assume that the choice of classifier in the best found model is, to some extent, a distinguishing characteristic for that model. We then compare how many times each of the classifiers has been found by RS and SMAC during the optimization, in the repeated runs for each split separately.

The results are varied. Figure 5.5 shows a selection of the 30 train-test splits that were part of the experiment. This selection was made to visualize the most important observations from the complete experiment. In general, similar observations can be made regarding each of the three datasets. First of all, in terms of the data, it is again apparent that the variation between the splits is substantial: not only in performance, as we saw previously, but also in the models that are found. For some splits, like number 6 from the DM dataset, 10 from the Lipo dataset, and 4 from the BLT dataset, there appears to be a single classifier that is most applicable for that split in the data. However, this one classifier can differ radically per split; compare for example splits 9 and 10 from the Lipo dataset. For other splits, two classifiers are suitable (see split 8 from BLT), or even more are able to achieve a good result (see DM split 1, Lipo split 3 or BLT split 6).

The differences between the two algorithms are perhaps less obvious. In a majority of the total cases, RS and SMAC seem to 'agree' to a great extent on which classifiers are best to use for that split. Only in very rare cases, such as in split 5 from the BLT dataset, we see a clearly pronounced difference in the classifiers of best found models. For the remainder of the splits, the level of agreement lies somewhere in between; DM split 8 would be a typical example of this.

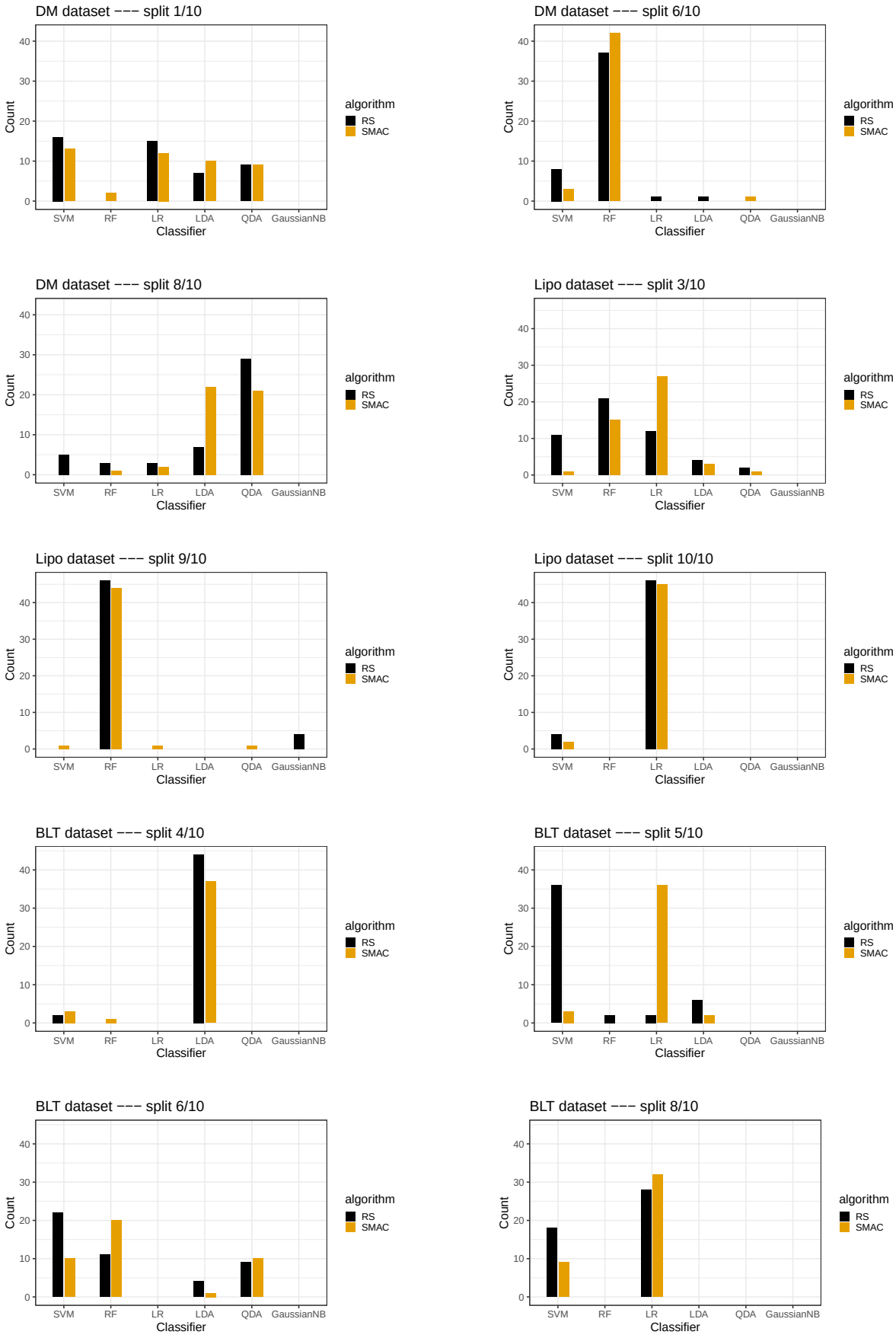


Figure 5.5: Counts of classifiers used in the best found models returned by the RS and SMAC algorithms during the optimization (DM: $N_{RS} = 47$, $N_{SMAC} = 46$; Lipo: $N_{RS} = 50$, $N_{SMAC} = 47$; BLT: $N_{RS} = 46$, $N_{SMAC} = 41$).

6

Discussion

This chapter summarizes and interprets the results presented in Chapter 5. First, the answers to the three main research questions are explored in Section 6.1. Then, in Section 6.2, we mention the most important limitations of this work.

6.1. The research questions

The goal of our experiments was to answer three main questions: (1) How do RS and SMAC compare in terms of performance, (2) what is the effect of different ensemble methods on the performance of the final model and (3) how do RS and SMAC compare in terms of solution consistency.

6.1.1. Research Question 1: RS and SMAC performance comparison

We evaluated the performance of both algorithms in terms of the average F1-score on the validation set and on the test set. For the discussion of this research question, we first examine the results on the validation set and then continue with the test set.

While having limited practical applicability, performance on the validation set is the best indicator of the pure optimization performance of our strategy. The results show that RS finds slightly better models for small computational budgets (up to roughly 250 hours, or around 10,000 RS evaluations). This is not unexpected, given that SMAC needs to train a random forest model to predict the performance of input configurations, before it can make more informed decisions about which configurations to evaluate. In these scenarios, SMAC basically acts like a slow random search. RS, on the contrary, does not require any initialization and only has to perform the function evaluations, resulting in slightly better performance. Once we reach larger computational budgets, the results show that, averaged over the 20 train-test splits, SMAC consistently finds better models than RS. We can conclude that the optimization strategy is working well; despite spending around 60% of the computation time on finding which points to query, in these experiments, SMAC consistently managed to come closer to the optimum of the objective function than RS when given enough time. It is important to note here that the function evaluations are relatively fast, compared to the time it takes to optimize the acquisition function and find new input to evaluate. Therefore, the relative advantage of Bayesian optimization over random search in this setting is likely not as large as it would be for higher function evaluation costs.

Both algorithms find models that perform better on the validation set when given increasing computation time budgets. We do observe that from around 25,000 RS evaluations, increasing the runtime provides diminishing returns for both RS and SMAC and the performance of RS appears to converge, while SMAC still maintains a slight upwards trend.

When evaluating the single best found models on the unseen test set, both RS and SMAC show comparable performance, with very large 95% confidence intervals that on average range between

0.15 and 0.20 in width around the average F1-score, while the difference between the mean score of the two methods never exceeds 0.05. In these experiments, the dependence of the performance on the computation time budget is no longer apparent. While running the algorithms for longer leads to better results on the validation set, the F1-score on the test set does not improve, nor does it deteriorate. However, even the RS instance with the lowest runtime in our experiment still evaluates 1,000 different models. Since each individual classifier has its own ways to optimize their fit to the data, it is not unlikely that 1,000 evaluations — which is already much more extensive than manual search — is enough to find models that generalize well, especially considering the tendency to overfit when models become more specialized. In this case, all the found models generalize poorly and show clear signs of overfitting, with average F1-scores on the test set being up to 20% lower than on the validation set. The generalization error increases with the computation time. Clearly, the random-split cross-validation is not able to estimate the performance on unseen data accurately. Given the heterogeneity of the dataset, this is not surprising, since parts of the data can have radically different characteristics. It remains challenging to conclude from these experiments to what extent the poor generalization is caused by the way we optimize and to what extent by the inconsistencies in the data. This discussion is continued in the context of the third research question in Section 6.1.3.

6.1.2. Research Question 2: Ensemble performance comparison

As a theoretical starting point for answering this research question, we expect RS to benefit more from the creation of an ensemble than SMAC, given the inherent diversity in the models that RS returns. SMAC, on the other hand, given its local search functionality, explores promising values of the search space repeatedly, leading to more similar models with less to gain from their combination. We explore this expectation using the results on the validation set, which first of all show that there is a constant benefit in terms of performance to applying the ensemble methods to the set of models found by RS. For SMAC, this benefit decreases when the algorithm is run with a larger computational budget. Only for the smaller runtime values we see comparable results; this is in line with the earlier observation that for small computation times SMAC behaves more like a random search. Therefore, in these scenarios, SMAC benefits greatly from the different ensemble methods and achieves validation set performances comparable to RS. For larger computation times, however, the set of models returned by SMAC becomes less suitable for ensemble creation, with the performance of all methods converging to that of the individual best model. As a result, RS is able to compensate for its worse individual best model through the creation of a strong ensemble and achieve even higher average F1-scores than SMAC.

The best ensembles not only achieve higher validation scores, but also improve test scores. The generalization error, however, remains very large, and the ensembles still overfit on the validation set. To achieve good generalized performance, we want to add regularization to our models, which Top 50 and Bagging are both able to do; hence we also observe the best F1-scores on the test set from these methods. An additional advantage to creating ensembles of models is their ability to reduce the average width of the confidence intervals around the mean; all of the evaluated ensemble methods manage to achieve this, again with Top 50 and Bagging showing the best results. Top 50 likely generalizes well because it does not optimize its model selection on the validation set at all, and Bagging because it only optimizes for a random selection of the models. Overall, despite minor differences in the effect of the ensemble, the final performance on the test set remains comparable between RS and SMAC.

6.1.3. Research Question 3: RS and SMAC solution consistency comparison

The results of comparing repeated runs on the same problem provide more evidence that SMAC performs slightly better on the validation set compared to RS. Regardless of how the data were split

in training and test set or from which dataset they originated, SMAC on average always found higher F1-scores on the validation set than RS. Additionally, SMAC is more consistent in its performance, with smaller variance between runs.

From the results it is apparent that, for each of the three datasets, the variance in performance within each train-test split is much smaller than the variance in performance between the different train-test splits. Depending on the data that are part of the training set and the random-split cross-validation, the performance of the models varies. RS and SMAC are always relatively close; they seem to agree on which splits are harder to classify than others, with SMAC being able to consistently achieve slightly higher performance. This observation implies that using an average over a set of very different test splits as the final test set performance will inevitably lead to large confidence intervals around that average. The previously observed uncertainty in performance on the test set is therefore at least partly the result of the variation between the different train-test splits and not exclusively the result of the variation in the optimization performance.

This conclusion is again emphasized when we analyze the choice of classifier within the returned models. We often find specific classifiers that work well for a specific train-test split, on which RS and SMAC generally agree. Which classifier works best according to these optimization algorithms can change entirely depending on the split. This makes it much more challenging to generalize to the unseen test set, because the data the algorithms have access to do not necessarily represent the dataset as a whole. This puts the validation performance of these algorithms in perspective. Achieving optimal results on these individual splits in the data is perhaps not instrumental to achieving good performance on unseen data. By optimizing extensively on the validation set — preferring models that specialize in this specific data distribution — we are perhaps not taking the right approach to achieve that goal. Indeed, the results indicate that we might be overfitting on the validation set for all tested runtimes, even the lowest ones. This could explain the apparent indifference of the performance on the test set to the choice of algorithm and the length of computation time. The next logical conclusion, from a practical perspective, would then be to simply apply RS for a relatively small number of iterations with a simple top 50 ensemble strategy to these problems, since there is little to gain from a more complicated and more computationally demanding Bayesian optimization approach. The results of this study therefore validate the choice of RS as an optimization strategy for AutoML in radiomics.

6.2. Limitations

This section briefly mentions the limitations to this work that should be taken into account when interpreting its conclusions.

First, the configuration of SMAC used in this work has not been optimized. The sensitivity to changes in the values of the hyperparameters of SMAC itself has not been studied; the default values from the original paper were used [31]. While there is no expectation that these changes would alter any of the conclusions, different design decisions have the potential to further improve performance. Most notably, this relates to the setup of the parallel computation and the details of the initialization phase. A study regarding the optimization of SMAC's hyperparameters using SMAC itself was planned [31], but to our best knowledge, never conducted.

Next, the results were based on three similar datasets: all data used in this work were MR images that originated from many different scanners. Since AutoML is applied to many different problems, it is important to limit our conclusions on the performance of the evaluated optimization strategies to the particular radiomics setting that these datasets represent.

Finally, due to the heavy computational demands of these experiments, we have been limited in the ability to validate our results extensively. This limitation has mainly manifested itself in the reduction of the number of train-test cross-validations from 100 to 20. For the same reason, we have not extensively tested the effect of increasing the number of train-validation cross-validations from

its default value of 5.

7

Conclusion

The goal of this work was to compare the performance of two AutoML optimization strategies, the RS algorithm and the Bayesian optimization algorithm SMAC, on three medical imaging datasets.

The results show that RS optimizes slightly better than SMAC when it is run up to 10,000 evaluations, with an increase in average F1-score on the validation set of 1 to 2%. However, for longer computation times, SMAC consistently outperforms RS on the validation set, with improvements ranging from 2.6 to 3.5%, depending on the dataset. When considering the individual train-test splits when the algorithms are run repeatedly on the same data, SMAC finds higher average F1-scores compared to RS on the validation set for every single split in all three datasets, reinforcing our belief that SMAC is more effective in its ability to optimize for the F1-score on the validation set. Additionally, SMAC is more consistent in its performance, reducing the average standard deviation in the performance by 13 to 18% compared to RS.

RS and SMAC generally output the same classifiers when optimizing on the same data, but the choice varies greatly depending on the specific train-test split. Moreover, the variance in performance between different train-test splits is much larger than the variance in performance within each split. A substantial part of the uncertainty around the performance of both algorithms is therefore caused by variation in the data between the splits, rather than caused by variation in the result of the optimization.

Next, the results show that the difference in average performance of RS and SMAC on the unseen test set is minimal and statistically insignificant (for $\alpha = 0.05$). The models found during the optimization by both algorithms generalize poorly, with at least a 20% decrease in performance for each of the datasets. For both RS and SMAC, increased computation time does not lead to better solutions. The 95% confidence intervals around the average F1-scores over the 20 train-test splits are very large ($\pm$ 10 to 15% of the average F1-score), as a result of the variation in performance between individual train-test splits.

Creating an ensemble from the set of best models that were found during the optimization benefits the performance on both the validation set and the test set. On the validation set, the ensemble methods that optimize the most, such as Caruana, achieve the highest performance with improvements up to 6%. For RS, this improvement does not depend on the amount of computation time spent on selecting the models. For SMAC, the improvement degrades with more computation time, as the set of models becomes increasingly specialized. On the test set, ensemble methods Bagging and Top 50 provide the largest increase in F1-score, ranging from 4 to 7% for RS compared to the individual best model. With improvements of less than 2%, the effect is less pronounced for SMAC. When comparing the performances on the test set of the best ensemble methods for both RS and SMAC, the conclusion remains the same: RS and SMAC achieve comparable average F1-scores.

To conclude, the results of this work show that SMAC and RS return models that achieve similar performance on the unseen test sets. We have shown that a random search with relatively few evaluations and a simple ensemble strategy is sufficient to achieve performance comparable to a more sophisticated and more computationally demanding Bayesian optimization approach, therefore validating the use of a random search optimization strategy in this medical image classification setting. All found models generalize poorly to the unseen tests sets. The difficulty to generalize in this setting is furthermore emphasized by the large differences between subsets of the evaluated datasets and by the observation that increasing the computation time of the optimization does not benefit the test set performance of the final solution.

Future work could further explore this generalization problem and the suggestion made in this work that, in this setting, extensively optimizing for the F1-score on the validation set is perhaps not the best way to find models that generalize. A different AutoML strategy for radiomics could instead aim at finding a diverse set of models for an ensemble. This diversity could for example be achieved through balancing performance on different subsets of patients — perhaps identified through clustering algorithms — or through balancing performance on multiple metrics. A first step towards the latter goal could be to explicitly separate precision and recall. One potential way to achieve this is by moving to a multi-objective optimization framework [44]. The goal would then be to not return the optimal solution for a single metric, as we have done in this work, but to return a set of solutions on the Pareto front of the search space: these are solutions that cannot be improved in one of the objectives, without decreasing at least one of the other objectives [46]. This set of solutions on the Pareto front could then be promising input to an ensemble. Additionally, besides diversity, it could be beneficial to focus on reducing the complexity of the returned models [56]. For example, Olson and Moore [47] reported that they found much less complex models using their genetic programming algorithm TPOT, compared to random search. Future work could investigate whether this can work for radiomics and if it can lead to better generalizing models.

Furthermore, there is the potential to expand the search space that was defined in this work. While for this set of problems with this hyperparameter space RS and SMAC perform comparably, this might change when more hyperparameters, methods or steps in the workflow are included in the optimization. Furthermore, since this work exclusively focused on classical machine learning, there is the potential to build upon this research and extend to neural networks.

Finally, instead of focusing on the performance benefits of AutoML for radiomics, future work could look more closely into the potential it has for explainability and transparency of the returned models. For example, using comprehensive surrogate models in the Bayesian optimization has the potential to improve explainability of the results [7]. This can help to increase trust in AutoML systems [18], which is essential for adoption in clinical practice.

Bibliography

[1] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[2] Candelieri Antonio. Sequential model based optimization of partially defined functions under unknown constraints. *Journal of Global Optimization*, pages 1–23, 2019.

[3] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming*. Springer, 1998.

[4] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.

[5] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[6] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[7] Alberto Blanco-Justicia and Josep Domingo-Ferrer. Machine learning explainability through comprehensible decision trees. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 15–26. Springer, 2019.

[8] Remco R Bouckaert and Eibe Frank. Evaluating the replicability of significance tests for comparing learning algorithms. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 3–12. Springer, 2004.

[9] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[10] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[11] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18, 2004.

[12] Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. Autostacker: A compositional evolutionary learning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 402–409, 2018.

[13] Wei-Neng Chen, Jun Zhang, Henry SH Chung, Wen-Liang Zhong, Wei-Gang Wu, and Yu-Hui Shi. A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE Transactions on evolutionary computation*, 14(2):278–300, 2009.

[14] Jacob Cohen, Patricia Cohen, Stephen G West, and Leona S Aiken. *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge, 2013.

[15] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[16] Angel Díaz-Pacheco, Jesús A Gonzalez-Bernal, Carlos Alberto Reyes-García, and Hugo Jair Escalante-Balderas. Full model selection in big data. In *Mexican International Conference on Artificial Intelligence*, pages 279–289. Springer, 2017.

[17] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[18] Jaimie Drozdal, Justin Weisz, Dakuo Wang, Gaurav Dass, Bingsheng Yao, Changruo Zhao, Michael Muller, Lin Ju, and Hui Su. Trust in automl: Exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, pages 297–307, 2020.

[19] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, page 3, 2013.

[20] Hugo Jair Escalante, Manuel Montes, and Luis Enrique Sucar. Particle swarm model selection. *Journal of Machine Learning Research*, 10(Feb):405–440, 2009.

[21] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

[22] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, 2019.

[23] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970, 2015.

[24] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Practical automated machine learning for the automl challenge 2018. In *International Workshop on Automatic Machine Learning at ICML*, pages 1189–1232, 2018.

[25] Jerome H Friedman. Regularized discriminant analysis. *Journal of the American statistical association*, 84(405):165–175, 1989.

[26] Robert J Gillies, Paul E Kinahan, and Hedvig Hricak. Radiomics: images are more than pictures, they are data. *Radiology*, 278(2):563–577, 2016.

[27] Silvio B Guerra, Ricardo BC Prudêncio, and Teresa B Ludermir. Predicting the performance of learning algorithms using support vector machines as meta-regressors. In *International Conference on Artificial Neural Networks*, pages 523–532. Springer, 2008.

[28] Nikolaus Hansen. The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[29] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[30] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[31] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[32] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An evaluation of sequential model-based optimization for expensive blackbox functions. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 1209–1216, 2013.

[33] Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. Beyond manual tuning of hyperparameters. *KI-Künstliche Intelligenz*, 29(4):329–337, 2015.

[34] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning*. Springer, 2019.

[35] Kenji Kira and Larry A Rendell. The feature selection problem: Traditional methods and a new algorithm. In *Aaai*, volume 2, pages 129–134, 1992.

[36] Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, volume 9. Citeseer, 2014.

[37] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research*, 18(1):826–830, 2017.

[38] Virendra Kumar, Yuhua Gu, Satrajit Basu, Anders Berglund, Steven A Eschrich, Matthew B Schabath, Kenneth Forster, Hugo JWL Aerts, Andre Dekker, David Fenstermacher, et al. Radiomics: the process and the challenges. *Magnetic resonance imaging*, 30(9):1234–1248, 2012.

[39] Philippe Lambin, Emmanuel Rios-Velazquez, Ralph Leijenaar, Sara Carvalho, Ruud GPM Van Stiphout, Patrick Granton, Catharina ML Zegers, Robert Gillies, Ronald Boellard, André Dekker, et al. Radiomics: extracting more information from medical images using advanced feature analysis. *European journal of cancer*, 48(4):441–446, 2012.

[40] Julien-Charles Lévesque, Audrey Durand, Christian Gagné, and Robert Sabourin. Bayesian optimization for conditional hyperparameter spaces. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 286–293. IEEE, 2017.

[41] David D Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *European conference on machine learning*, pages 4–15. Springer, 1998.

[42] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, Stefan Falkner, André Biedenkapp, and Frank Hutter. Smac v3: Algorithm configuration in python. *URL https://github.com/automl/SMAC3*, 2017.

[43] Gang Luo. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5(1):18, 2016.

[44] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[45] Claude Nadeau and Yoshua Bengio. Inference for the generalization error. *Machine learning*, 52(3):239–281, 2003.

[46] Patrick Ngatchou, Anahita Zarei, and A El-Sharkawi. Pareto multi objective optimization. In *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*, pages 84–91. IEEE, 2005.

[47] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Automated Machine Learning*, pages 151–160. Springer, 2019.

[48] Randal S Olson, Nathan Bartley, Ryan J Urbanowicz, and Jason H Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 485–492, 2016.

[49] Randal S Olson, Ryan J Urbanowicz, Peter C Andrews, Nicole A Lavender, Jason H Moore, et al. Automating biomedical data science through tree-based pipeline optimization. In *European Conference on the Applications of Evolutionary Computation*, pages 123–137. Springer, 2016.

[50] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[51] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[52] M Martin Salvador, Marcin Budka, and Bogdan Gabrys. Automatic composition and optimisation of multicomponent predictive systems. *IEEE Transactions on Knowledge and Data Engineering (under review-available at http://bit. ly/automatic-mcps-paper)*, 2016.

[53] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2015.

[54] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf`.

[55] Jiangdian Song, Yanjie Yin, Hairui Wang, Zhihui Chang, Zhaoyu Liu, and Lei Cui. A review of original articles published in the emerging field of radiomics. *European Journal of Radiology*, page 108991, 2020.

[56] David J Spiegelhalter, Nicola G Best, Bradley P Carlin, and Angelika Van Der Linde. Bayesian measures of model complexity and fit. *Journal of the royal statistical society: Series b (statistical methodology)*, 64(4):583–639, 2002.

[57] Martijn Starmans, Milea JM Timbergen, Melissa Vos, Michel Renckens, Dirk J Grünhagen, Geert JLH van Leenders, Roy S Dwarkasing, François EJA Willemssen, Wiro J Niessen, Cornelis Verhoef, et al. Differential diagnosis and molecular stratification of gastrointestinal stromal tumors on ct images using a radiomics approach. *arXiv preprint arXiv:2010.06824*, 2020.

[58] Martijn PA Starmans. Workflow for optimal radiomics classification. `https://github.com/MStarmans91/WORC`, 2018.

[59] Martijn PA Starmans, Razvan L Miclea, Sebastian R van der Voort, Wiro J Niessen, Maarten G Thomeer, and Stefan Klein. Classification of malignant and benign liver tumors using a radiomics approach. In *Medical Imaging 2018: Image Processing*, volume 10574, pages 1–7. International Society for Optics and Photonics, 2018.

[60] Martijn PA Starmans, Wiro J Niessen, Ivo Schoots, Stefan Klein, Jifke F Veenland, et al. Classification of prostate cancer: high grade versus low grade using a radiomics approach. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 1319–1322. IEEE, 2019.

[61] Martijn PA Starmans, Sander R Van der Voort, Melissa Vos, et al. Fully automatic construction of optimal radiomics workflows. In *European Conference of Radiology (ECR)*, 2019.

[62] Martijn PA Starmans, Sebastian R van der Voort, Jose M Castillo Tovar, Jifke F Veenland, Stefan Klein, and Wiro J Niessen. Radiomics: data mining using quantitative medical image features. In *Handbook of Medical Image Computing and Computer Assisted Intervention*, pages 429–456. Elsevier, 2020.

[63] Xiaorui Su, Ni Chen, Huaiqiang Sun, Yanhui Liu, Xibiao Yang, Weina Wang, Simin Zhang, Qiaoyue Tan, Jingkai Su, Qiyong Gong, and Qiang Yue. Automated machine learning based on radiomics features predicts h3 k27m mutation in midline gliomas of the brain. *Neuro-oncology*, 22(3):393–401, 2020.

[64] Kevin Swersky, David Duvenaud, Jasper Snoek, Frank Hutter, and Michael A Osborne. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. *arXiv preprint arXiv:1409.4011*, 2014.

[65] Laura P Swiler, Patricia D Hough, Peter Qian, Xu Xu, Curtis Storlie, and Herbert Lee. Surrogate models for mixed discrete-continuous variables. In *Constraint Programming and Decision Making*, pages 181–202. Springer, 2014.

[66] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

[67] James H Thrall, Xiang Li, Quanzheng Li, Cinthia Cruz, Synho Do, Keith Dreyer, and James Brink. Artificial intelligence and machine learning in radiology: opportunities, challenges, pitfalls, and criteria for success. *Journal of the American College of Radiology*, 15(3):504–508, 2018.

[68] Milea JM Timbergen, Martijn PA Starmans, Guillaume A Padmos, Dirk J Grünhagen, Geert JLH van Leenders, David Hanff, Cornelis Verhoef, Wiro J Niessen, Stefan Sleijfer, Stefan Klein, et al. Differential diagnosis and mutation stratification of desmoid-type fibromatosis on mri using radiomics. *European Journal of Radiology*, page 109266, 2020.

[69] Melissa Vos, MPA Starmans, MJM Timbergen, SR van der Voort, GA Padmos, W Kessels, WJ Niessen, GJLH van Leenders, DJ Grünhagen, Stefan Sleijfer, et al. Radiomics approach to distinguish between well differentiated liposarcomas and lipomas on mri. *The British journal of surgery*, 106(13):1800, 2019.

[70] Jinhua Wang and Zeyong Yin. A ranking selection-based particle swarm optimizer for engineering design optimization problems. *Structural and multidisciplinary optimization*, 37 (2):131–147, 2008.

[71] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

[72] Jonathan Waring, Charlotta Lindvall, and Renato Umeton. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine*, page 101822, 2020.

[73] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Hu Yi-Qi, Li Yu-Feng, Tu Wei-Wei, Yang Qiang, and Yu Yang. Taking the human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

[74] Zhenwei Zhang and Ervin Sejdić. Radiological images and machine learning: trends, perspectives, and prospects. *Computers in biology and medicine*, 2019.