Automated Machine Learning

Bayesian Optimization, Meta-Learning & Applications

A DISSERTATION PRESENTED

BY

MARTIN WISTUBA

TO

THE DEPARTMENT OF COMPUTER SCIENCE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF NATURAL SCIENCES

IN THE SUBJECT OF

COMPUTER SCIENCE

UNIVERSITY OF HILDESHEIM

HILDESHEIM, LOWER SAXONY, GERMANY

APRIL 2017

Thesis advisor: Professor Dr. Dr. Lars Schmidt-Thieme Martin Wistuba

Automated Machine Learning

Bayesian Optimization, Meta-Learning & Applications

Abstract

Automating machine learning by providing techniques that autonomously find the best algorithm, hyperparameter configuration and preprocessing is helpful for both researchers and practitioners. Therefore, it is not surprising that automated machine learning has become a very interesting field of research.

Bayesian optimization has proven to be a very successful tool for automated machine learning. In the first part of the thesis we present different approaches to improve Bayesian optimization by means of transfer learning. We present three different ways of considering meta-knowledge in Bayesian optimization, i.e. search space pruning, initialization and transfer surrogate models. Finally, we present a general framework for Bayesian optimization combined with meta-learning and conduct a comparison among existing work on two different meta-data sets. A conclusion is that in particular the meta-target driven approaches provide better results. Choosing algorithm configurations based on the improvement on the meta-knowledge combined with the expected improvement yields best results.

The second part of this thesis is more application-oriented. Bayesian optimization is applied to large data sets and used as a tool to participate in machine learning challenges. We compare its autonomous performance and its performance in combination with a human expert. At two ECML-PKDD Discovery Challenges, we are able to show that automated machine learning outperforms human machine learning experts.

Finally, we present an approach that automates the process of creating an ensemble of several layers, different algorithms and hyperparameter configurations. These kinds of ensembles are jokingly called Frankenstein ensembles and proved their benefit on versatile data sets in many machine learning challenges. We compare our approach Automatic Frankensteining with the current state of the art for automated machine learning on 80 different data sets and can show that it outperforms them on the majority using the same training time. Furthermore, we compare Automatic Frankensteining on a large-scale data set to more than 3,500 machine learning expert teams and are able to outperform more than 3,000 of them within 12 CPU hours.

Doktorvater: Professor Dr. Dr. Lars Schmidt-Thieme Martin Wistuba

Automated Machine Learning

Bayesian Optimization, Meta-Learning & Applications

Zusammenfassung

Die Automatisierung des Maschinellen Lernens erlaubt es ohne menschliche Mitwirkung den besten Algorithmus, die dazugehörige beste Konfiguration und die optimale Vorverarbeitung des Datensatzes zu bestimmen und ist daher hilfreich für Anwender mit und ohne fachlichen Hintergrund. Aus diesem Grund ist es wenig überraschend, dass die Automatisierung des Maschinellen Lernens zu einem populären Forschungsgebiet aufgestiegen ist.

Bayessche Optimierung hat sich als eins der erfolgreicheren Werkzeuge für das automatisierte Maschinelle Lernen hervorgetan. Im ersten Teil dieser Arbeit werden verschiedene Methoden vorgestellt, die Bayessche Optimierung mittels Lerntransfer auch über Probleme hinweg verbessern kann. Es werden drei Möglichkeiten vorgestellt, um Wissen von zuvor adressierten Problemen auf neue zu Übertragen: Suchraumreduzierung, Initialisierung und transferierende Ersatzmodelle. Schließlich wird ein allgemeines Framework für Bayessche Optimierung beschrieben, welches existierende Metalernansätze berücksichtigt und mit schon existierenden Arbeiten auf zwei Meta-Datensätzen verglichen. Die beschriebenen Ansätze, die direkt die Meta-Zielfunktion optimieren, liefern tendenziell bessere Ergebnisse. Die Wahl der Algorithmuskonfiguration basierend auf Meta-Wissen kombiniert mit der zu erwartenen Verbesserung erweist sich als beste Methode.

Der zweite Teil der Arbeit ist anwendungsorientierter. Bayessche Optimierung wird im Rahmen von Wettbewerben auf großen Datensätzen angewandt, um Algorithmen des Maschinellen Lernens zu optimieren. Es wird sowohl die eigenständige Leistung der automatisierten Methode als auch die Leistung in Kombination mit einem menschlichen Experten bewertet. Durch die Teilnahme an zwei ECML-PKDD Wettbewerben wird gezeigt, dass das automatisierte Verfahren menschliche Konkurrenten übertreffen kann.

Abschließend wird eine Methode vorgestellt, die automatisch ein mehrschichtiges Ensemble erstellt, welches aus verschiedenen Algorithmen und entsprechenden Konfigurationen besteht . In der Vergangenheit hat sich gezeigt, dass diese Art von Ensemble die besten Vorhersagen liefern kann. Die beschriebende Methode zur automatisierten Erstellung dieser Ensemble wird mit Hilfe von 80 Datensätzen mit existierenden Konkurrenzansätzen verglichen und erreicht innerhalb derselben Zeit auf der Mehrzahl der Datensätze bessere Ergebnisse. Diese Methode wird zusätzlich mit 3.500 Teams von Experten des Maschinellen Lernens auf einem größeren Datensatz verglichen. Es zeigt sich, dass die automatisierte Methodik schon innerhalb von 12 CPU Stunden bessere Ergebnisse liefert als 3.000 der menschlichen Teilnehmer des Wettbewerbs.

Contents

Listing of figures

List of Algorithms

To the people who have made me who I am today.

Acknowledgments

Foremost, I want to thank Lars for supervising me over all these years. The fruitful environment he provided for me allowed me to follow my studies but also gave me the freedom to look beyond my own field of research.

I also want to thank Prof. Dr. Pavel Brazdil. Meeting him in person at the ECML-PKDD 2015 and 2016 and his acknowledgments of our work has been a big motivation. In particular his contributions and feedback to our journal paper and this thesis was very helpful and is very much appreciated.

I am grateful to Josif who guided me when I was lost and lacking a research topic. He introduced me to time-series classification, taught me the basics of successful academic writing and is a good friend apart from work and research. Furthermore, he supported me during the project together with Panasonic which I really appreciate.

A special gratitude goes to Nico. Besides the research topic, we have many things in common and it is fun talking with him about research, the lab's daily life and other topics.

I am grateful to my siblings Stefanie and Oliver and my parents who supported me during my entire life and have been a reliable constant.

I am also very grateful to Kathrin. She was always able to contribute to hard mathematical questions that arose over the past years.

With a special mention to Rasoul, a really nice person who introduced me to active learning and his interesting culture. To Carlotta, my office mate throughout my studies who provided me a silent and calm environment.

And finally, last but by no means least, also to everyone else at ISMLL. It was great sharing laboratory with all of you during the last four years.

Thank you all for your encouragement and the great time we had and will have.

Part I

Introduction and Basics

1 Introduction

Algorithm selection and hyperparameter optimization are omnipresent problems for researchers and practitioners. The selection of an algorithm for a specific problem and furthermore the respective hyperparameter configuration has a crucial impact on the quality of the final predictions. Algorithm selection is a well-studied problem that is not limited to machine learning but also finds application in artificial intelligence and operations research[67]. The most conventional method for selecting the algorithm is usually based on the practitioner's past experience. The hyperparameters are then usually tuned using a combination of manual search and grid or random search. This has two drawbacks. First, inexperienced researchers will have difficulties in choosing the right combination of algorithm and hyperparameter configuration. Second, finding the best hyperparameter configuration by using a grid search will be a time-consuming task. For larger data sets and more advanced algorithms, only few hyperparameter evaluations are feasible with respect to the whole search space.

Recent research proposes automatic algorithm selection and hyperparameter optimization as a solution for these problems. There are methods that need less computational time than manual or grid search and additionally find better hyperparameter configurations than human domain experts[6,56]. Recently, a program for combined algorithm selection and hyperparameter optimization was published for the well-known data mining tool Weka[75]. The current direction of research tries to mimic the optimization behavior of human experts. The information of past optimization processes is transferred to current optimization processes. This is done either by initializing the optimization process with configurations that performed well on previous experiments[58,23] or by using specific machine learning models that predict the performance of an algorithm and hyperparameter

configuration on the current problem based on previous results[4,68,87,61].

1.1 Overview

This thesis focuses on improving the current state of the art in algorithm selection and hyperparameter optimization. The problem is formally defined in Chapter 2 and the current state of the art is discussed. Chapter 3 is used to explain how we created the meta-data sets and the experimental setup. In the chapters 4 to 8 we present different approaches that accelerate the search by means of meta-knowledge. In the final chapters 10 and 11 we compare the state of the art against human machine learning experts. We discuss our results in the ECML-PKDD 2016 challenges and how to create complex ensemble methods autonomously.

1.2 Main Contributions

In this thesis we present different techniques that simulate the human behavior of hyperparameter optimization. In the following, we give an overview of these techniques.

1.2.1 Hyperparameter Search Space Pruning

Pruning techniques are a typical way of accelerating searches in general. However, pruning has not been applied to hyperparameter optimization yet. In Chapter 5 we propose to discard regions of the search space that are unlikely to contain better hyperparameter configurations. We do so by transferring knowledge from past experiments on other data sets as well as taking into account the evaluations already done on the current data set.

1.2.2 Two-Stage Transfer Surrogate Model

Bayesian optimization is a global optimization method that has been proposed for automated machine learning. One of the typical ideas of using knowledge about various problems in Bayesian optimization is the use of transfer surrogate models. Surrogate models are able to predict the loss for each configuration and are employed to select interesting configurations for evaluation. We propose a specific transfer surrogate model in Chapter 7. This surrogate consists of two stages. On the first stage, several Gaussian processes are created that reconstruct the response function for each data set. On the second stage, this meta-knowledge is combined based on the similarity between each data set to the new data set. We compare it to the state of the art and show that it provides very competitive results and easily scales to large meta-data sets.

1.2.3 Meta-Target Driven Optimization

Traditional machine learning methods such as regression or classification have natural evaluation measures such as the squared error or the classification error. A machine learning model is typically trained by minimizing a loss function which is directly derived from the given evaluation measure. The Bayesian optimization methods for hyperparameter optimization as proposed so far do not follow this principled way of minimizing the loss of interest. During my studies I developed different ways of finding this principled way.

In Chapter 4 we present the very first idea of optimizing directly for the evaluation measure. We propose to choose hyperparameter configurations based on a cost function that depends on the performance of the respective configuration on other problems. The evaluations on the new data set are only taken into account to predict the similarity between the current and former problems. This simple idea is the first step into the right direction but provides some disadvantages. Its biggest disadvantage is that the configuration candidates are limited to those used in the meta-data.

We get rid of this restriction in Chapter 6. We formalize a meta-loss for hyperparameter optimization. This loss is used to compute a meta-initialization for hyperparameter optimization methods which determines which configurations are tried first.

Our final contribution in this direction is presented in Chapter 8. The meta-loss defined in Chapter 6 is used directly within the hyperparameter configuration acquisition procedure. Thus, we achieve an effect that can be considered as a soft meta-initialization. But in comparison to a meta-initialization, the meta-knowledge about previous data sets and the new data set is considered for each hyperparameter configuration choice. The impact of the meta-knowledge about previous data sets vanishes over time since this knowledge has been exploited and only the knowledge from the new data set is used. This is an intended result that all transfer surrogates fail to achieve because they consider meta-knowledge equally, independent on the progress of the optimization process.

1.2.4 Applications

In most of our results we restrict ourselves to lab experiments. However, in Chapter 10 and 11 we conduct experiments on large data sets and compare to human expert performance.

We first explain a very elegant way of using Bayesian optimization in distributed systems with remote method invocation (RMI) in Chapter 10. Our implemented system is finally evaluated in the participation in two ECML-PKDD Discovery Challenges (European Conference on Machine Learning and Principles and Practice of Knowledge Discovery). In the Bank Card Usage Prediction Challenge, we combined Bayesian optimization with human interaction and achieved the first place. In the Network Traffic Classification Challenge, we participated without human interaction, letting

Bayesian optimization do the job for us and placed third, outperforming many human competitors.

In Chapter 11 we describe a way of creating complex ensembles autonomously by using Bayesian optimization as the core optimization method. We compare our performance to the state of the art in automated machine learning on 80 UCI data sets and compare to more than 3,500 human machine learning experts by participating in one Kaggle challenge. In an extensive evaluation we can show that we outperform the state of the art in automated machine learning and most human machine learning experts.

1.3 Published Works

The different chapters are mostly based on published peer-reviewed work[82,79,81,84,78,83,85].

Chapter 4 Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2015). Sequential model-free hyperparameter tuning. In *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015* (pp. 1033-1038).

Chapter 5 Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2015). Hyperparameter search space pruning - A new component for sequential model-based hyperparameter optimization. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML-PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II* (pp. 104-119).

Chapter 6 Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2015). Learning hyperparameter optimization initializations. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Campus des Cordeliers, Paris, France, October 19-21, 2015* (pp. 1-10).

Chapter 7 Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2016). Two-stage transfer surrogate model for automatic hyperparameter optimization. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML-PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I* (pp. 199-214).

Chapter 8 Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2016). Hyperparameter optimization machines. In *2016 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2016, Montreal, QC, Canada, October 17-19, 2016* (pp. 41-50).

Chapter 10 Wistuba, M., Duong-Trung N., Schilling, N., & Schmidt-Thieme, L. (2016). Bank card usage prediction exploiting geolocation information. *CoRR*, abs/1610.03996.

Chapter 11 Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2017). Automatic Frankensteining: Creating complex ensembles autonomously. Accepted at the *2017 SIAM International Conference on Data Mining, SDM 2017, Houston, Texas, USA, April 27-29, 2017*.

2

Problem Definition & Related Work

In this chapter, we formally define the problem addressed in this thesis and the notation used. We review how configurations of algorithms are typically optimized in machine learning and recent progresses achieved with Bayesian optimization.

2.1 Problem Definition

When tackling a machine learning problem, let us say classification, a machine learning expert has to make many decisions. One aspect of her decision process is the selection of the algorithm and its parameters. Unfortunately, most algorithms have many parameters that need to be defined by the machine learning expert before training the classifier. To distinguish them from model parameters, which are estimated during the learning procedure, we call them *hyperparameters*. Most hyperparameters define the model complexity (e.g. number of nodes in a neural network, depth of a tree, regularization parameters) or have influence on the learning procedure (e.g. learning rate, momentum, number of iterations). These specific parameters have high impact on how good an algorithm performs. Hence, the algorithm and its hyperparameter need to be chosen in combination. This thesis will present methods that can autonomously find the right combination of algorithm and its configuration.

In the following, we formally define the problem of *configuration optimization* and the notation used throughout the thesis. For notational convenience we assume that there is only one possible learning task applicable for a data set. We denote the space of all data sets as $\mathcal{D}$ and the space of all

models as $\mathcal{M}$. $\mathcal{X}$ is the space of all configurations. Then, we define a general learning algorithm $\mathcal{A}$ as a mapping

$$\mathcal{A} : \mathcal{D} \times \mathcal{X} \rightarrow \mathcal{M} \ . \tag{2.1}$$

The configuration $\mathbf{x} \in \mathcal{X}$ encodes the configuration, i.e. it defines which machine learning algorithm and the hyperparameter configuration is selected. Further properties that might be included in $\mathcal{X}$ are preprocessing, feature selection or feature engineering.

Given a data set $D \in \mathcal{D}$, which is partitioned into D_{train} and D_{valid}, and a configuration $\mathbf{x} \in \mathcal{X}$, the general learning algorithm $\mathcal{A}$ estimates a prediction model $M \in \mathcal{M}(\mathbf{x})$. This model is estimated by minimizing a loss function $\mathcal{L}$ (e.g. residual sum of squares) which is penalized with a regularization term $\mathcal{R}$ (e.g. Tikhonov regularization) with respect to the training data D_{train}. That is,

$$\mathcal{A}(D, \mathbf{x}) = \underset{M \in \mathcal{M}(\mathbf{x})}{\arg\min}\, \mathcal{L}(M, D_{\text{train}}) + \mathcal{R}(M) \ . \tag{2.2}$$

The task in this thesis is to find the configuration $\mathbf{x}^*$ that leads to a prediction model which minimizes the loss on the validation partition D_{valid}. Formally,

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}}{\arg\min}\ \mathcal{L}(\mathcal{A}(D_{\text{train}}, \mathbf{x}), D_{\text{valid}}) = \underset{\mathbf{x} \in \mathcal{X}}{\arg\min}\ f_D(\mathbf{x}) \ . \tag{2.3}$$

The function $f_D : \mathcal{X} \rightarrow \mathbb{R}$ with

$$f_D(\mathbf{x}) = \mathcal{L}(\mathcal{A}(D_{\text{train}}, \mathbf{x}), D_{\text{valid}}) \tag{2.4}$$

is the *response function* of data set D. In many applications, the configuration space $\mathcal{X}$ equals the hyperparameter space of a single algorithm and hence, we call this task also *hyperparameter optimization* and the configurations hyperparameter configurations.

For the sake of demonstration, we consider the problem of optimizing the hyperparameters of classifiers in this thesis. Thus, the response function f_D maps a configuration to the classification error. This is no limitation, but shall help the reader to understand the concepts.

2.2 Standard Techniques for Configuration Optimization

Evaluating the response function f_D at a single point involves training a machine learning model which is a time-consuming task. Hence, the minimization of f_D cannot be achieved with standard optimization techniques. A very common technique is grid search. Given the configuration space,

$$\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_P \ , \tag{2.5}$$

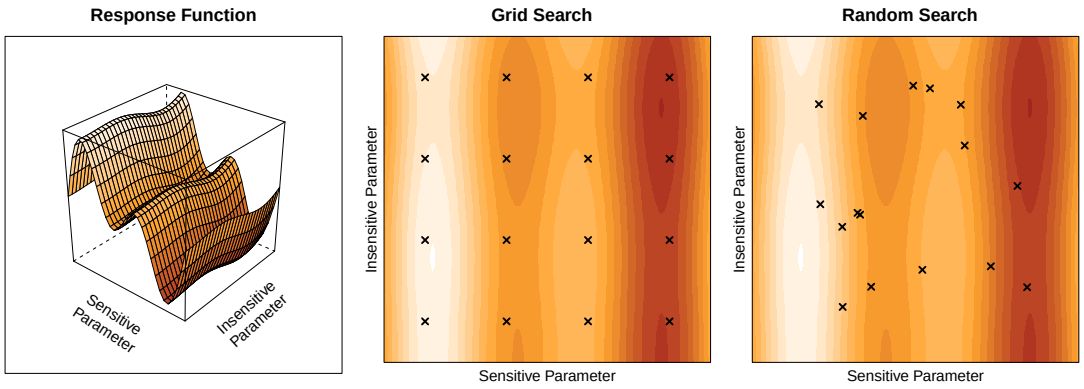

Figure 2.1: Random search works better than grid search for problems with low effective configuration dimensionality.

a set of k_i values is chosen for each dimension.

$$G_i = \{\mathbf{x}_1, \ldots, \mathbf{x}_{k_i}\}\,,\ \mathbf{x}_1, \ldots, \mathbf{x}_{k_i} \in \mathcal{X}_i \tag{2.6}$$

Then, all configurations in

$$G = G_1 \times \ldots \times G_P \tag{2.7}$$

are evaluated and the best performing is selected.

An alternative to grid search is random search[7]. Similarly to grid search, the upper and lower bounds per dimension need to be defined. Then, configurations $\mathbf{x}$ are sampled uniformly at random according to

$$\mathbf{x}_i \sim \mathcal{U}\left(b_i^{\min}, b_i^{\max}\right) \quad , \tag{2.8}$$

where $b_i^{\min}$ and $b_i^{\max}$ are the lower and upper bound in the i-th dimension, respectively. This method works well, in particular if the algorithm has a low effective configuration dimensionality. The reason for this is that in many problems some dimensions are insensitive to changes.

It try to explain this at the example given in Figure 2.1. It shows a two-dimensional response function where only one dimension is sensitive to changes. Thus, the effective dimensionality is just one. Applying a grid search will lead to many redundant function evaluations. In fact, only four different values of the sensitive parameter are tested. Otherwise, random search efficiently uses every evaluation and tests sixteen different values of the sensitive parameter.

Both, grid and random search, are stateless optimization techniques which do not take previous evaluations of f into account. Bayesian optimization[53] can be used to overcome this disadvantage. Considering the choice of algorithm configurations as a black-box global optimization problem[37] as defined in Equation 2.3, Bayesian optimization can be used for finding optimal configurations automatically[68].

Bayesian optimization consists of two components, a surrogate model and an acquisition function. We collect all evaluations of f in the *observation history*

$$\mathcal{H} = \{(\mathbf{x}_1, f(\mathbf{x}_1)), (\mathbf{x}_2, f(\mathbf{x}_2)), \ldots\} \quad . \tag{2.9}$$

A surrogate model provides a distribution $p(f_*|\mathbf{x}_*, \mathcal{H})$ over response function values $f_* \in \mathbb{R}$ given a configuration $\mathbf{x}_* \in \mathcal{X}$ and for a given observation history $\mathcal{H} \in \mathcal{X} \times \mathbb{R}$. We assume that the predictive posterior distribution of a surrogate model is Gaussian distributed with mean $m_{f|\mathcal{H}}$ and covariance $\mathbf{\Sigma}_{f|\mathcal{H}}$.

$$p(f_*|\mathbf{x}_*, \mathcal{H}) = \mathcal{N}\left(f_*|m_{f|\mathcal{H}}(\mathbf{x}_*), \mathbf{\Sigma}_{f|\mathcal{H}}(\mathbf{x}_*, \mathbf{x}_*)\right) \tag{2.10}$$

The acquisition function a evaluates configurations based on their expected utility. Given a utility function u_f, the acquisition is determined by

$$\begin{aligned} a(\mathbf{x}_*, p(f_*|\mathbf{x}_*, \mathcal{H})) &= E\left[u_f(\mathbf{x}_*)|\mathbf{x}_*, \mathcal{H}\right] && (2.11)\\ &= \int u_f(\mathbf{x}_*)\, p(f_*|\mathbf{x}_*, \mathcal{H})\, \mathrm{d}f_* \quad . && (2.12) \end{aligned}$$

The configuration with highest expected utility

$$\mathbf{x} = \underset{\mathbf{x}_* \in \mathcal{X}}{\arg\max}\, a(\mathbf{x}_*, p(f_*|\mathbf{x}_*, \mathcal{H})) \tag{2.13}$$

is evaluated next. While the acquisition function introduces a further optimization problem, the evaluation of a is much faster than the evaluation of f.

Algorithm 1 outlines Bayesian optimization for minimizing the function f and Figure 2.2 visualizes the optimization process. In each iteration, f is approximated by the surrogate model using the observation history $\mathcal{H}$. The acquisition function a finds a trade-off between exploitation and exploration and determines the next configuration $\mathbf{x}_*$. This configuration $\mathbf{x}_*$ is evaluated and the new observation is added to the observation history $\mathcal{H}$. After a convergence criterion is met, the best performing configuration is returned. Possible convergence criteria are a time budget or that

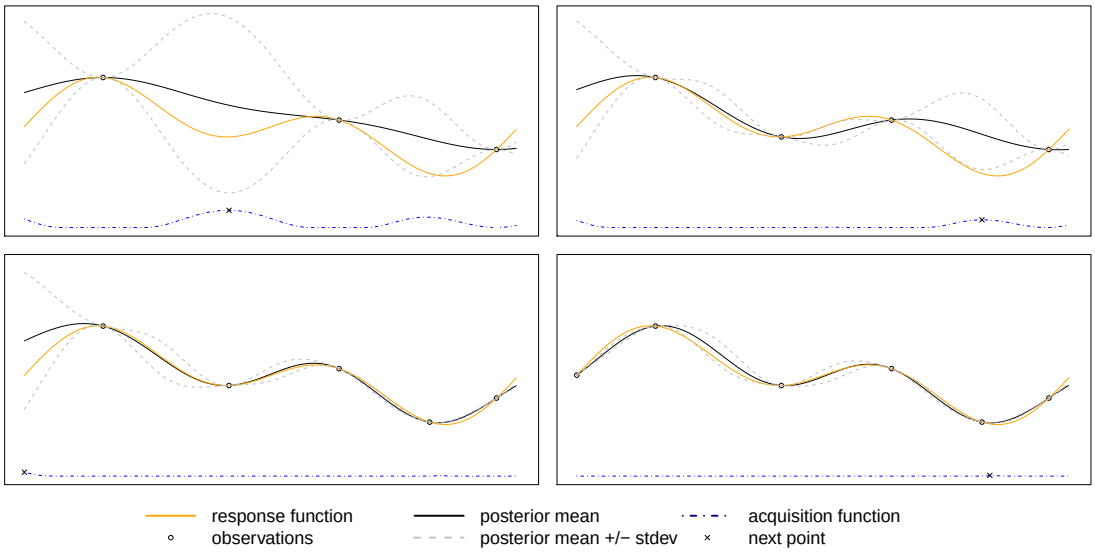

Figure 2.2: In this example, we demonstrate Bayesian optimization by minimizing the function $f(x)=sin(x)+sin(2x)$. At the bottom the acquisition function is plotted and the cross ($\times$) indicates its global maximum. The predictive posterior distribution is visualized by plotting the mean and standard deviation. Starting with three observations ($\circ$), the response function is approximated. The value with highest expected utility (in this case expected improvement) is selected for the next evaluation. This process is continued and as we see, we find values close to the optimum quickly.

the highest score of the acquisition function is below a threshold ε, i.e.

$$\max_{\mathbf{x}_* \in \mathcal{X}} a\left(\mathbf{x}_*, p\left(f_* | \mathbf{x}_*, \mathcal{H}\right)\right) < \varepsilon \quad . \tag{2.14}$$

Different acquisition functions and surrogate models have been proposed. We will review them in the next sections. We recommend the recent review by Shahriari et al.[65] as an alternative source for information about Bayesian optimization.

2.4 Meta-Learning

A large part of this thesis is focused on accelerating Bayesian optimization by means of meta-learning. Meta-learning dates back to the 1970s[60] and can be considered as an alternative to Bayesian optimization because a considerable amount of work on meta-learning focuses on recommending configurations for algorithms. No consensus on the definition of meta-learning has been reached. Vilalta and Drissi[76] define meta-learning as follows:

> "Meta-learning studies how learning systems can increase in efficiency through experience; the goal is to understand how learning itself can become flexible according to the domain or task under study."

Algorithm 1 Bayesian Optimization

Input: Configuration space $\mathcal{X}$, observation history $\mathcal{H}$, acquisition function a.
Output: Best configuration found.

```
while not converged do
    Update the surrogate model p(f_*|x_*, H).
    x_* ← arg max_{x_* ∈ X} a(x_*, p(f_*|x_*, H))
    f_* ← f(x_*)
    H ← H ∪ {(x_*, f_*)}
    if f_* < f^min then
        x^min, f^min ← x_*, f_*
    end if
end while
return x^min
```

A similar definition is given by Brazdil et al.[8]:

> "Meta-learning is the study of principled methods that exploit meta-knowledge to obtain efficient models and solutions by adapting machine learning and data mining processes."

The idea of meta-learning for configuration recommendation is based on a simple assumption. Algorithms show similar performance for the same configuration for similar problems.

In this thesis we will combine the ideas of meta-learning with Bayesian optimization. This accelerates the search for good configurations. One way of doing this is by using a meta-initialization. For example, consider configurations that have been good on similar problems will be evaluated first. We discuss this idea in Chapter 6. Another idea is the use of transfer surrogate models. These surrogate models do not only learn from observations of the response function of the current problem but learn across problems. This idea is explained in detail in Section 2.7. Further ways of using meta-learning are presented in the upcoming chapters.

In this thesis the term meta-knowledge is used frequently. We define it as the common knowledge about response functions and data sets. This includes meta-features and response function evaluations for different data sets. Meta-features are data set descriptors. One example is the number of instances in a data set, further examples are given in Section 3.1. Typically, a data scientist has applied the same algorithm for different data sets. Assuming, she gathered observations from M many data sets, she collected the meta-knowledge

$$\mathcal{H} = \{(\mathbf{x}_1, f_{D_1}(\mathbf{x}_1)), \ldots, (\mathbf{x}_1, f_{D_1}(\mathbf{x}_{N_1})), \ldots, (\mathbf{x}_1, f_{D_M}(\mathbf{x}_1)), \ldots, (\mathbf{x}_1, f_{D_M}(\mathbf{x}_{N_M}))\} \quad . \tag{2.15}$$

In the following, we use the prefix *meta* to distinguish between the different levels of machine learning problems. The traditional machine learning problem is to learn some parameters $\boldsymbol{\theta}$ on a given data set containing instances with predictors. For the configuration optimization problem, we create meta-data sets consisting of meta-instances with meta-predictors. A meta-data set contains meta-instances $(\mathbf{x}_i, f_D(\mathbf{x}_i))$ where $f_D(\mathbf{x}_i)$ is the target and $\mathbf{x}_i$ are the predictors. These meta-predictors can be enriched by meta-features.

2.5 Acquisition Functions

As discussed earlier, finding optimal configurations for machine learning algorithms can be considered a global optimization of a function f where f is an expensive black-box function. Given a set of observations $\mathcal{H}$, a surrogate model can be computed that provides a distribution over f. The important question is now, how to decide which point of the function to evaluate next. This is done by the acquisition function a that scores for every point how desirable its evaluation for our minimization problem is. Hence, another optimization problem, the maximization of a, is introduced. Fortunately, the function a is much cheaper to evaluate than f.

Many different acquisition functions have been proposed to evaluate the expected value of f for a specific argument. Acquisition functions can be categorized into three different classes. Improvement-based policies such as probability of improvement[42] and expected improvement[53] consider the currently best observation in their decision. Information-based policies aim at reducing the entropy of the posterior distribution around the optimal value

$$\mathbf{x}^{\min} = \arg\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) \quad . \tag{2.16}$$

Examples are Thompson sampling[74] and entropy search[77,31,32]. The idea of optimistic policies is to minimize the regret during Bayesian optimization. There are various representatives for these policies[16,70,52]. Furthermore, a combination of different acquisition functions have been proposed[34]. We will review some of the more prominent acquisition functions in the following.

2.5.1 Probability of Improvement

Given a function f to minimize and the observation history $\mathcal{H} = (\mathbf{X}, \mathbf{f})$, the best value of f observed so far is

$$f^{\min} = \min \mathbf{f} \ . \tag{2.17}$$

The acquisition function called probability of improvement[42] estimates the probability that the value of f for the configuration $\mathbf{x}_*$ is better than currently best value $f^{\min}$. Hence, it is based on a

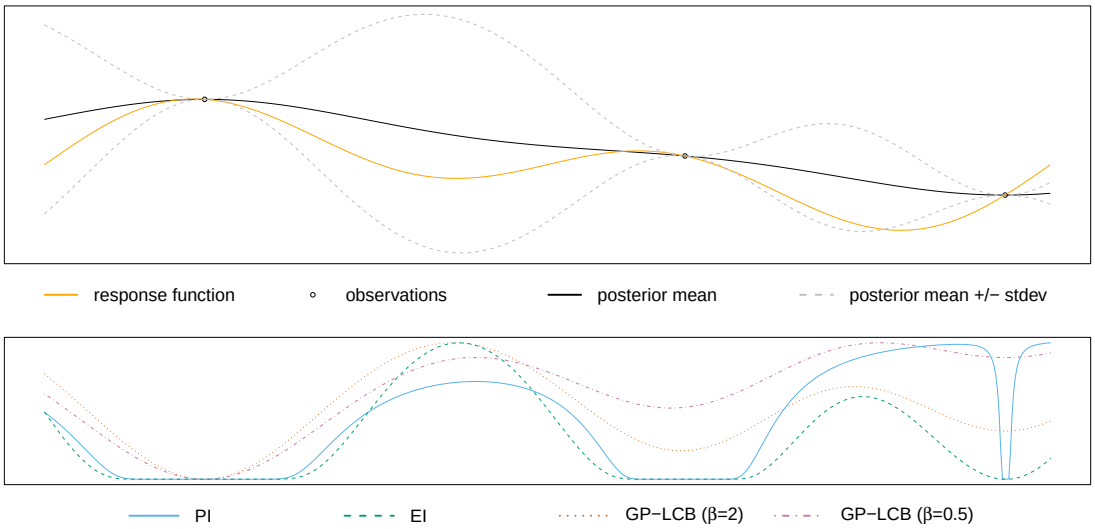

Figure 2.3: Different acquisition functions with different settings are compared on our previous example. All acquisition functions have maximums in the same areas but the global maximum might be different.

utility function that is 1 if the value of f for the configuration $\mathbf{x}_*$ is better than currently best value $f^{\min}$ and 0 otherwise. Formally,

$$u_f(\mathbf{x}_*) = \begin{cases} 1 & f(\mathbf{x}_*) < f^{\min} \\ 0 & \text{otherwise} \end{cases}. \tag{2.18}$$

As one can see, the utility function of the probability of improvement does not depend on how much $f(\mathbf{x}_*)$ improves over $f^{\min}$. This leads to a greedy behavior which can be seen in Figure 2.3, where we compare this approach to other acquisition functions. Following the definition in Equation 2.13, probability of improvement can be derived as follows

$$\begin{aligned} a_{\text{PI}}\left(\mathbf{x}_*, p\left(f_*|\mathbf{x}_*, \mathcal{H}\right)\right) &:= E\left[u_f(\mathbf{x}_*)\,|\mathbf{x}_*, \mathcal{H}\right] && (2.19)\\ &= \int u_f(\mathbf{x}_*)\, p\left(f_*|\mathbf{x}_*, \mathcal{H}\right) \mathrm{d}f_* && (2.20)\\ &= \int_{-\infty}^{f^{\min}} \mathcal{N}\left(f_*|m_{f|\mathcal{H}}\left(\mathbf{x}_*\right), \Sigma_{f|\mathcal{H}}\left(\mathbf{x}_*, \mathbf{x}_*\right)\right) \mathrm{d}f_* && (2.21)\\ &= \Phi\left(f^{\min}|m_{f|\mathcal{H}}\left(\mathbf{x}_*\right), \Sigma_{f|\mathcal{H}}\left(\mathbf{x}_*, \mathbf{x}_*\right)\right) && (2.22) \end{aligned}$$

where $\mathcal{N}\left(\cdot|m, \sigma\right)$ and $\Phi\left(\cdot|m, \sigma\right)$ denote the general normal distribution and cumulative distribution function with mean m and standard deviation σ, respectively. The predictive posterior distri-

bution of the surrogate model $p\left(f_*|\mathbf{x}_*, \mathcal{H}\right)$ is defined as

$$p\left(f_*|\mathbf{x}_*, \mathcal{H}\right) = \mathcal{N}\left(f_*|m_{f|\mathcal{H}}\left(\mathbf{x}_*\right), \Sigma_{f|\mathcal{H}}\left(\mathbf{x}_*, \mathbf{x}_*\right)\right) \quad . \tag{2.23}$$

2.5.2 Expected Improvement

The expected improvement is the most prominent choice for hyperparameter optimization. Snoek et al.[68] provided experiments that showed that is performing best for our task of configuration optimization. For these two reasons we used this acquisition function in all our experiments.

The difference to the probability of improvement is small but important. Expected improvement considers how much a configuration likely improves over the currently best solution. Formally, the utility for a configuration $\mathbf{x}_* \in \mathcal{X}$ is defined as

$$u_f(\mathbf{x}_*) = \max\left\{f^{\min} - f(\mathbf{x}_*), 0\right\} \quad . \tag{2.24}$$

Given this utility, the expected improvement for a configuration $\mathbf{x}_*$ is defined as

$$\begin{aligned}
a_{\mathrm{EI}}\left(\mathbf{x}_*, p\left(f_*|\mathbf{x}_*, \mathcal{H}\right)\right) &:= E\left[u_f(\mathbf{x}_*)\,|\mathbf{x}_*, \mathcal{H}\right] && (2.25)\\
&= \int u_f(\mathbf{x}_*)\, p\left(f_*|\mathbf{x}_*, \mathcal{H}\right) \mathrm{d}f_* && (2.26)\\
&= \int_{-\infty}^{f^{\min}} \left(f^{\min} - f_*\right) \mathcal{N}\left(f_*|m_{f|\mathcal{H}}\left(\mathbf{x}_*\right), \Sigma_{f|\mathcal{H}}\left(\mathbf{x}_*, \mathbf{x}_*\right)\right) \mathrm{d}f_* && (2.27)\\
&= \left(f^{\min} - m_{f|\mathcal{H}}\left(\mathbf{x}_*\right)\right) \Phi\left(f^{\min}|m_{f|\mathcal{H}}\left(\mathbf{x}_*\right), \Sigma_{f|\mathcal{H}}\left(\mathbf{x}_*, \mathbf{x}_*\right)\right) \\
&\quad + \sqrt{\Sigma_{f|\mathcal{H}}\left(\mathbf{x}_*, \mathbf{x}_*\right)} \mathcal{N}\left(f^{\min}|m_{f|\mathcal{H}}\left(\mathbf{x}_*\right), \Sigma_{f|\mathcal{H}}\left(\mathbf{x}_*, \mathbf{x}_*\right)\right) \quad . && (2.28)
\end{aligned}$$

Proof. In the following, we will prove the step from Equation (2.27) to Equation (2.28). For notational convenience, we will use $\mu = m_{f|\mathcal{H}}\left(\mathbf{x}_*\right)$ and $\sigma^2 = \Sigma_{f|\mathcal{H}}\left(\mathbf{x}_*, \mathbf{x}_*\right)$.

$$\begin{aligned}
&\int_{-\infty}^{f^{\min}} \left(f^{\min} - f_*\right) \mathcal{N}\left(f_*|\mu, \sigma^2\right) \mathrm{d}f_* && (2.29)\\
= &\int_{-\infty}^{f^{\min}} f^{\min} \mathcal{N}\left(f_*|\mu, \sigma^2\right) \mathrm{d}f_* - \int_{-\infty}^{f^{\min}} f_* \mathcal{N}\left(f_*|\mu, \sigma^2\right) \mathrm{d}f_* && (2.30)\\
= &\; f^{\min} \Phi\left(f^{\min}|\mu, \sigma^2\right) - \int_{-\infty}^{f^{\min}} f_* \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(f_* - \mu\right)^2}{2\sigma^2}\right) \mathrm{d}f_* \quad . && (2.31)
\end{aligned}$$

We can rewrite

$$\int_{-\infty}^{f^{\min}} f_* \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(f_*-\mu)^2}{2\sigma^2}\right) \mathrm{d}f_* \tag{2.32}$$

$$= \int_{-\infty}^{f^{\min}} (f_*-\mu) \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(f_*-\mu)^2}{2\sigma^2}\right) \mathrm{d}f_*$$

$$+ \int_{-\infty}^{f^{\min}} \mu \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(f_*-\mu)^2}{2\sigma^2}\right) \mathrm{d}f_* \tag{2.33}$$

$$= \int_{-\infty}^{f^{\min}} \frac{(f_*-\mu)}{\sigma} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(f_*-\mu)^2}{2\sigma^2}\right) \mathrm{d}f_* + \mu\Phi\left(f^{\min}|\mu,\sigma^2\right) \quad . \tag{2.34}$$

Then, by substituting $t = \frac{f_*-\mu}{\sigma}$, we derive

$$\int_{-\infty}^{f^{\min}} \frac{(f_*-\mu)}{\sigma} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(f_*-\mu)^2}{2\sigma^2}\right) \mathrm{d}f_* \tag{2.35}$$

$$= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{f^{\min}-\mu}{\sigma}} t \cdot \exp\left(-\frac{t^2}{2}\right) \mathrm{d}t \tag{2.36}$$

$$= \frac{1}{2\cdot\sqrt{2\pi}} \int_{-\infty}^{\frac{f^{\min}-\mu}{\sigma}} 2t \cdot \exp\left(-\frac{t^2}{2}\right) \mathrm{d}t \quad . \tag{2.37}$$

Defining $g(t) = e^{-t/2}$ and $h(t) = t^2$, we get

$$\frac{1}{2\sqrt{2\pi}} \int_{-\infty}^{\frac{f^{\min}-\mu}{\sigma}} \left(\frac{d}{dt}h(t)\right) g(h(t)) \, \mathrm{d}t \quad . \tag{2.38}$$

Integrating by substitution, we obtain

$$\frac{1}{2\sqrt{2\pi}} \int_{\left(\frac{f^{\min}-\mu}{\sigma}\right)^2}^{\infty} g(u) \, \mathrm{d}u = \frac{1}{2\sqrt{2\pi}} \int_{\left(\frac{f^{\min}-\mu}{\sigma}\right)^2}^{\infty} e^{-u/2} \, \mathrm{d}u \tag{2.39}$$

$$= \frac{1}{2\sqrt{2\pi}} \left[-2e^{-u/2}\right]_{\left(\frac{f^{\min}-\mu}{\sigma}\right)^2}^{\infty} \tag{2.40}$$

$$= \frac{\sigma}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(f^{\min}-\mu\right)^2}{2\sigma^2}\right) \tag{2.41}$$

$$= \sigma\mathcal{N}\left(f^{\min}|\mu,\sigma^2\right) \quad . \tag{2.42}$$

Finally, we can conclude

$$\int_{-\infty}^{f^{\min}} \left(f^{\min} - f_*\right) \mathcal{N}\left(f_* | \mu, \sigma^2\right) \mathrm{d}f_* \tag{2.43}$$

$$= f^{\min} \Phi\left(f^{\min} | \mu, \sigma^2\right) - \left(\sigma \mathcal{N}\left(f^{\min} | \mu, \sigma^2\right) + \mu \Phi\left(f^{\min} | \mu, \sigma^2\right)\right) \tag{2.44}$$

$$= \left(f^{\min} - \mu\right) \Phi\left(f^{\min} | \mu, \sigma^2\right) - \sigma \mathcal{N}\left(f^{\min} | \mu, \sigma^2\right) \quad . \tag{2.45}$$

□

The utility function in Equation (2.24) is called improvement and in a similar way applied by other approaches in meta-learning. Leite et al. make use of relative landmarks which estimate the improvement of one configuration over another based on observations on previous problems[47]. We employ this related improvement also in chapters 4 and 8.

2.5.3 Entropy Search

The information-based acquisition functions are inspired by techniques proposed in active learning[49]. The idea is to maximize the information about the location at the current optimal configuration

$$\mathbf{x}^{\min} = \arg\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) \quad . \tag{2.46}$$

The utility of a configuration is measured in terms of the change of entropy for the posterior distribution $p\left(\mathbf{x}^{\min} | \mathcal{H}\right)$. Thus, the utility function is defined as

$$u_f(\mathbf{x}_*) = h\left[\mathbf{x}^{\min} | \mathcal{H}\right] - h\left[\mathbf{x}^{\min} | \mathcal{H} \cup \{(\mathbf{x}_*, f(\mathbf{x}_*))\}\right] \quad , \tag{2.47}$$

where $h[\mathbf{x}]$ is the continuous entropy

$$h[\mathbf{x}] = -\int p(\mathbf{x}) \ln p(\mathbf{x}) \, \mathrm{d}\mathbf{x} \quad . \tag{2.48}$$

The higher the information gain after adding the observation for configuration $\mathbf{x}_*$, the higher the values of the utility function. The acquisition function is then the expected information gain when choosing a configuration $\mathbf{x}_*$:

$$a_{\mathrm{ES}}(\mathbf{x}_*) := h\left[\mathbf{x}^{\min} | \mathcal{H}\right] - \int h\left[\mathbf{x}^{\min} | \mathcal{H} \cup \{(\mathbf{x}_*, f_*)\}\right] p\left(f_* | \mathbf{x}_*, \mathcal{H}\right) \mathrm{d}f_* \quad . \tag{2.49}$$

Computing the expected value of the continuous entropy and computing the continuous entropies itself are intractable in practice. Hence, different approximations have been proposed[77,31,32].

2.5.4 Lower/Upper Confidence Bounds

The GP-UCB acquisition function[16,70] is based on the seminal work by Lai and Robbins on the multi-armed bandit problem[44]. The idea is to be very optimistic about the outcome, always expecting the best case scenario. The acquisition function was introduced for function maximization, hence, upper confidence bounds are used

$$a_{\text{GP-UCB}}\left(\mathbf{x}_*, p\left(f_*|\mathbf{x}_*, \mathcal{H}\right)\right) := m_{f|\mathcal{H}}\left(\mathbf{x}_*\right) + \beta_t \sqrt{\Sigma_{f|\mathcal{H}}\left(\mathbf{x}_*, \mathbf{x}_*\right)} \ , \tag{2.50}$$

where β_t is used to balance exploitation and exploration at step t. In case of function minimization, the acquisition function can be redefined by using lower confidence bounds:

$$a_{\text{GP-LCB}}\left(\mathbf{x}_*, p\left(f_*|\mathbf{x}_*, \mathcal{H}\right)\right) := -m_{f|\mathcal{H}}\left(\mathbf{x}_*\right) + \beta_t \sqrt{\Sigma_{f|\mathcal{H}}\left(\mathbf{x}_*, \mathbf{x}_*\right)} \ . \tag{2.51}$$

In contrast to the previous acquisition functions, the GP-UCB/LCB acquisition function requires the optimization of the hyperparameter β_t. It has been proven that with high probability GP-UCB has no regret for specific β_t[70].

2.6 Gaussian Processes

Gaussian processes are the most dominant surrogate model for Bayesian optimization and hence we recapture its definition. For more information, we refer the interested reader to the excellent book by Rasmussen and Williams, entitled Gaussian Processes for Machine Learning[57].

The regression problem is to model relationships between one dependent variable f_i and multiple independent variables $\mathbf{x}_i$. This relationship is assumed to be described by a latent function

$$f\colon \mathcal{X} \to \mathbb{R} \ . \tag{2.52}$$

The function f is unknown but noisy observations $(\mathbf{x}_i, f_i)$ of this functions are available. The relationship to f can be explained by decomposing it into a signal and a noise part

$$f_i = f(\mathbf{x}_i) + e\left(\mathbf{x}_i\right) \ , \tag{2.53}$$

where e is the stochastic error. In the context of machine learning, the set of pairs of observations is

also called training data.

$$\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N) \tag{2.54}$$
$$\mathbf{f} = (f_1, \ldots, f_N) \tag{2.55}$$

Gaussian process models assume a Gaussian prior on f. Every observation f_i is considered to be a random variable and the joint distribution of all f_i is assumed to be multivariate Gaussian distributed:

$$p(\mathbf{f}|\mathbf{X}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{f}|m(\mathbf{X}), k(\mathbf{X}, \mathbf{X})) \quad . \tag{2.56}$$

A Gaussian process is completely specified by its mean function m and its covariance function k and possibly depends on some parameters $\boldsymbol{\theta}$. In order to predict the labels $\mathbf{f}_*$ for some test instances $\mathbf{x}_*$, the Gaussian process assumption is that $\mathbf{f}$ and $\mathbf{f}_*$ are jointly Gaussian

$$p(\mathbf{f}, \mathbf{f}_*|\mathbf{X}, \mathbf{X}_*, \boldsymbol{\theta}) = \mathcal{N}\left(\begin{pmatrix} \mathbf{f} \\ \mathbf{f}_* \end{pmatrix} \middle| \begin{pmatrix} m(\mathbf{X}) \\ m(\mathbf{X}_*) \end{pmatrix}, \begin{pmatrix} \mathbf{K}_n & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix} \right) \quad , \tag{2.57}$$

where

$$\mathbf{K}_n = \mathbf{K} + \sigma_n^2 \mathbf{I} \tag{2.58}$$
$$\mathbf{K}_* = k(\mathbf{X}, \mathbf{X}_*) \tag{2.59}$$
$$\mathbf{K}_{**} = k(\mathbf{X}_*, \mathbf{X}_*) \tag{2.60}$$

for brevity. In this notation, σ_n^2 is a noise hyperparameter added to the diagonal of the kernel matrix $\mathbf{K} = k(\mathbf{X}, \mathbf{X})$. The predictive posterior distribution can be obtained from the joint distribution.

$$p(\mathbf{f}_*|\mathbf{X}, \mathbf{f}, \mathbf{X}_*, \boldsymbol{\theta}) = \mathcal{N}\left(\mathbf{f}_*|m(\mathbf{X}_*) + \mathbf{K}_*^T \mathbf{K}^{-1}(\mathbf{f} - m(\mathbf{X})), \mathbf{K}_{**} - \mathbf{K}^T \mathbf{K}^{-1} \mathbf{K}_*\right) \tag{2.61}$$

From now on we assume $m(\mathbf{x}) = 0$ to simplify the notation.

A typical covariance function used is the squared exponential kernel with automatic relevance determination

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2} \sum_{p=1}^{\dim \mathcal{X}} \frac{\left(\mathbf{x}_{i,p} - \mathbf{x}_{j,p}\right)^2}{\boldsymbol{\sigma}_p^2} \right) \quad . \tag{2.62}$$

Kernel hyperparameters $\boldsymbol{\theta}$ can be estimated and optimized by maximizing the log marginal likeli-

hood on the training data which is given by

$$\log p\left(\mathbf{f}|\mathbf{X},\boldsymbol{\theta}\right) = -\frac{1}{2}\mathbf{f}^T\mathbf{K}_n^{-1}\mathbf{f} - \frac{1}{2}\log|\mathbf{K}_n| - \frac{N}{2}\log\left(2\pi\right) \quad . \tag{2.63}$$

Bayesian optimization requires frequent updates of the Gaussian process. Retraining it completely is computationally expensive and dominated by the inversion of the kernel matrix which is cubic in the number of training instances. Using a little trick, the update can be reduced to squared run time complexity. The kernel matrix is decomposed using the Cholesky decomposition $\mathbf{K}_n = \mathbf{L}\mathbf{L}^T$. Then, the predicted probability distribution for a single instance $\mathbf{x}_*$ is

$$p\left(f_*|\mathbf{x}_*,\mathbf{X},\mathbf{f},\boldsymbol{\theta}\right) = \mathcal{N}\left(f_*|\mathbf{k}_*^T\boldsymbol{\alpha}, k_{**} - \mathbf{l}^T\mathbf{l}\right) \quad , \tag{2.64}$$

with

$$\boldsymbol{\alpha} = \mathbf{L}^T \setminus \left(\mathbf{L} \setminus \mathbf{f}\right) \tag{2.65}$$

$$\mathbf{l} = \mathbf{L} \setminus \mathbf{k}_* \quad , \tag{2.66}$$

where $\setminus$ is the operator for solving an equation system. Since $\mathbf{L}$ is a triangular matrix, the equation system can be solved in quadratic time. If now a new instance needs to be added, the triangular matrix $\mathbf{L}$ can be updated by

$$\mathbf{L}_{\text{new}} = \begin{pmatrix} \mathbf{L} & 0 \\ \mathbf{l}^T & l_* \end{pmatrix} \quad , \tag{2.67}$$

where

$$l_* = \sqrt{k\left(\mathbf{x}_*,\mathbf{x}_*\right) - \|\mathbf{l}\|_2^2 + \sigma_y^2} \quad . \tag{2.68}$$

Now $\boldsymbol{\alpha}$ and $\mathbf{l}$ can be recomputed as described in Equation (2.65) and (2.66).

2.7 Surrogate Models

In this section, we review the most important surrogate models for Bayesian optimization for configuration optimization in machine learning. We distinguish between plain and transfer surrogate models which we define as follows. Let D_{new} be the data set for which we search for the optimal configurations. Then, a plain surrogate model is one that considers only observations and information of $f_{D_{\text{new}}}$ and D_{new}. Otherwise, we define transfer surrogate models as those surrogate models that use information of additional data sets besides D_{new}.

2.7.1 Plain Surrogate Models

Spearmint

Snoek et al.[68] are the first to propose the use of Bayesian optimization for hyperparameter optimization in machine learning. They use a Gaussian process as a surrogate model. In their work they investigate different aspects of Bayesian optimization with respect to hyperparameter optimization. They compare different acquisition functions, propose a run-time-aware acquisition function and describe how to run the optimization in parallel.

Sequential Model-based Algorithm Configuration

Hutter et al.[35] propose the use of random forests as a surrogate model. While the original work focuses on the algorithm configuration problem for solvers of hard computational problems (Boolean satisfiability problem (SAT) and mixed integer programming (MIP)), it is currently used in libraries for autonomous machine learning such as auto-sklearn[23] and Auto-WEKA[75].

2.7.2 Transfer Surrogate Models

Transfer surrogate models use meta-knowledge of various data sets. In practice, this usually means nothing else but extending the observation history by adding observations from other data sets. Assuming we have observations from M many data sets, we start Bayesian optimization with the non-empty observation history

$$\mathcal{H} = \{(\mathbf{x}_1, f_{D_1}(\mathbf{x}_1)), \ldots, (\mathbf{x}_1, f_{D_1}(\mathbf{x}_N)), (\mathbf{x}_1, \ldots, f_{D_M}(\mathbf{x}_1)), \ldots, (\mathbf{x}_1, f_{D_M}(\mathbf{x}_N))\} \quad . \tag{2.69}$$

Furthermore, various techniques of using meta-features might be considered but often they are simply added to the meta-predictors $\mathbf{x}$.

Surrogate Collaborative Tuning

Bardenet et al.[4] are the first to propose a transfer surrogate. They show how to learn a single surrogate model over observations from many data sets. Since the same algorithm applied to different data sets leads to loss values that can differ significantly in scale, they recommend tackling this problem using a ranking model instead of a regression model. They finally propose to use SVM^{RANK} with an RBF kernel to learn a ranking of hyperparameter configurations per data set. The ranker itself does not provide the needed uncertainty estimations. Thus, they finally fit a Gaussian process to the output of the ranker.

Gaussian Process with Multi-Kernel Learning

Yogatama and Mann[87] propose to train a Gaussian process directly on the meta-data. To overcome the problem of different scales on different data sets, they propose to standardize the loss per data set by removing the mean and scaling to unit variance. Furthermore, they propose a linear combination of a squared exponential kernel with automatic relevance determination (SE-ARD) for points in the same data set and a nearest neighbor kernel for modeling similarities between data sets. They define the kernel as

$$k_{\text{MKL}}\left(\left(\mathbf{x}_i, D_k\right), \left(\mathbf{x}_j, D_l\right)\right) = \alpha \mathbb{I}\left(D_k = D_l\right) k_{\text{SE-ARD}}\left(\mathbf{x}_i, \mathbf{x}_j\right) + (1-\alpha)\, \mathbb{I}\left(D_l \in \mathcal{N}\left(D\right)\right) k_{\text{NN}}\left(\mathbf{x}_i, \mathbf{x}_j\right) \tag{2.70}$$

where the SE-ARD kernel is defined as

$$k_{\text{SE-ARD}}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \exp\left(-\frac{1}{2}\sum_p \frac{\left(\mathbf{x}_{i,p} - \mathbf{x}_{j,p}\right)^2}{\boldsymbol{\sigma}_p^2}\right) \tag{2.71}$$

and the data set similarity kernel as

$$k_{\text{NN}}\left(\mathbf{x}_i, \mathbf{x}_j\right) = 1 - \frac{1}{B}\left\|\mathbf{x}_i - \mathbf{x}_j\right\|, \tag{2.72}$$

where B must be chosen such that k_{NN} is always non-negative and $\mathcal{N}\left(D\right)$ denotes the set of neighbored data sets with respect to a distance function. The distance between two data sets is defined as the Euclidean distance between its meta-features. Meta-features are used only to determine the distance between data sets. They are not used within the kernels. Hence, meta-features are only used to estimate the values of k_{NN}.

Factorized Multilayer Perceptron

Schilling et al.[61] propose to use a modified multilayer perceptron as a surrogate model. Meta-instances are extended by meta-features and data set indicators. Data set indicators are nothing else but $M+1$ additional binary predictors, one for each data set. The indicator is 1 if the meta-instance belongs to the corresponding data set, 0 otherwise. The modified multilayer perceptron uses a different activation function in the first layer than the standard multilayer perceptron. Instead of using the typical sigmoid activation function, they propose

$$\text{logistic}\left(\mathbf{w}_0 + \sum_{i=1}^{P} \mathbf{w}_i \mathbf{x}_i + \sum_{i=1}^{P}\sum_{j=i+1}^{P} \mathbf{v}_i^T \mathbf{v}_j \mathbf{x}_i \mathbf{x}_j\right) \tag{2.73}$$

where logistic is the logistic function,

$$\text{logistic}(x) = \left(1 + e^{-x}\right)^{-1} \quad , \tag{2.74}$$

and $\mathbf{V} \in \mathbb{R}^{P \times K}$ are latent variables. This model is based on factorization machines[59] which are a prediction model for recommender systems. The underlying idea is to learn a latent representation for each data set to model similarities between data sets. Simultaneously, Snoek et al.[69] proposed the use of neural networks as a plain surrogate model.

2.8 Further Related Work

We want to briefly mention further work directly related to Bayesian optimization and other related work for automatic hyperparameter optimization.

There are several strategies to find a set of initial configurations for hyperparameter optimization methods. Reif et al.[58] propose to initialize a hyperparameter search based on genetic algorithms with the best hyperparameters on other data sets, where the similarity of data sets is defined through meta-features. Feurer et al.[24] propose the same idea for Bayesian optimization which was later extended[23,80]. The drawback of these approaches are that they do not consider whether the initial hyperparameter configurations are very close to each other and therefore may waste computation time by choosing too similar hyperparameters initially. Thus, we propose to learn a set of initial hyperparameter configurations by optimizing a meta-loss that maximizes the overall improvement on the meta-data in Chapter 6.

The idea of using meta-knowledge in Bayesian optimization is to find better performing prediction models within a smaller fraction of time. Another idea applicable to models that are learned in an iterative fashion is to predict the learning curve, i.e. the performance of the resulting model after a number of epochs. Domhan et al.[21] predict the performance of the hyperparameter configuration based on the partially observed learning curve after a few iterations. If the final performance is likely worse than the current best configuration, the process is stopped, the configuration discarded and they continue with another configuration. Swersky et al.[73] propose a similar approach but they never discard a configuration. Instead, they learn the models for various hyperparameter configurations at the same time and switch from one learning process to another if it turns out to be more promising.

There exists a plethora of methods for hyperparameter optimization designed for a specific algorithm. Resulting from its popularity, there are several methods that optimize SVM hyperparameters directly for the particular choice of a least squares SVM[2,14,28]. These methods are based on genetic algorithms[18,25], whereas some are deterministic[39]. There are various other approaches for different

algorithms or specific problem settings, such as for general regression and time-series models[51], for Bayesian topical trend analysis[50], for log-linear models[20] and for regression when the sample size is small[13]. Moreover, hyperparameter learning is used for probabilistic-prototype-based models[63], large scale hierarchical kernel methods[64] and graph-based semi-supervised classification models[38]. The major limitation of all of these methods is that they are specifically tailored to one particular model and only work well in certain scenarios, this is a drawback that Bayesian optimization-based methods alleviate.

Meta-features are descriptive characteristics of a data set and thus an essential component of all traditional meta-learning methods that are learning across problems. In this work, we often use pairwise comparisons of the performance of two hyperparameter configurations on one data set compared to another. This is a very special instance of landmarkers[55], sometimes called relative landmarkers. Landmark features are created by applying very fast machine learning algorithms (e.g. decision stumps, linear regression) to the data. Their performance is added as a meta-feature. In contrast to our approach, we only use the performance of algorithms and hyperparameter configurations which we have evaluated during our optimization process. Hence, no additional time was spend for estimating these landmarkers. This idea has been already employed by some others[47,71]. In contrast to their work, we propose a way of using these meta-features also in cases with continuous hyperparameters. For continuous hyperparameters it is very unlikely that we have seen the same hyperparameter configurations for all data sets. Hence, the approach of pairwise comparisons proposed by the literature works only if we either only want to find the best algorithm and ignore the hyperparameters[71] or discretize the hyperparameters[47]. We overcome this problem by predicting the performance of a hyperparameter configuration if it is not part of our meta-data set.

Average ranking ranks all configurations based on their average performance in former experiments and provides a static sequence for new data sets[9]. Active testing[47] is using relative landmarkers in order to adapt to the new data set. The configuration which is chosen next for evaluation depends on how much the loss was decreased over the current best configuration on other data sets weighted by the data set similarity. In contrast to average ranking and active testing, A-SMFO and NN-SMFO presented in Chapter 4 optimize directly for a given meta-loss and consider all evaluated configurations. Average ranking does not consider them at all, active testing only considers the current best configuration. Average ranking and active testing are reviewed and extended recently by Abdulrahman et al.[1].

3

Meta-Data Sets & Experimental Setup

We will use this chapter to explain how we created our meta-data set and give detailed insights into the meta-data. Furthermore, we describe the experimental setup, define our evaluation metrics and motivate why these are good metrics to judge an optimization method.

3.1 Meta-Data Sets

In the following section, we will describe the meta-data sets which we created to evaluate the different optimization techniques. We will describe how we created them and which data sets we used. Furthermore, we conduct an in-depth analysis of the meta-data and the hyperparameter sensitivity of different algorithms. Finally, we provide some examples of response functions.

3.1.1 AdaBoost Meta-Data Set

The AdaBoost was created using 50 classification data sets chosen at random from the UCI repository. Existing train/test splits were merged, shuffled and split into 80% train and 20% test. The AdaBoost meta-data set is our smallest meta-data set. It was created using Adaboost with decision products as weak learners[40]. This algorithm has two hyperparameters, the number of iterations I and the number of product terms M. The classification error was precomputed on the grid $I \in \{2, 5, 10, 20, 50, 10^2, 2 \cdot 10^2, 5 \cdot 10^2, 10^3, 2 \cdot 10^3, 5 \cdot 10^3, 10^5\}$ and $M \in \{2, 3, 4, 5, 7, 10, 15, 20, 30\}$.

Figure 3.1 presents some information about the AdaBoost meta-data set. Each of the 108 hyperparameter configurations is represented by a colored square. The left plot visualizes on how many of

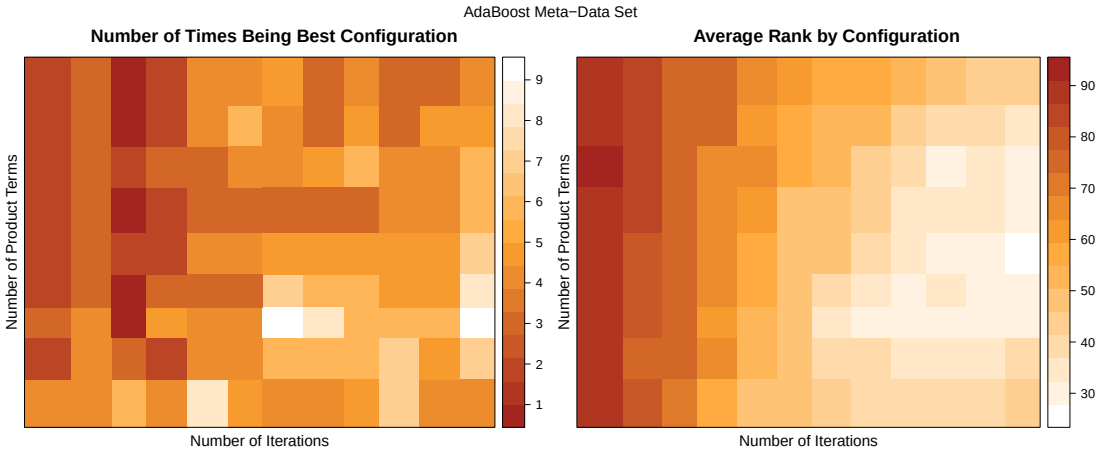

Figure 3.1: Left: The number of times each hyperparameter configuration has been the best configuration on the 50 data sets. There is no configuration that has not been best on any data set at least once. Right: The average rank by configuration over all data sets. Unsurprisingly, the higher the number of iterations, the better the configuration on average.

the 50 data set each hyperparameter configuration has been the best configuration among the 108 tested. The right plot presents the average rank for each configuration over all 50 data sets. The rank of a hyperparameter configuration for one data set is computed by ordering all configurations by loss. The lower the loss, the lower the rank. If the loss is equal, the average rank is used. The estimated ranks per data set are averaged over all data sets. Hence, hyperparameter configurations with a small average rank are on average good configurations. We can observe that configurations with more iterations and less product terms tend to be better. This behavior can be explained easily. With an increasing number of iterations, more and more weak learners are added to the boosted ensemble. Typically, this leads to a boost in performance. Increasing the number of product terms leads to an increase of complexity for each weak learner. Depending on the data set, this can lead to overfitting. Thus, the combination of many weak learners with moderate complexity is often the best choice. Figure 3.2 presents the first 21 response functions of our data sets. The before mentioned behavior can be observed on many data sets (e.g. A9A and australian). However, there are also some data sets that show a completely different behavior (e.g. coil2000 or diabetes). The increase in iterations seems to lead to overfitting and a decrease in performance can be observed.

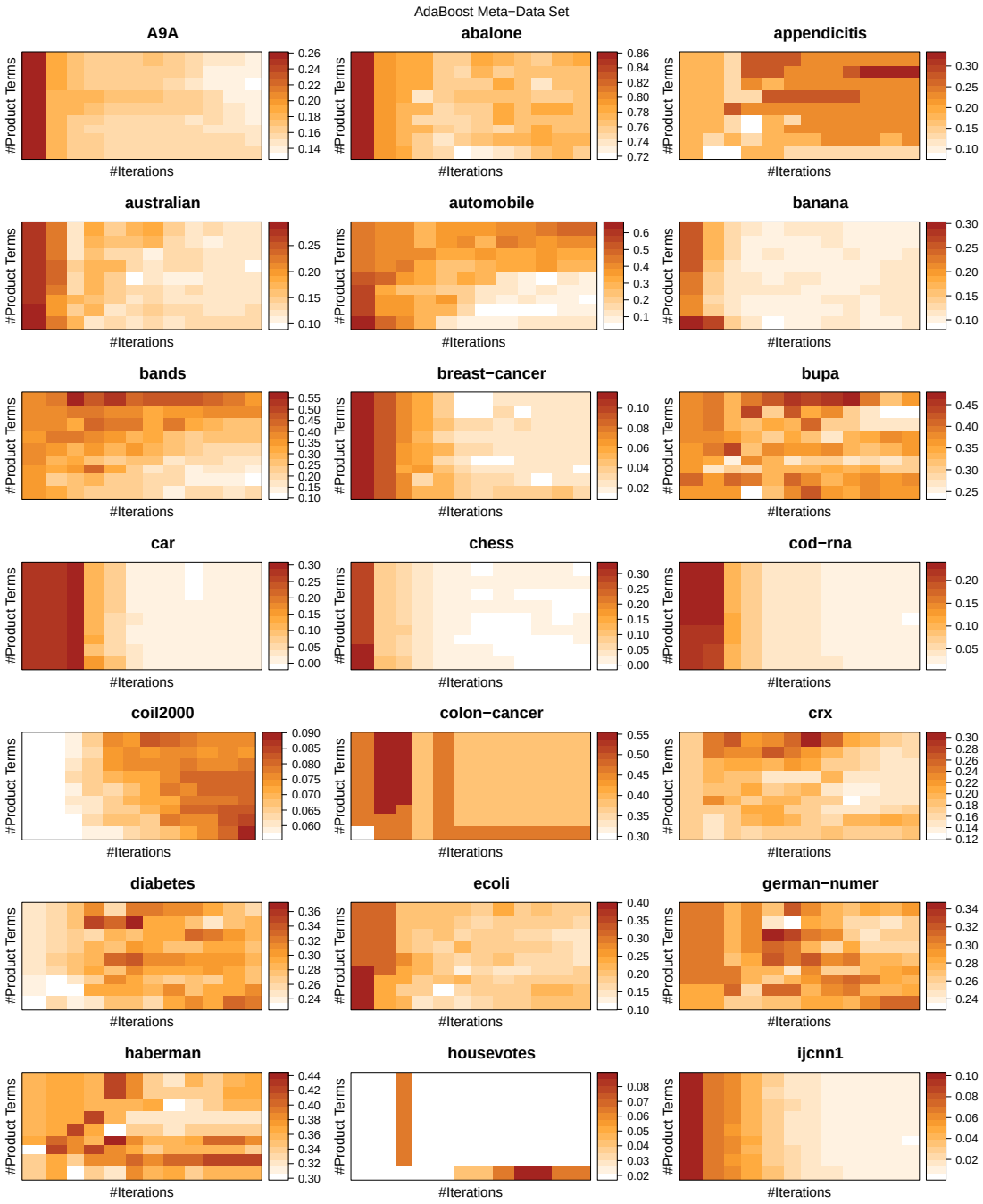

Figure 3.2: The classification error of the AdaBoost classifier on multiple data sets. On many data sets we can observe that the performance increases with a growing number of iterations. The visualization of all 50 data sets is available at `http://www.hylap.org/meta_data/adaboost/`.

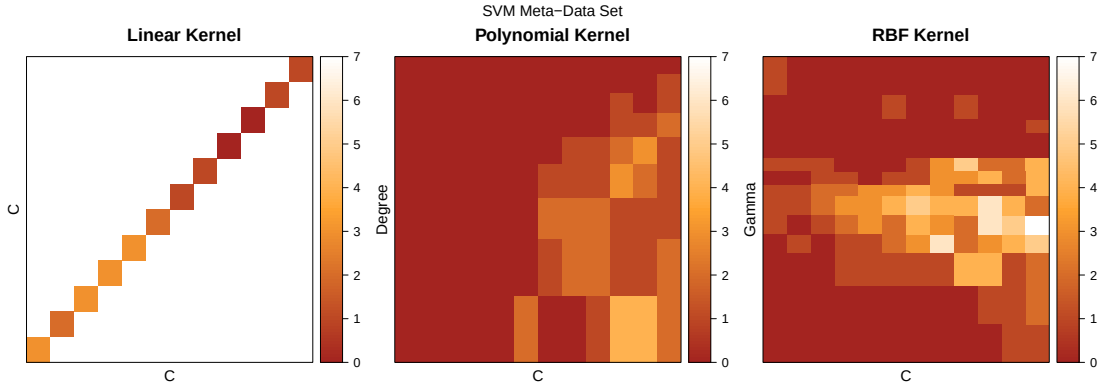

Figure 3.3: In this plot we visualize which specific configuration has been best on how many of the 50 data sets. There are clear regions of good algorithm configurations.

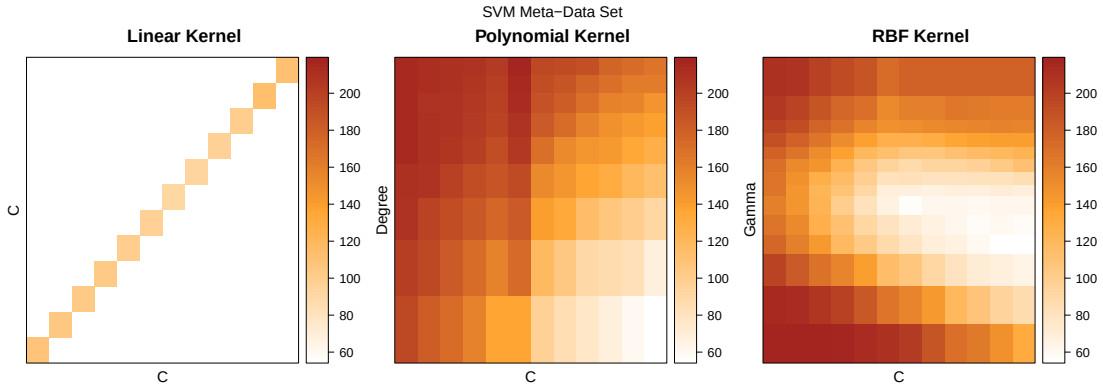

Figure 3.4: In this plot we plot the average rank each configuration has achieved over all 50 data sets. The smaller the better. Unsurprisingly, we see parallels to Figure 3.3.

3.1.2 SVM Meta-Data Set

We created the SVM meta-data set using the same 50 classification data sets used for the creation of the AdaBoost meta-data set. We trained an SVM[12] with a linear, polynomial and Gaussian kernel. We optimized the trade-off parameter C, the degree of the polynomial kernel d and the width γ of the Gaussian kernel. If a hyperparameter was not involved, its value was set to 0. We precomputed the classification error on the grid $C \in \{2^{-5}, \ldots, 2^{6}\}$, $d \in \{2, \ldots, 10\}$ and $\gamma \in \{10^{-4}, 10^{-3}, 10^{-2}, 0.05, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 10^{2}, 10^{3}\}$. To represent the choice of the kernel, we used one hot encoding. This means, we added three indicator variables, one for each kernel. Their values are 1 if the kernel is used and 0 otherwise. Concluding, we computed 288 meta-instances per data set in about 160 CPU hours.

As for the AdaBoost meta-data set, we provide similar statistics for the SVM meta-data set. Figures

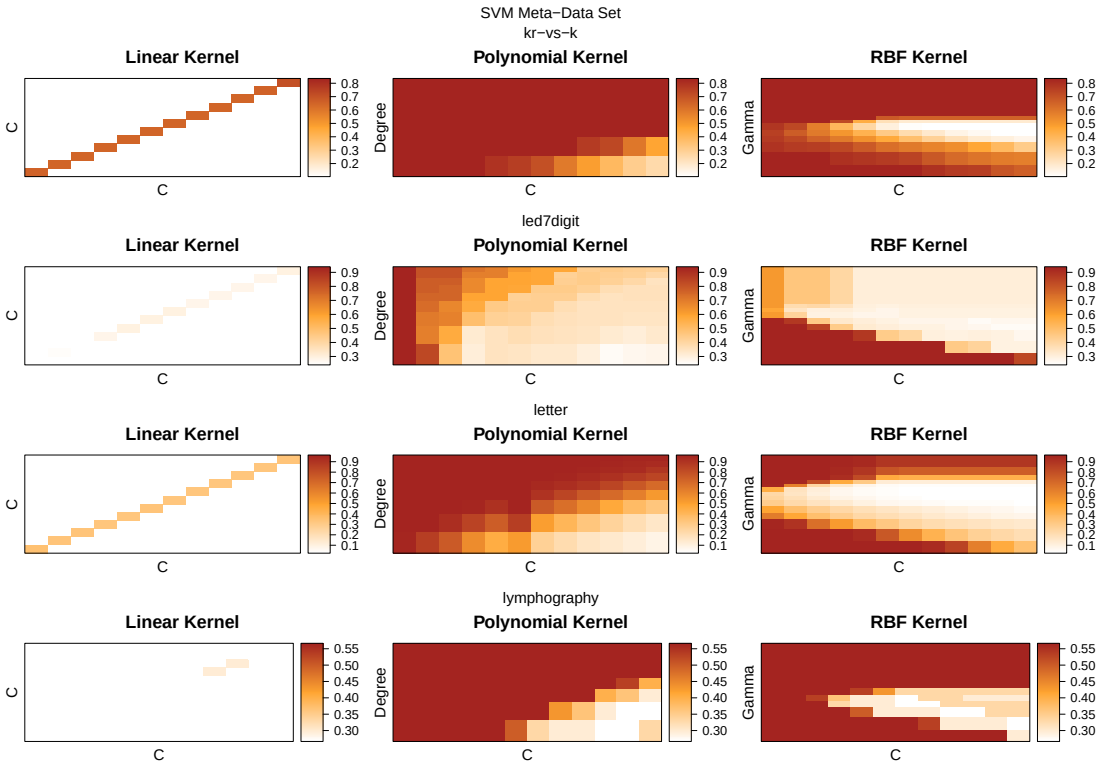

Figure 3.5: The classification error of the SVM classifier on multiple data sets. The visualization of all 50 data sets is available at `http://www.hylap.org/meta_data/svm/`.

3.3 and 3.4 represent each hyperparameter configurations with a colored square, broken down by kernel. The linear kernel has only a single hyperparameter, hence only the diagonal is of importance. Figure 3.3 shows on how many data set each hyperparameter configuration has been the best one. Figure 3.4 visualizes the average rank of each hyperparameter configuration over all 50 data sets. The average rank is computed as explained in the previous section for Figure 3.1. It seems that there is a valley of good hyperparameter configurations for the polynomial and RBF kernel. Having a look into the example response functions in Figure 3.5, 3.6, 3.7 and 3.8, one can indeed find a valley shaped response function. There is always some valley with sharp edges where the response function changes to a constant function. These valleys differ in their position and their shape. However, this is an interesting property which we did not investigate. It allows the use of a surrogate model that is designed to model this special function behavior. In combination with active learning, one might be able to estimate the shape and position of the valley accurately and quickly.

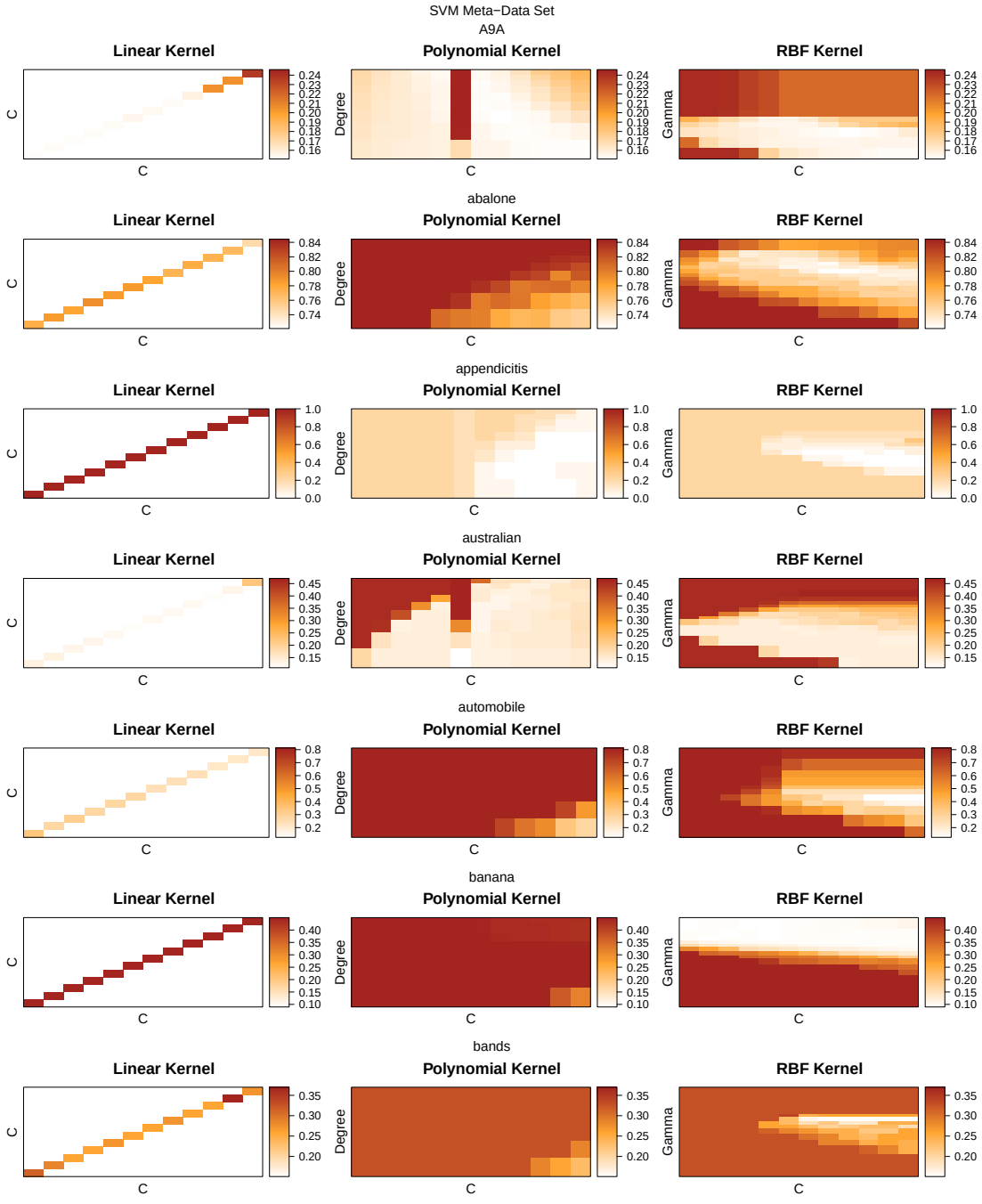

Figure 3.6: The classification error of the SVM classifier on multiple data sets. The visualization of all 50 data sets is available at `http://www.hylap.org/meta_data/svm/`.

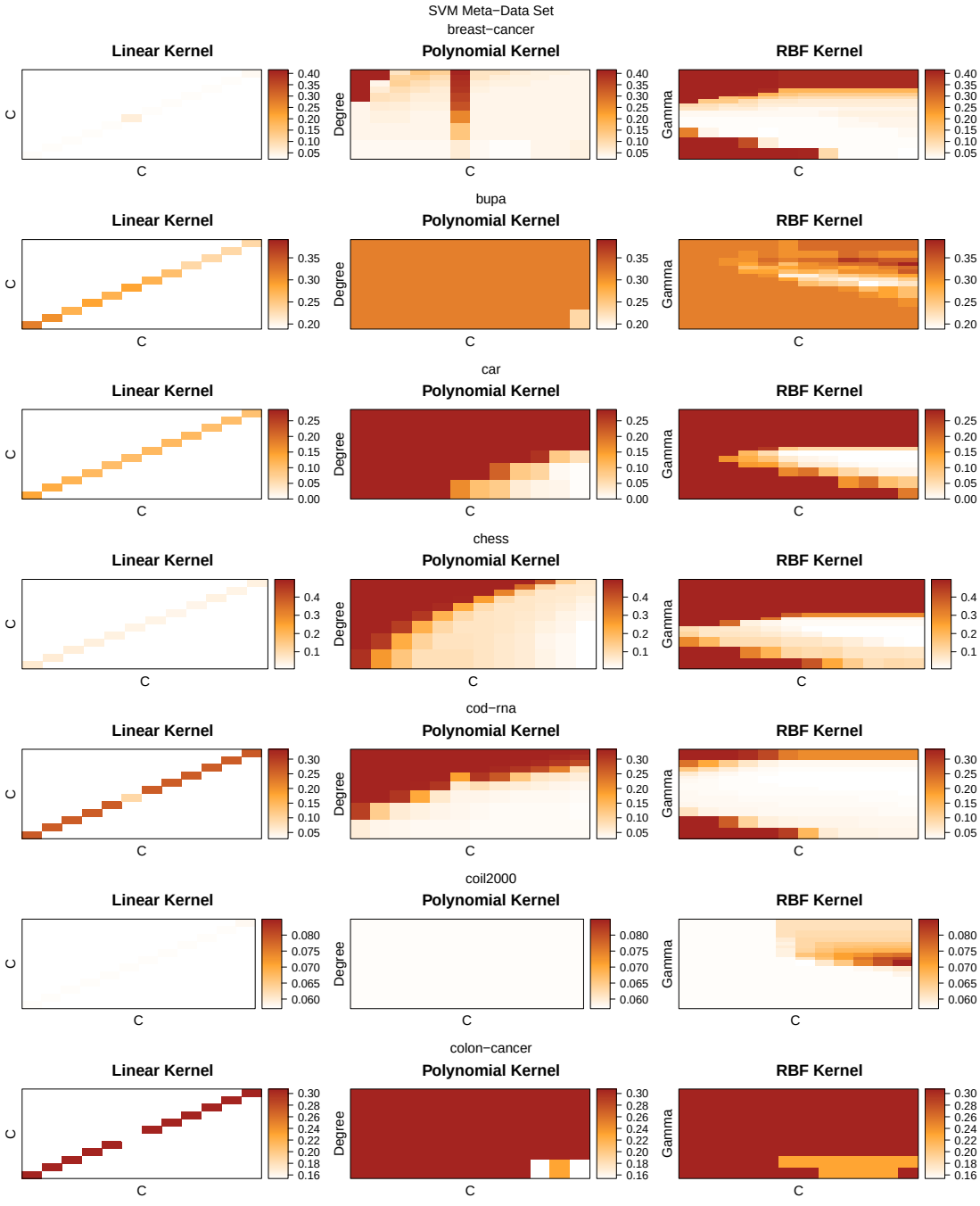

Figure 3.7: The classification error of the SVM classifier on multiple data sets. The visualization of all 50 data sets is available at `http://www.hylap.org/meta_data/svm/`.

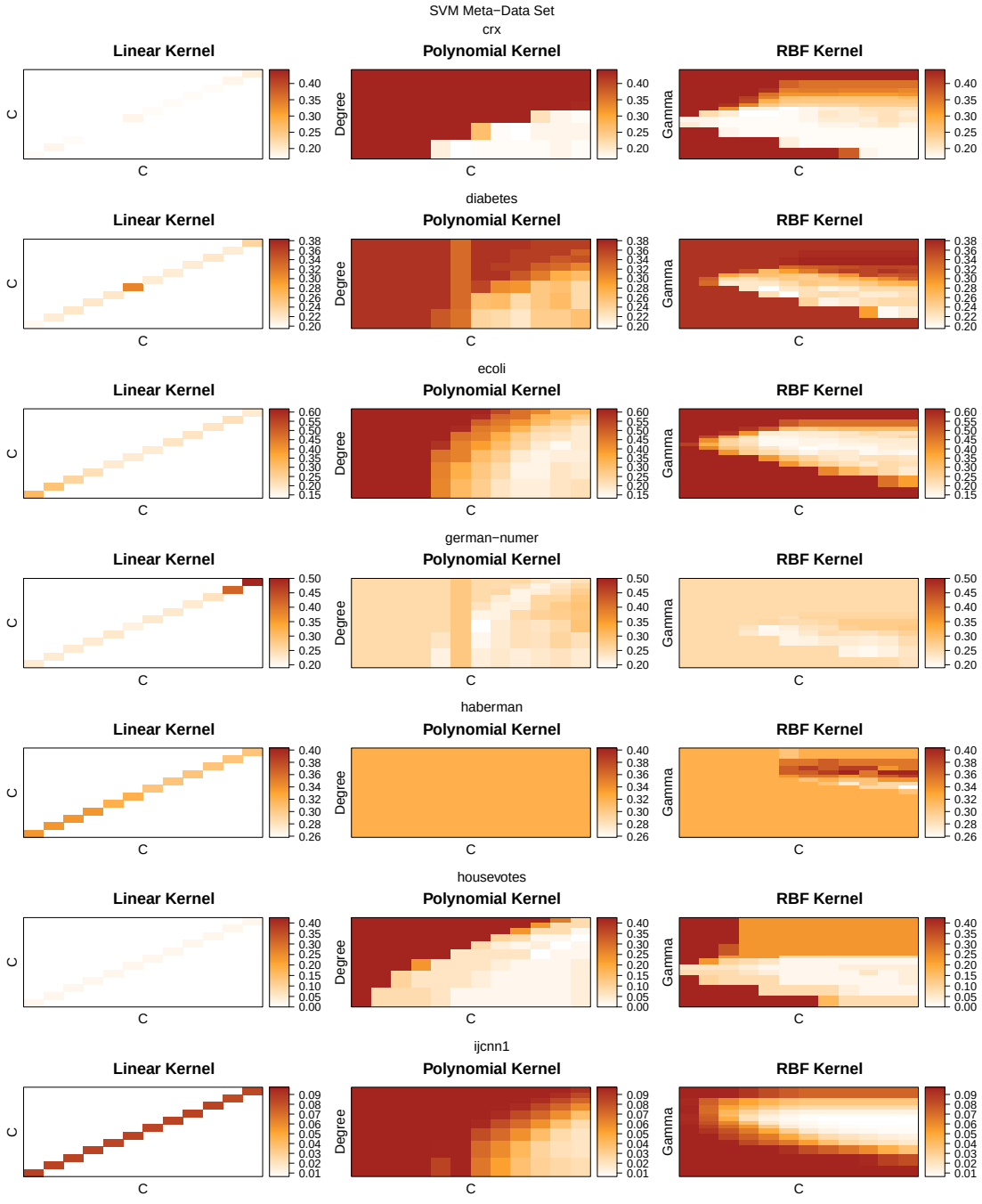

Figure 3.8: The classification error of the SVM classifier on multiple data sets. The visualization of all 50 data sets is available at `http://www.hylap.org/meta_data/svm/`.

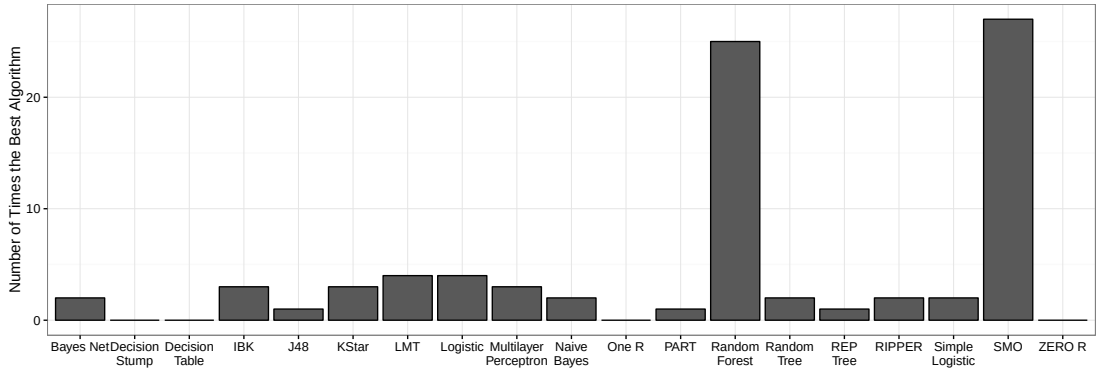

Figure 3.9: This plot presents the number of times each algorithm had a hyperparameter configuration that yield the best classification performance. Support Vector Machines and Random Forests are among the best classifiers.

3.1.3 Weka Meta-Data Set

In our endeavor to evaluate different hyperparameter optimization strategies for the task of combined algorithm selection and hyperparameter configuration optimization, we created a meta-data set containing multiple different algorithms. We used 19 different Weka classifiers[29] to precompute the classification error on a grid which resulted into 21,871 hyperparameter configurations per data set. The exact grid is given in Table 3.1 to Table 3.3. We selected 59 smaller classification data sets which we preprocessed in the same way as done for the AdaBoost and SVM meta-data set. In more than 891 CPU hours we created the Weka meta-data set which contains 1,290,389 meta-instances.

An in-depth analysis for the Weka meta-data set as provided for the AdaBoost and SVM meta-data set is difficult due to the high dimensionality of the meta-data set. Hence, we only provide statistics by algorithm. Figure 3.9 presents how frequently we found a configuration for an algorithm that provided the best classification performance. Unsurprisingly, more complex classifiers such as SVM and Random Forest outperform simpler classifiers such as decision stumps or One R. However, Figure 3.10 shows that choosing a more complex classifier does not automatically lead to better results. In this plot we see the average rank over all data sets for the worst (Maximum Average Rank) and the best (Minimum Average Rank) hyperparameter configuration on average. This value is equal for algorithms without hyperparameters (decision stump and Zero R). However, most algorithms have a big difference between the best and worst configuration, once again showing how important the configuration optimization problem is. Furthermore, in most cases the worst configuration of one algorithm rarely outperforms the best configuration of another. The logistic regression is strictly outperforming the decision stump but this is expected. The decision stump's model assumption is too simple and will not provide good results for most data sets. Methods that are using meta-

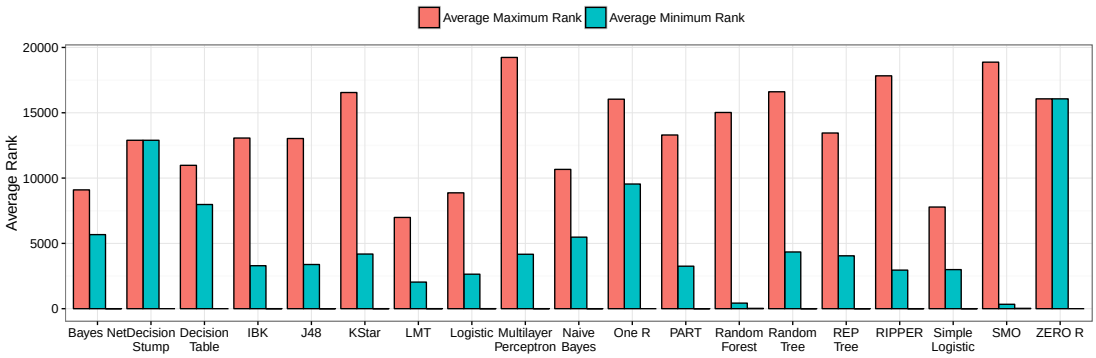

Figure 3.10: Average difference between the best and worst hyperparameter configuration per algorithm. Obviously, optimizing the hyperparameters make a huge difference. In many cases a well-tuned algorithm can outperform any other algorithm with random configurations.

knowledge should be able to identify these relationships and use this knowledge while guiding the optimization process.

3.1.4 Meta-Features

We enriched the meta-data sets with the 22 meta-features listed in Table 3.4 if not otherwise stated. Meta-features are extracted from the data set itself. In the following, we describe the meta-features of a given classification data set D with predictors $\mathbf{X} \in \mathbb{R}^{n \times m}$ and targets $\mathbf{y} \in \{1, \ldots, c\}^n$ in detail. The class probability for a class i is defined as the number of instances belonging to this class divided by all instances.

$$p\left(C = i|D\right) = \frac{1}{n} \sum_{j=1}^{n} \mathbb{I}\left(\mathbf{y}_i = i\right) \tag{3.1}$$

Then, minimum, maximum, mean and standard deviation class probability can be estimated directly from these probabilities.

The skewness and kurtosis are the third and fourth standardized moment, respectively. They are estimated for each of the m predictors. The k-th standardized moment is defined as

$$\frac{1}{n} \frac{\sum_{i=1}^{j} \left(\mathbf{x}_{i,j} - \mathbf{m}_j\right)^k}{\boldsymbol{\sigma}_j^k} \quad , \tag{3.2}$$

where $\mathbf{m}_j$ and $\boldsymbol{\sigma}_j$ are the mean and standard deviation of the j-th predictor, respectively. Then, similarly to the class probability, the min, max, mean and standard deviation kurtosis and skewness

Table 3.1: The grid used to create the Weka meta-data set (Part 1).

Bayes-Net	
estimator	SimpleEstimator
searchAlgorithm	HillClimber, K2, LAGDHillClimber, TabuSearch, TAN
scoreType	BAYES, BDeu, MDL, AIC
Naive Bayes	
No hyperparameters	
Logistic	
maxIts	10, 50, 100, 200, 500, 1000
ridge	0, 1E-8, 1E-7, 1E-6, 1E-5, 1E-4, 1E-3, 1E-2, 0.1, 0.5, 1, 5, 10
MultiLayerPerceptron	
hiddenLayers	1
learningRate	0.001, 0.01, 0.1, 0.25, 0.5
momentum	0.001, 0.01, 0.1, 0.25, 0.5
trainingTime	10, 50, 100, 250, 500, 1000
validationSetSize	0, 2, 4, 6, 8, 10
validationThreshold	5, 10, 20, 50, 100
SMO	
c	0.03125, 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32
kernel	PolyKernel, RBFKernel, Puk
gamma	0.001, 0.01, 0.1, 0.25, 0.5
omega	0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 1000
sigma	0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 1000
degree	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
useLowerOrder	True, False
SimpleLogistic	
weightTrimBeta	0, 0.0001, 0.001, 0.01
heuristicStop	0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
maxBoostingIterations	100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
useAIC	True, False
useCrossValidation	True, False

Table 3.2: The grid used to create the Weka meta-data set (Part 2).

IBk	
KNN	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50
distanceWeighting	No distance weighting, Weight by 1/distance, Weight by 1-distance
KStar	
globalBlend	0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100
entropicAutoBlend	True, False
DecisionTable	
search	BestFirst, GreedyStepwise
conservativeForward	True, False
direction	Forward, Backward
searchTermination	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
JRip	
folds	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
minNo	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
optimizations	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
usePruning	True, False
OneR	
minBucketSize	1, 2, ..., 50
PART	
confidenceFactor	0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4
minNumObj	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
numFolds	2, 3, 4, 5, 6, 7, 8, 9, 10
Pruning	ReducedErrorPruning, Pruning, Unpruned
ZeroR	
No hyperparameters	
DecisionStump	
No hyperparameters	

Table 3.3: The grid used to create the Weka meta-data set (Part 3).

J48	
confidenceFactor	0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4
minNumObj	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
numFolds	2, 3, 4, 5, 6, 7, 8, 9, 10
Pruning	ReducedErrorPruning, Pruning, Unpruned
subtreeRaising	True, False
useLaplace	True, False
LMT	
convertNominal	True, False
minNumInstances	2, 4, 6, 8, 10, 12, 15, 17, 20, 25, 50
splitOnResiduals	True, False
useAIC	True, False
weightTrimBeta	0, 0.0001, 0.001, 0.01
RepTree	
minNum	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
minVarianceProb	0, 1E-8, 1E-7, 1E-6, 1E-5, 1E-4, 1E-3, 1E-2, 0.1, 0.5, 1
numFolds	2, 3, 4, 5, 6, 7, 8, 9, 10
noPruning	True, False
RandomForest	
maxDepth	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
numFeatures	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
numTrees	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 500
RandomTree	
maxDepth	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
minNum	0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, 9, 10
numFold	0, 2, 3, 4, 5

Table 3.4: The list of all meta-features used by us.

Meta-Features	
Number of Classes	Class Probability Max
Number of Instances	Class Probability Mean
Log Number of Instances	Class Probability Standard Deviation
Number of Features	Kurtosis Min
Log Number of Features	Kurtosis Max
Data Set Dimensionality	Kurtosis Mean
Log Data Set Dimensionality	Kurtosis Standard Deviation
Inverse Data Set Dimensionality	Skewness Min
Log Inverse Data Set Dimensionality	Skewness Max
Class Cross Entropy	Skewness Mean
Class Probability Min	Skewness Standard Deviation

are estimated. The data set dimensionality is defined by n/m and the class cross entropy by

$$H[C|D] = -\sum_{i=1}^{c} p\left(C = i|D\right) \log p\left(C = i|D\right) \quad . \tag{3.3}$$

3.2 Experimental Details

In this section we will discuss the experimental setup used throughout the thesis. Furthermore, we formally define our evaluation metrics and motivate their use.

3.2.1 Evaluation Metrics

Throughout this thesis we compare different optimization methods with respect to up to three different evaluation metrics. We will present them in the upcoming sections and argue why these metrics represent the quality of an optimization method.

Average Rank

The *average rank among different hyperparameter optimization strategies* or for short simply *average rank* is a relative metric between different optimization strategies. In order to estimate the average rank, the optimization strategies are ranked by the best hyperparameter configuration that they have found so far for each data set. Ties are solved by granting them the average rank. Assuming we have four different optimization strategies which have found hyperparameter configurations on a specific

data set that achieve a classification error of 0.13, 0.15, 0.15 and 0.16, respectively. Then the ranking is 1, 2.5, 2.5, 4. The average over all data sets of these ranks yield the average rank.

The average rank is used in every related work we compare to and hence, is automatically a metric we consider. While the average rank gives an impression which methods tend to be better, it does not show how big the difference actually is.

Average Distance to the Global Minimum

Compared to the average rank metric, the average distance to the global minimum is not a metric that is relative to different optimization strategies. In our experiments, we study the hyperparameter optimization of classification models and thus the average distance to the global minimum reflects the average classification error across data sets. To overcome the different error scales among data sets, the classification error is scaled between 0 and 1 only for evaluation purposes. This scaling is not applied during the experiments because the knowledge of maximum and minimum errors is unknown and is part of the problem. Using the notation from Chapter 2, the average distance to the global minimum is defined as

$$\text{ADTM}\left(\mathcal{D}, \mathcal{X}_T\right) = \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \frac{\min_{\mathbf{x} \in \mathcal{X}_T} f_D\left(\mathbf{x}\right) - f_D^{\min}}{f_D^{\max} - f_D^{\min}} \quad , \tag{3.4}$$

where $f_D^{\max}$ and $f_D^{\min}$ are the worst and best possible value of the response function, respectively. We approximate these values by using the smallest and largest value that was found during the grid search when creating the meta-data set.

In contrast to the average rank, this metric gives information about two other aspects. Firstly, it gives insight how big the difference is on average between data sets with respect to the normalized loss. Secondly, it enables to see how close the methods are to the global optimum.

Fraction of Unsolved Data Sets

The fraction of unsolved data sets is the fraction of data sets where the method has not achieved a DTM of 0. This is of interest because a method can only improve on a data set if it does not achieve a DTM of 0. Thus, a method that has a small fraction of unsolved data sets has less data sets where it can improve on. Furthermore, it is a direct indication how far the methods have converged.

3.2.2 Experimental Setup

We created our meta-data sets by precomputing the response function values on a grid. We defined the grid

$$G = \mathbf{X}_1 \times \cdots \times \mathbf{X}_P \tag{3.5}$$

by selecting n_i many arguments for dimensionality i. All methods based on Bayesian optimization have the possibility to recommend configurations which they have never seen before. To simulate this behavior, each optimization method has only access to a subset of the grid. For each continuous or ordinal hyperparameter, the methods only use one third of the precomputed values. Examples for these kind of hyperparameters are the ridge parameter of a logistic regression or the depth of a decision tree. For the other hyperparameter dimensions, we use the entire set. The reason for this is that these hyperparameters usually define the choice of an algorithm or kernel. If our optimization methods do not see any example for them, learning across problems becomes difficult. Assuming we have a hyperparameter dimensionality defined by

$$\mathbf{X}_i = \{\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,n_i}\} \quad . \tag{3.6}$$

Then, we define the subset for training as

$$\mathbf{X}_i^{\text{train}} = \left\{\mathbf{x}_{i,1}, \mathbf{x}_{i,4}, \mathbf{x}_{i,7}, \ldots\right\} \quad . \tag{3.7}$$

Then, only the configurations in

$$G^{\text{train}} = \mathbf{X}_1^{\text{train}} \times \cdots \times \mathbf{X}_P^{\text{train}} \tag{3.8}$$

are used for training. The evaluation of each optimization method is conducted on the entire grid G.

3.2.3 Competitor Methods & Meta-Hyperparameters

In the previous chapter we introduced the main competitor methods we are comparing to throughout the thesis. We want to avoid mentioning and explaining these methods in every chapter again. However, we also want to avoid that the reader needs to go back in case he forgot the abbreviations or which method was doing what. Therefore, we add a summary table of all methods with their corresponding abbreviation in each chapter. Table 3.5 is an example for such a summary table. It contains the name of the method and the abbreviation used in the text and the plots. Furthermore, it shows which surrogate model the method utilizes, whether it employs meta-knowledge and where

Table 3.5: An example for the brief overview of competitor methods.

Name	Abbrev.	Surrogate Model	Meta-Knowledge	Details
Random Search	Random	None	✗	Sec. 2.2
Independent Gaussian Process	I-GP	Gaussian Process	✗	Sec. 2.7.1
Independent Random Forest	I-RF	Random Forest	✗	Sec. 2.7.1
Meta-Initialization	XYZ (init)	Depends on XYZ	✓	Chap. 6
Surrogate Collaborative Tuning	SCoT	SVMRank + Gaussian Process	✓	Sec. 2.7.2
Gaussian Process with Multi-Kernel Learning	MKL-GP	Gaussian Process	✓	Sec. 2.7.2
Factorized Multilayer Perceptron	FMLP	Neural Network	✓	Sec. 2.7.2

more information about the method can be found in this thesis. The meta-initialization can be used with various methods, hence we used the placeholder XYZ. We typically combine it with I-GP and I-RF. The resulting abbreviations are I-GP (init) and I-RF (init).

The meta-hyperparameters of the different methods are optimized as follows. Meta-hyperparameters of Gaussian processes are estimated by maximizing the marginal likelihood (see Equation (2.63)). Other meta-hyperparameters are estimated using a grid search on the meta-data using leave-one-data-set-out cross-validation. We searched the trade-off parameter C of SVMRank on a logarithmic scale from 10^{-4} to 10^{0}. We achieved best results for many data sets for $C = 10^{-2}$ on the AdaBoost and SVM meta-data set.

For MKL-GP we set $\alpha = 0.3$ as recommended by the authors. We first manually identified a useful region for the size of the neighborhood k. Then, we estimated the optimal k for values smaller or equal than 10 using leave-one-data-set-out cross-validation. Best results were usually achieved with $2 \leq k \leq 4$ on the AdaBoost and SVM meta-data set.

We used the provided implementation of FMLP. Since the authors conducted their results on our meta-data sets, we simply used their recommended meta-hyperparameters.

With a grid search using leave-one-data-set-out cross-validation we estimated the optimal number of trees and the maximum proportion of the training set in the leaf node for I-RF. Best results are observed with 100 trees and at most 10% of meta-training instances in the leaf nodes.

Meta-hyperparameters of our own methods are estimated in the very same way. Their values are reported in the corresponding chapters.

Part II

Meta-Learning for Bayesian Optimization

4

Surrogate Model-Free Hyperparameter Optimization

In the Chapter 2, different approaches for optimizing hyperparameters have been introduced. They all have in common that they use surrogate models in combination with an acquisition function. The surrogate models are learned by minimizing a regression loss and do not directly optimize for the hyperparameter optimization. Only in combination with a heuristic, i.e. the acquisition function, a useful search is possible.

There are two problems that we approach in this chapter: i) the choice of the hyperparameter configuration depends on a model optimized for the squared error combined with a simple heuristic and ii) the similarity between data sets depends on meta-features. Item i) is a problem because it does not find the best hyperparameter configuration directly and item ii) is a problem because meta-features do not guarantee that they are descriptive for the data set and are problem-dependent. We want to overcome these problems by proposing a model- and meta-feature-free hyperparameter optimization strategy that is optimized for a hyperparameter optimization loss. Its benefit is shown empirically in comparison to state of the art model-based optimization strategies that optimize its models for regression losses combined with heuristics. Since a hyperparameter optimization loss was never introduced before, we propose a cost measure for hyperparameter optimization. We use it to optimize our method for but it also enables a better comparison of optimization strategies across papers compared to strategies such as average rank. Because we do not want to rely on meta-

features, a distance measure between data sets that does not depend on meta-features is proposed. We empirically show that this new distance measure is able to improve existing hyperparameter optimization strategies. That is, we apply and evaluate our distance measure on existing model-based optimization strategies and compare it to the currently used meta-feature-based distance functions.

Our final contribution is interesting for practitioners that want to tune their hyperparameters easily and efficiently. So far, a random hyperparameter search is the easiest hyperparameter optimization strategy if a grid search is not possible. We demonstrate empirically that a *static* hyperparameter optimization strategy can outperform the random search substantially and that it is also very competitive in comparison to more complex state of the art hyperparameter optimization strategies. Just as random search, it is fully parallelizable. Because authors that are using random search for hyperparameter optimization often only report the distribution from which the hyperparameter combinations are sampled, reproducibility is only possible to some degree. Contrary, our static strategy is 100% reproducible just as it is the case with a grid search. In Appendix A, we provide look-up tables for two classifiers: Support Vector Machine (with linear, polynomial and RBF-kernel) and AdaBoost.

4.1 Contributions

1. We propose surrogate model-free optimization (SMFO), a variant of Bayesian optimization that uses one policy which is optimized for the task of hyperparameter optimization instead of a surrogate model combined with an acquisition function. This policy generalizes over previous experiments but neither uses a model nor uses meta-features. Nevertheless, it outperforms the state of the art.

2. Furthermore, we propose a similarity measure for data sets that yields more comprehensible results than those using meta-features. We show how this similarity measure can be applied to surrogate models in Bayesian optimization and empirically show that this change leads to better hyperparameter configurations in less trials.

3. We show that a static ranking of hyperparameter configurations yields competitive results and substantially outperforms a random hyperparameter search. Thus, it is a fast and easy alternative to complex hyperparameter optimization strategies and allows practitioners to tune their hyperparameters by simply using a look-up table. We made look-up tables for two classifiers publicly available: SVM and AdaBoost.

4.2 A New Evaluation Metric

It is very common to compare different optimization strategies either by an average rank between the methods over different data sets or by the performance of the found configurations on the data sets

directly[4,87]. Both comparisons make absolutely sense but come with some disadvantages. First of all, for both evaluation strategies you can either compare the plots depending on the number of tries done so far or you fix the number of tries to T and report the average rank after T tries. Additionally, the average rank between optimization strategies changes if you add or remove strategies. Hence, the values cannot be reused in further publications. Furthermore, if one optimization strategy has a better average rank than another, it does not necessarily mean that it is also the better strategy in general. Lets assume there are three data sets where the best hyperparameter configuration needs to be found. Optimization strategy A finds better hyperparameter configuration than strategy B for two of them but does not find any useful configuration for the third data set at all. Strategy B reliably finds very good configurations for all three data sets. By average rank, strategy A has a rank of 1.3 compared to strategy B with rank 1.7. This pairwise ranking does not take into account that strategy B provides reliably good configurations. While strategy A might have found configurations on two data sets that yield small improvement over the configurations found by strategy B, strategy B clearly outperforms strategy A on the third data set. Hence, the average rank does not tell you anything how big the differences with respect to the loss are. However, this is important as explained in the previous example.

We want to overcome both disadvantages. The evaluation measure should be a number that rewards fast convergence to the best hyperparameter configuration and it should be invariant to the performance of other methods in comparison. Thus, it can be easily compared to new methods, if they are applied on the same meta-data set. At this point, one could think about a ranking measure such as normalized discounted cumulative gain (NDCG). But we are not interested in finding the perfect ranking of hyperparameter configurations (this means first trying the best hyperparameter configuration, then the second best and so on) but on finding a decent hyperparameter configuration as soon as possible. Additionally, after finding the best hyperparameter configuration, the choice of further hyperparameter configurations should not affect the metric.

We propose the area under the average distance to the global minimum curve (AUC-ADTM) as a new metric which we define as

$$\text{AUC-ADTM}\left(\mathcal{D}, \mathcal{X}_T\right) = \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \sum_{t=1}^{T} \frac{\min_{\mathbf{x} \in \mathcal{X}_t} f_D\left(\mathbf{x}\right) - f_D^{\min}}{f_D^{\max} - f_D^{\min}} \quad , \tag{4.1}$$

where $\mathcal{D}$ is the set of data sets, $\mathcal{X}_T = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$ is the ordered sequence of evaluated hyperparameter configurations at time T, $\mathcal{X}_t$ the ordered subsequence of $\mathcal{X}_T$ until time $t < T$, $f_D\left(\mathbf{x}\right)$ is the error of the hyperparameter configuration $\mathbf{x}$ on the validation partition of data set D and $f_D^{\max}$ and $f_D^{\min}$ are the maximum and minimum value of f_D, respectively. Figure 4.1 visualizes the definition of this metric. This metric can only be applied if the hyperparameter optimization is limited to a

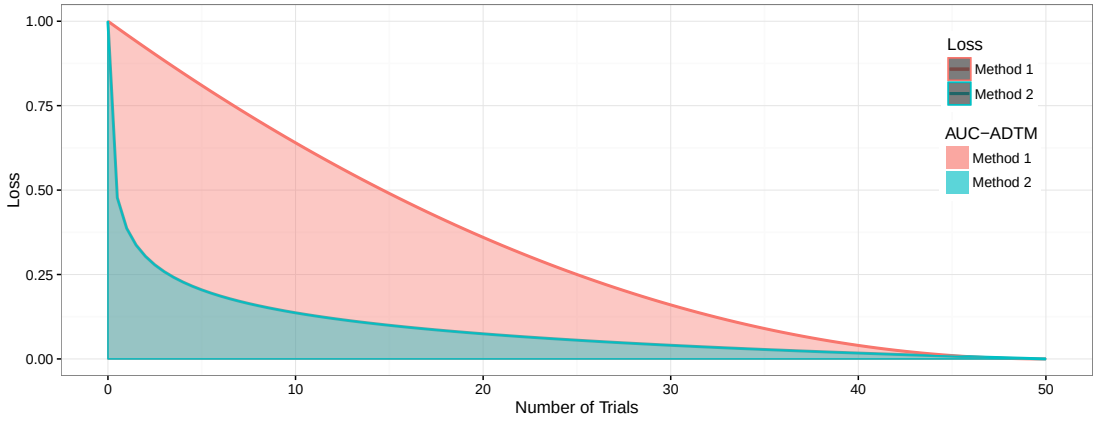

Figure 4.1: AUC-ADTM is the area under the loss curves. Since Method 1 is converging slower than Method 2, its AUC-ADTM is larger.

finite subset of all possible hyperparameter configurations. In real applications, the number of hyperparameter configurations is often infinite but for the evaluation of hyperparameter optimization strategies it is common that the meta-data is generated by applying a grid search using a finite set of hyperparameter configurations that is used on all data sets.

AUC-ADTM is lower bounded by 0 and attains this value only if in the first try on every data set the best hyperparameter configuration was chosen. The value of $f_D(\mathbf{x})$ is scaled between 0 and 1 per data set in Equation (4.1) to overcome different scales between data sets.

4.3 Surrogate Model-free Optimization

This section introduces one of the core contribution of this chapter. With A-SMFO we propose a fast and parallelizable hyperparameter optimization strategy that can be applied easily. This will be extended to NN-SMFO which uses information about the similarity between past seen data sets. Both methods directly optimize for AUC-ADTM, the evaluation metric discussed in the previous section. To this end, we propose a new distance function to measure the similarity.

4.3.1 Average SMFO

Our idea is to combine the surrogate model with the acquisition function. Additionally, we want a method that is independent of any meta-features. Given a new, unknown data set, the best configuration recommendation is the hyperparameter configuration that has proven to be best on average in past experiments. This intuition is extended to arbitrary many tries. Formally, at time step t, we choose the hyperparameter configuration $\mathbf{x}$ that improves the utility on the meta-training set the

Algorithm 2 AUC-ADTM Optimizing Sequence

Input: Set of feasible hyperparameter configurations $\mathcal{X}$, set of data sets from previous experiments $\mathcal{D}^{\text{train}}$, observation history $\mathcal{H}$, number of maximal tries T.
Output: Sequence of hyperparameter configurations to evaluate.

```
function OptimalSequence(X, T, D^train)
    X_0 ← ()
    for t = 1 to T do
        x_t ← arg min_{x∈X} c(x, X_{t−1}, D^train)
        X_t ← (X_{t−1}, x_t)
        if (1/|D^train|) Σ_{D∈D^train} min_{x∈X_t} r_D(x, X) = 1 then
            return X_t
        end if
    end for
    return X_T
end function
```

most. Let $\mathcal{X}_t$ be the set of evaluated hyperparameter configurations at step t. Then, we define the cost of a hyperparameter configuration $\mathbf{x}$ at t as

$$c\left(\mathbf{x}, \mathcal{X}_{t-1}, \mathcal{D}^{\text{train}}\right) = \sum_{D \in \mathcal{D}^{\text{train}}} \mathcal{L}\left(\mathbf{x}, \mathcal{X}_{t-1}, D\right) \quad . \tag{4.2}$$

Since we are facing a ranking problem, an obvious choice is a loss function based on the normalized discounted cumulative gain (NDCG). The NDCG is defined as the discounted cumulative gain

$$\text{DCG@k} = \sum_{i=1}^{k} \frac{2^{\text{rel}(i)} - 1}{\log_2\left(i+1\right)} \tag{4.3}$$

normalized by the maximum possible DCG, the ideal DCG (IDCG), where $\text{rel}\left(i\right)$ is the relevance score of the item which was predicted to be the i most relevant item. Then, we can set the loss function to

$$\mathcal{L}\left(\mathbf{x}, \mathcal{X}_{t-1}, D\right) = 1 - \frac{1}{\text{IDCG@k}} \sum_{i=1}^{k} \frac{2^{\text{rel}(i,D)} - 1}{\log_2\left(i+1\right)} \quad . \tag{4.4}$$

To evaluate the NDCG, a predicted ranking is needed. For a data set D, we predict that the hyperparameter configurations $\{\mathbf{x}\} \cup \mathcal{X}_{t-1}$ are most relevant, ordered increasingly by their f value. These f values are given in our meta-data set and do not need to be estimated. All hyperparameter configurations that are not in this set are predicted to be least relevant in arbitrary order. We argue that we are only interested in the best hyperparameter configurations and thus use the NDCG@1

by setting $k = 1$. Our loss function simplifies to

Algorithm 3 Average SMFO

Input: Set of feasible hyperparameter configurations $\mathcal{X}$, set of data sets from previous experiments $\mathcal{D}^{\text{train}}$, observation history $\mathcal{H}$, number of maximal tries T.
Output: Sequence of hyperparameter configurations to evaluate.

1: $\hat{\mathcal{X}} \leftarrow ()$
2: **while** $T > 0$ **do**
3: $\quad \mathcal{X}' \leftarrow \text{OPTIMALSEQUENCE}\left(\mathcal{X} \setminus \hat{\mathcal{X}}, T, \mathcal{D}^{\text{train}}\right)$
4: $\quad T \leftarrow T - |\mathcal{X}'|$
5: $\quad \hat{\mathcal{X}} \leftarrow \left(\hat{\mathcal{X}}, \mathcal{X}'\right)$
6: **end while**
7: **return** $\hat{\mathcal{X}}$

$$\mathcal{L}\left(\mathbf{x}, \mathcal{X}_{t-1}, D\right) = 1 - \frac{2^{\text{rel}(1,D)} - 1}{\text{IDCG@1}} \quad . \tag{4.5}$$

Replacing relevances by ranks and ignoring constants, we derive our final cost function

$$c\left(\mathbf{x}, \mathcal{X}_{t-1}, \mathcal{D}^{\text{train}}\right) = \sum_{D \in \mathcal{D}^{\text{train}}} \min_{\mathbf{x}' \in \mathcal{X}_{t-1} \cup \{\mathbf{x}\}} \left\{ r_D\left(\mathbf{x}', \mathcal{X}\right) \right\} \quad , \tag{4.6}$$

where $r_D\left(\mathbf{x}, \mathcal{X}\right)$ is the rank of $\mathbf{x}$ on data set D over all hyperparameter configuration in $\mathcal{X}$. Again, $\mathcal{X}$ is the finite set of feasible hyperparameter configurations as defined above that were evaluated on $D \in \mathcal{D}^{\text{train}}$ in previous experiments. The ranking for a new data set is obviously unknown. Then, at each time step t, $f\left(\arg\min_{\mathbf{x} \in \mathcal{X}} c\left(\mathbf{x}, \mathcal{X}_{t-1}, \mathcal{D}^{\text{train}}\right)\right)$ is evaluated. Algorithm 2 will find the best hyperparameter configurations for the meta-training set by sequentially selecting the hyperparameter configurations that minimize Equation (4.6) given $\mathcal{X}$. The stopping criterion in Line 5 is fulfilled as soon as $\mathcal{X}_t$ contains all hyperparameter configurations that were best on the data sets $D \in \mathcal{D}^{\text{train}}$. The resulting sequence $\mathcal{X}_T$ is optimal for the meta-training set but not necessarily for the meta-testing set. Actually, it is likely that more trials are needed on meta-test than on meta-train which means that the resulting sequence returned by Algorithm 2 is too short. To overcome this problem, the final optimization strategy is given in Algorithm 3. The idea is to minimize the cost on meta-train in the first iteration. In the next iterations, the cost are minimized by not considering previously chosen hyperparameter configurations in the evaluation and in the pool of feasible candidates $\mathcal{X}$. This is equivalent to optimizing for NDCG@t in iteration t instead of optimizing for NDCG@1. The assumption is that there is a relationship between hyperparameters across data sets, meaning

that a good hyperparameter configuration on one data set is also a potentially good candidate for another data set and the best might be located in its neighborhood. This assumption is also made in Bayesian optimization using meta-learning where this relationship is attempted to be learned using surrogate models such as Gaussian processes.

Algorithm 4 Nearest Neighbor AUC-ADTM Optimal Sequence

Input: Set of feasible hyperparameter configurations $\mathcal{X}$, set of data sets $\mathcal{D}^{\text{train}}$, meta-test data set D_{new}, number of maximal tries T, number of nearest neighbors k.
Output: Sequence of hyperparameter configurations to evaluate.

$\mathcal{X}_0 \leftarrow ()$
for $t = 1$ to T **do**
  Compute the subset $\mathcal{D}' \subseteq \mathcal{D}^{\text{train}}$ of the k most similar data sets to D_{new}.
  $\mathbf{x} \leftarrow \arg\min_{\mathbf{x} \in \mathcal{X}} c(\mathbf{x}, \mathcal{X}_{t-1}, \mathcal{D}')$
  $\mathcal{X}_t \leftarrow (\mathcal{X}_{t-1}, \mathbf{x})$
  if $\frac{1}{|\mathcal{D}^{\text{train}}|} \sum_{D \in \mathcal{D}^{\text{train}}} \min_{\mathbf{x} \in \mathcal{X}_t} r_D(\mathbf{x}, \mathcal{X}) = 1$ **then**
    return $\mathcal{X}_t$
  end if
end for
return $\mathcal{X}_T$

4.3.2 Nearest Neighbor SMFO

Average SMFO acquires hyperparameter configurations only using the data sets from previous experiments $D \in \mathcal{D}^{\text{train}}$. This means, there is a fixed sequence of hyperparameter configurations to evaluate and hence this method can be parallelized and implemented easily. Prediction can be done in constant time, memory consumption is linear in the number of feasible hyperparameter configurations and this static sequence of hyperparameter configurations can be shared among researchers and practitioners. Nevertheless, Average SMFO has one big disadvantage. It does not consider the informative evaluations on the current data set D_{new}. Therefore, Average SMFO is extended to Nearest Neighbor SMFO (NN-SMFO) to overcome this problem. NN-SMFO is not considering every data set when predicting the best hyperparameter configuration but the k most similar to D_{new}. The distance function between D_{new} and any other data set $D \in \mathcal{D}^{\text{train}}$ is inspired by the Kendall tau rank correlation[41] coefficient. Assuming that the ranking of hyperparameter configurations contains solely concordant or discordant pairs and ignoring the constants, the resulting KTRC

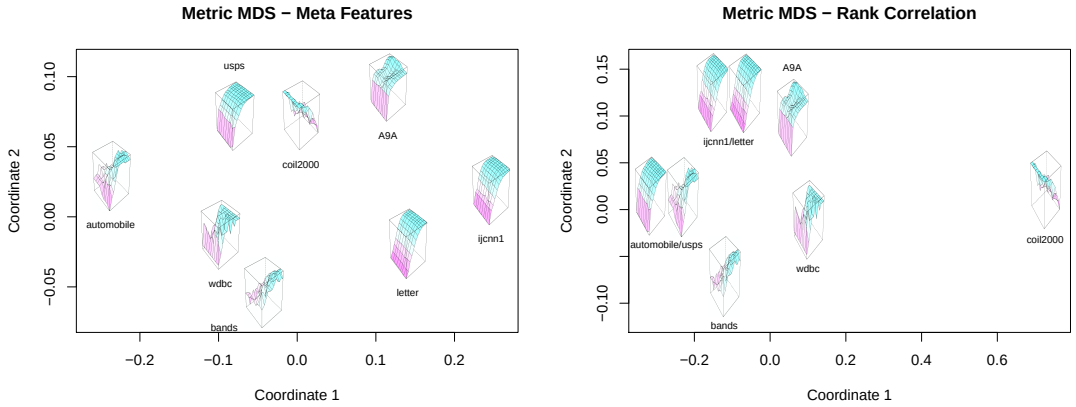

Figure 4.2: Metric multidimensional scaling of a distance metric using Euclidean distance on the meta-features (left) and Equation (4.7) using only the first four hyperparameter configurations recommended by Average SMFO (right). The shown response functions are that from an AdaBoost classifier. The meta-features used are described in Section 4.5.

distance function is

$$\mathrm{KTRC}\left(D_1, D_2, \mathcal{X}_t\right) = \begin{cases} \frac{\sum_{\mathbf{x}_1 \in \mathcal{X}_t} \sum_{\mathbf{x}_2 \in \mathcal{X}_t} s(\mathbf{x}_1, D_1, \mathbf{x}_2, D_2)}{(|\mathcal{X}_t| - 1)|\mathcal{X}_t|} & |\mathcal{X}_t| \geq 2 \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

$$\mathcal{X}_t = \mathcal{X}_t^{(D_1)} \cap \mathcal{X}_t^{(D_2)} \tag{4.8}$$

$$s\left(\mathbf{x}_1, D_1, \mathbf{x}_2, D_2\right) = \mathbb{I}\left(\left(f_{D_1}\left(\mathbf{x}_1\right) > f_{D_1}\left(\mathbf{x}_2\right)\right) \oplus \left(f_{D_2}\left(\mathbf{x}_1\right) > f_{D_2}\left(\mathbf{x}_2\right)\right)\right) \quad , \tag{4.9}$$

where $\oplus$ is the symbol for an exclusive or and $\mathcal{X}_t^{(D)}$ is the set of hyperparameter configurations $\mathbf{x} \in \mathcal{X}$ where the response function $f_D\left(\mathbf{x}\right)$ has already been evaluated for data set $D \in \mathcal{D}^{\text{train}}$.

To implement this method, Line 3 in Algorithm 3 needs to be changed such that it calls Algorithm 4 instead of Algorithm 2. We cannot compute a useful KTRC for less than two observations on the new data set D_{new}. In that case, we consider all data set as most similar to D_{new}.

4.4 On a Distance Measure Between Data Sets

The current state of the art models estimate the similarity between data sets either by learning it implicitly using meta-features[4,35] or modelling them explicitly using the Euclidean distance on the meta-features[5,8,87]. What is meant with similar data sets is that they behave similarly with respect to the hyperparameter configurations. This means, that two similar data sets have a similar ranking of hyperparameter configurations. Measuring this ranking using a rank correlation metric such as proposed in Equation (4.7) is actually a natural choice. Otherwise, the similarity cannot be estimated

if no evaluations are observed. We claim that few evaluations are enough to approximate the true rank correlation such that this distance measure is nevertheless more expressive than alternatives such as distance functions based on meta-features. Figure 4.2 supports this claim.

In the left plot, it shows the multidimensional scaling (MDS) of the distance matrix using the Euclidean distance on the meta-features. The data set names indicate the position in the space, the near-by little plots the (hidden) response function. The response function in this case maps two hyperparameters (x and z-axis) of f, in this case the classification accuracy. One can see that this similarity measure works in some cases. But the data sets *usps* and *coil2000* provide an example that demonstrate that there are also cases where it does not work. These data sets are very close to each other with respect to the similarity metric but the hyperparameter configurations are almost ranked contrary.

In the right plot the KTRC distance function defined in Equation (4.7) is used. The distance is computed using only four hyperparameter configurations, i.e. the best four recommended by Average SMFO. Clearly, *coil2000* is identified as completely different to any other data set and the reader may see for herself that the distances are more intuitive just from visual inspection. The conclusion is that only little information about the ranking can already provide a good clustering.

4.4.1 Rank Correlation-based Gaussian Process

After discussing the quality of different distance functions used for data set similarities, we want to show how to apply the new distance function to Bayesian optimization.

Yogatama and Mann[87] introduced the multiple kernel framework to Bayesian optimization for machine learning. They proposed a linear combination of a squared exponential kernel with automatic relevance determination (SE-ARD) for points in the same data set and a nearest neighbor kernel for modeling similarities between data sets. They defined the kernel as

$$\begin{aligned} k\left((\mathbf{x}, D), \left(\mathbf{x}', D'\right)\right) &= \alpha \mathbb{I}\left(D = D'\right) k_{\text{SE-ARD}}\left(\mathbf{x}, \mathbf{x}'\right) \\ &\quad + (1-\alpha)\, \mathbb{I}\left(D' \in \mathcal{N}(D)\right) k_{\text{NN}}\left(\mathbf{x}, \mathbf{x}'\right) \quad , \end{aligned} \tag{4.10}$$

where the SE-ARD kernel is defined as

$$k_{\text{SE-ARD}}\left(\mathbf{x}, \mathbf{x}'\right) = \exp\left(-\frac{1}{2} \sum_p \frac{\left(\mathbf{x}_p - \mathbf{x}'_p\right)^2}{\sigma_p^2}\right) \tag{4.11}$$

and the data set similarity kernel as

$$k_{\text{NN}}\left(\mathbf{x}, \mathbf{x}'\right) = 1 - \frac{1}{B}\left\|\mathbf{x} - \mathbf{x}'\right\| \quad , \tag{4.12}$$

where B must be chosen such that k_{NN} is always non-negative. $\mathcal{N}(D)$ denotes the set of neighbored data sets with respect to a distance function. The distance between two data sets is defined as the Euclidean distance between its meta-features.

Since we want to measure the impact of a different data set distance measure, we use exactly the same setup but use the distance function defined in Equation (4.7) to determine the neighborhood. We call this surrogate model Rank Correlation-based Gaussian Process.

4.5 Experimental Evaluation

The optimization strategies proposed in Section 4.3 are arguably simple and still amazingly competitive. We are comparing them to state of the art hyperparameter optimization strategies (competitors are published on top conferences, e.g. NIPS 2012, ICML 2013, AISTATS 2014). Empirically, we show that they can outperform those strategies and have the capability to scale to big meta-data.

4.5.1 Experimental Setup

This is the only section where we do not follow the setup as described in Section 3.2.2. This has various reasons. First, we are using all meta-instances for the training phase. The reason for this is that SMFO cannot recommend configurations it has never seen before. With the increase of meta-instances, the size of the training data set increased exponentially. Thus, we were not able to conduct the experiments for SCoT and MKL-GP. Thus, we selected 25 of our 50 data sets at random to conduct our experiments. Finally, we added only the meta-features to our meta-data that were also used by the authors of SCoT and MKL-GP[4,87]. First, we are extracting the number of training instances n, the number of classes c and the number of predictors p. The final meta-features are c, $\log(p)$ and $\log(n/p)$ scaled to $[0, 1]$. These are the same meta-features as originally used by Yogatama[87] and allows us to redo exactly their setup. Otherwise, nothing else changed compared to Section 3.2.2.

Table 4.1: Brief overview of the competitor methods. An explanation of this table is given in Section 3.2.3.

Name	Abbrev.	Surrogate Model	Meta-Knowledge	Details
Random Search	Random	None	✗	Sec. 2.2
Independent Gaussian Process	I-GP	Gaussian Process	✗	Sec. 2.7.1
Independent Random Forest	I-RF	Random Forest	✗	Sec. 2.7.1
Surrogate Collaborative Tuning	SCoT	SVMRank + Gaussian Process	✓	Sec. 2.7.2
Gaussian Process with Multi-Kernel Learning	MKL-GP	Gaussian Process	✓	Sec. 2.7.2

4.5.2 Configuration Optimization Strategies

The optimization strategies introduced in Section 4.3 and 4.4.1 are compared to state of the art optimization strategies. We summarized our competitor methods in Table 4.1. We compare the competitor methods to the methods proposed in this chapter:

Rank Correlation-based Gaussian Process (RC-GP) This is the variation of MKL-GP proposed in Section 4.4.1. It is exactly the same as MKL-GP but uses the distance function from Equation (4.7).

Average SMFO (A-SMFO) This is the optimization strategy introduced in Section 4.3. It chooses the hyperparameter configurations in the particular order learned from previous experiments without the use of a surrogate model. The feedback on the current experiment is not taken into account. It is trivially parallizable and is the only strategy in our experimental evaluation that has this property apart from random search.

Nearest Neighbor SMFO (NN-SMFO) NN-SMFO is very similar to Average SMFO but it takes into account the feedback on the current experiment. The first two iterations are always the same as in Average SMFO. Afterwards, the ranking of the hyperparameter configurations on the current experiment is used to estimate the nearest neighbored data sets and the Average SMFO is then estimated on this subset of data sets. We estimated the best size of the neighborhood with a leave-one-data-set-out cross-validation. This procedure identified 2 as the optimal size.

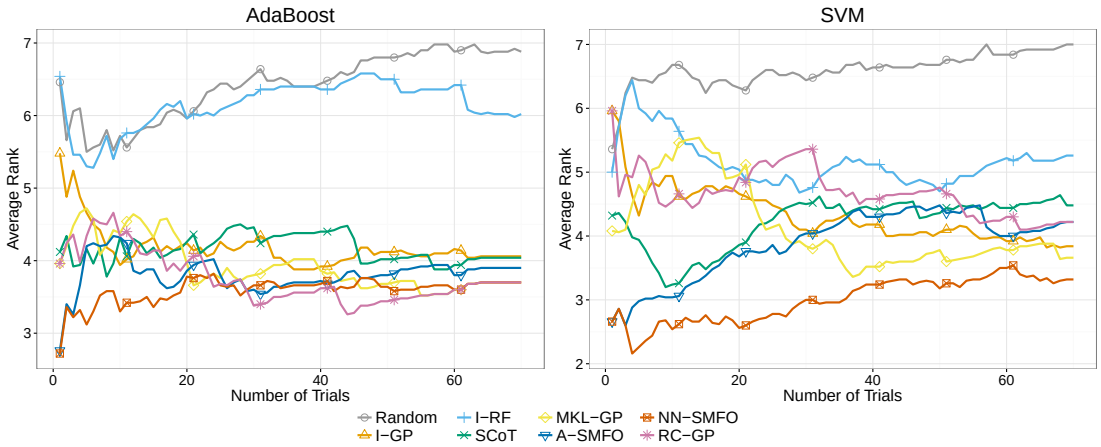

Figure 4.3: Development of the average rank among different hyperparameter tuning strategies with increasing number of trials. NN-SMFO shows strong performance especially on the SVM meta-data set.

The results reported are the average of at least ten repetitions. For the strategies with random initialization (Random, I-GP and I-RF), the mean of 1,000 repetitions is reported.

4.5.3 Results

Our hypotheses are:

1. The strategies proposed in Section 4.3 are better because they are optimized directly for AUC-ADTM.

2. The KTRC distance measure proposed in Equation (4.7) provides better information about the distance between data sets than a distance function based on the meta-features.

To confirm the first hypothesis, consider Figure 4.3. It shows the development of the average rank among different hyperparameter optimization strategies. On the AdaBoost meta-data set all optimization strategies but Random and I-RF are close together. Nevertheless, NN-SMFO is one of the best. Especially in the beginning, a larger gap to the other approaches is recognizable. Starting at iteration 30, RC-GP is taking over the lead. The good performance of NN-SMFO becomes substantial on the SVM meta-data set. Here it takes about 40 trials until any other optimization strategy gets even close to the performance of NN-SMFO. A-SMFO performs worse with respect to the average rank than NN-SMFO but still performs amazingly well considering that it is a static sequence and does not use any information about the data set that is currently being investigated. We want to remark that the performance of the optimization strategies will always converge to the same value since at some point they have tried all feasible hyperparameter configurations.

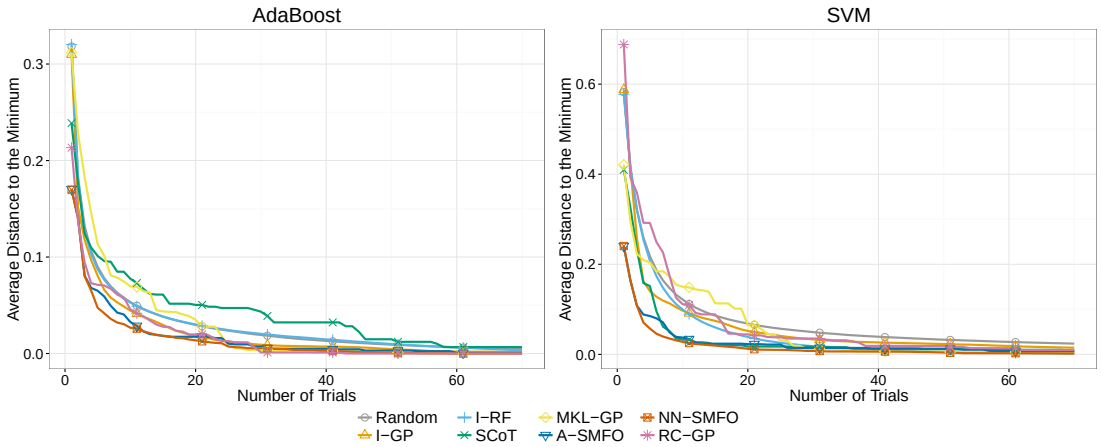

Figure 4.4: Development of the average distance to the global minimum with increasing number of trials. RC-GP quickly finds the best configuration on the AdaBoost meta-data set. NN-SMFO provides good results on the AdaBoost meta-data set and the best results on the SVM meta-data set.

Having a look at the development of the average distance to the global minimum in Figure 4.4 gives the reader the impression how fast the optimization strategies actually converge to the best hyperparameter configuration on average. One can see that both, A-SMFO and NN-SMFO, are converging considerably faster than the other strategies. Only 8 trials are needed to cross the 5% average distance to the global minimum threshold. Again, this marked difference is more substantial on the SVM meta-data set. Our assumption is that the SVM meta-data set is more difficult because it contains more than twice that many feasible hyperparameter configurations and the SVM has more hyperparameters to tune.

Finally, also the results of the third metric presented in Table 4.2 lead to the same conclusions.

To validate the second hypothesis, we have a closer look at the optimization strategies MKL-GP (yellow diamond) and RC-GP (pink star) in the previously discussed figures because the only difference between those two is the distance function.

Again, we first compare them with respect to the average rank. On the AdaBoost meta-data set, RC-GP is having a small advantage over MKL-GP. On the SVM data set, MKL-GP is looking weak in the beginning but finally outperforms RC-GP.

Comparing RC-GP and MKL-GP with respect to the average rank does not lead to an unambiguous conclusion. RC-GP performs better on the AdaBoost meta-data set and MKL-GP is better on the SVM meta-data set. Unfortunately, similar results are observed with respect to the other two evaluation metrics.

Table 4.2: Summary of the AUC-ADTM of the optimization strategies on both meta-data sets. Number in brackets indicate the ranking across the strategies. The strategies introduced in this chapter are bold.

	Random	I-GP	I-RF	SCoT
AdaBoost	0.089 (7)	0.065 (4)	0.088 (6)	0.120 (8)
SVM	0.280 (8)	0.254 (7)	0.145 (4)	0.115 (3)
	MKL-GP	**RC-GP**	**A-SMFO**	**NN-SMFO**
AdaBoost	0.082 (5)	**0.052 (3)**	**0.047 (2)**	**0.040 (1)**
SVM	0.174 (5)	**0.210 (6)**	**0.082 (2)**	**0.053 (1)**

4.5.4 Remarks on the Proposed Evaluation Measure

Now, we want to continue the discussion about the usefulness of the average rank between the optimization strategies as an evaluation measure. We argue in Section 4.2 that the average rank may lead to wrong conclusions with respect to the speed of convergence. In our experiments, SCoT may be the most drastic example. While it shows good performance with respect to the average rank for the AdaBoost meta data-set, the average convergence, i.e. the average distance to the global minimum, is clearly the worst (see Figure 4.4 (left)). Then again, random search shows mediocre performance with respect to average distance to the global minimum but is clearly the worst with respect to average rank. The reason is simple: the performance of random search is independent of the investigated data set. In expectation, it is always the same. Hence, random search is a very stable strategy. Otherwise, SCoT can be very good on some data sets finding the best hyperparameter within the first few iterations while it does not find any useful hyperparameter at all on all other data sets.

Our conclusion to this discussion is that the AUC-ADTM provides the reader more concrete insight. In the end, what matters is how fast the strategies converge on average and how reliable they are. We want to stress the point that the choice of the evaluation criterion does not affect the verity of our hypotheses.

4.5.5 Running Time

The idea of the hyperparameter optimization strategies is to speed up the optimization process. Therefore, the overhead for carefully selecting hyperparameter configurations has to be small. In Figure 4.5 we report the overhead of one hyperparameter selection process in milliseconds for the AdaBoost meta-data set. Our implementation is not optimized for speed which needs to be considered especially for MKL-GP and RC-GP. Both strategies use a Gaussian process with a very sparse

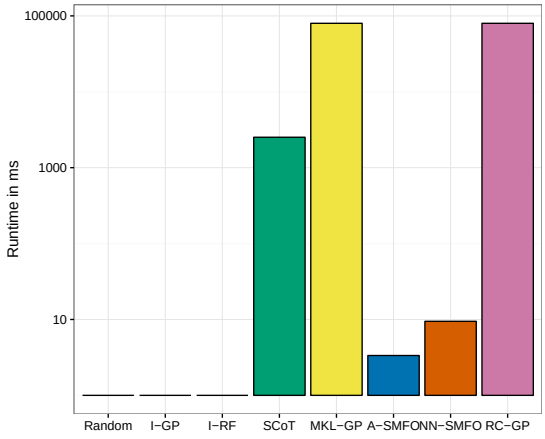

Figure 4.5: Running time of the various optimization strategies for the AdaBoost meta-data set in milliseconds on a logarithmic scale.

kernel as a surrogate model. The speed for updating the surrogate model can be significantly decreased if the sparseness is exploited. Yogatama and Mann [87] have shown that MKL-GP is actually faster than SCoT.

Even though some methods are faster than others, the running time overhead is a matter of seconds. Otherwise, testing a single hyperparameter configurations can be a matter of hours or days. Creating the AdaBoost meta-data set took us 582 CPU days while the total overhead for the same number of hyperparameter decisions is 2.5 CPU days for the slowest optimization strategy. Thus, our conclusion is that all methods are feasible and their running times are negligible if the data set of interest and the applied model's complexity is just large enough. If this is not the case, usually a grid search is the best solution, anyway.

4.6 Conclusion

We introduced three new hyperparameter optimization strategies and demonstrated that hyperparameter optimization is also possible without surrogate models. A-SMFO has very nice properties. It is parallelizable which is only possible for grid search and random search while all other optimization strategies are not trivially parallelizable. Additionally, it only depends on the training meta-data and not on the currently evaluated data set. Therefore, we made the best predicted ranking for the models SVM and AdaBoost accessible to the community. This offers practitioners a static hyperparameter optimization strategy that has proven to be very competitive compared to state of the art and easy to apply for future experiments. Another nice property is that it makes results repro-

ducible. Reproducing random search is not exactly possible because often only the distribution over the hyperparameters is made public and not which hyperparameter configurations are finally chosen.

NN-SMFO is an improvement over A-SMFO and can be used to further improve the results. It was compared to important state of the art competitor strategies and has empirically shown to be effective.

Furthermore, a new distance measure between data sets was discussed. It was applied to a state of the art Bayesian optimization strategy and its improvement was empirically shown.

Finally, some discussions about a good quality measure for hyperparameter optimization strategies was conducted and an interesting new one was introduced.

5

Hyperparameter Search Space Pruning

Bayesian optimization is the current state of the art for automatic hyperparameter optimization. Currently, it consists of three components: a surrogate model, an acquisition function and an initialization technique. We propose to add a fourth component, a way of pruning the hyperparameter search space which is a common way of accelerating the search in many domains but yet has not been applied to hyperparameter optimization. We propose to discard regions of the search space that are unlikely to contain better hyperparameter configurations by transferring knowledge from past experiments on other data sets as well as taking into account the evaluations already done on the current data set.

Pruning as a new component for Bayesian optimization is an orthogonal contribution but nevertheless we compare it to surrogate models that learn across data sets and extensively investigate the impact of pruning with and without initialization for various state of the art surrogate models. The experiments are conducted on our SVM and Weka meta-data sets.

5.1 Introduction

Bayesian optimization currently has at most three components. First, the surrogate model that predicts the performance for each possible hyperparameter configuration. Secondly, the acquisition function which uses the surrogate model to propose the next hyperparameter configuration to evaluate. These are the two mandatory components. The third optional component is some initialization technique which usually starts with a hyperparameter configuration that has proven to be good on

other data sets[24,26]. We propose to add a fourth component which is orthogonal to all the others. Our idea is to reduce the hyperparameter search space by using knowledge from past experiments to discard regions that are very likely not interesting. This avoids that the acquisition function chooses hyperparameter configurations in these regions because of high uncertainty and therefore avoids unnecessary function evaluations.

Pruning is a well known technique to accelerate the search in several domains. For example, various pruning techniques are applied to the minimax algorithm such as the killer heuristic or null move pruning[17]. Branch-and-Bound[46] is a pruning technique that is applied in the domain of operations research for discrete and combinatorial optimization problems and is very common for NP-hard optimization problems[45]. Nevertheless, we are not aware of any published work that is trying to prune the search space in Bayesian optimization for hyperparameter optimization.

Since pruning as proposed by us is some way of transferring knowledge from past experiments to a new experiment, other techniques that try exactly the same are the closest related work but, as we will see, orthogonal to our contribution. One common and easy way to use experience in the hyperparameter optimization domain is to define an initialization, a sequence of hyperparameter configurations that are chosen first. These are usually those hyperparameter configurations that performed best on average across other data sets[24,26]. The second and last method to do so is by using the surrogate model. Instead of learning the surrogate model only on the new data set, the surrogate model is learned across all data sets[4,72,87]. We want to highlight that all these three possibilities are not mutually exclusive and can be combined so that these ideas are orthogonal to each other.

Leite et al.[47] propose a similar distance function between data sets but they propose a hyperparameter selection strategy that is limited to the hyperparameter configurations that have been seen on the meta-training data.

Furthermore, there also exist strategies to optimize hyperparameters that are based on optimization techniques from artificial intelligence such as tabu search[11], particle swarm optimization[28] and evolutionary algorithms[25] as well as gradient-based optimization techniques[14] designed for support vector machines.

5.2 Pruning the Search Space

The idea of pruning is to consider only a subset of the hyperparameter configuration space $\mathcal{X}$ to avoid unnecessary function evaluations in regions where we do not expect any improvements. If regions can be identified in advance, that are not of interested then this would obviously accelerate the hyperparameter optimization process. We propose to predict the utility of regions by transferring

knowledge from past experiments. The key idea is that similar data sets to the new data set have similar or even the same regions that are not interesting and therefore not worth to be investigated.

5.2.1 Formal Description

We define a region $R(\mathbf{x}, \delta)$ by its center $\mathbf{x} \in \mathcal{X}$ and radius $\delta \in \mathbb{R}^P,\ \delta > 0$. The utility of this region after t trials on the new data set D_{new} is defined by

$$u\left(R(\mathbf{x}, \delta), \mathcal{X}_t\right) = \sum_{D' \in \mathcal{N}(D_{\text{new}})} \min_{\mathbf{x}' \in \mathcal{X}_t} \tilde{f}_{D'}\left(\mathbf{x}'\right) - \tilde{f}_{D'}\left(\mathbf{x}\right) \quad , \tag{5.1}$$

where $\mathcal{X}_t$ is the set of already evaluated hyperparameter configurations on D_{new} and $\mathcal{N}\left(D_{\text{new}}\right)$ is the set of data sets that are closest to the new data set. $\tilde{f}_D$ is the normalized version of the response function f_D of data set D. f_D is scaled to the interval $[0, 1]$ such that each data set has the same influence on the utility. Thus, the utility is the predicted improvement when choosing $\mathbf{x}$ over the hyperparameter configurations already evaluated. Since f_D is not fully observed for $D \in \mathcal{D}$, where $\mathcal{D}$ is the meta-training set, we approximate $\tilde{f}_D$ with a plug-in estimator $\hat{y}_D$. We use a Gaussian process [57] that is trained on all normalized meta-instances of a data set such that we get for each training data set a plug-in estimator

$$\tilde{f}_D\left(\mathbf{x}\right) \approx \hat{y}_D\left(\mathbf{x}\right) \sim \mathcal{GP}\left(m_D\left(\mathbf{X}\right), k_D\left(\mathbf{X}, \mathbf{X}\right)\right) \quad , \tag{5.2}$$

where we define m_D as the mean function and k_D as the covariance function of $\tilde{f}_D$. As a covariance function we are using the squared exponential kernel

$$k\left(\mathbf{x}, \mathbf{x}'\right) = \exp\left(-\frac{\left\|\mathbf{x} - \mathbf{x}'\right\|_2^2}{2\sigma^2}\right) \quad . \tag{5.3}$$

This allows to estimate $\tilde{f}_D$ for arbitrary hyperparameter configurations. Then, we replace the definition from Equation (5.1) with

$$u\left(R(\mathbf{x}, \delta), \mathcal{X}_t\right) = \sum_{D' \in \mathcal{N}(D_{\text{new}})} \min_{\mathbf{x}' \in \mathcal{X}_t} \hat{y}_{D'}\left(\mathbf{x}'\right) - \hat{y}_{D'}\left(\mathbf{x}\right) \quad . \tag{5.4}$$

To estimate the nearest neighbors of the new data set D_{new}, we have to define a distance function between data sets. A common choice for this is the Euclidean distance with respect to the meta-features [4,87]. Since we experienced better results with the KTRC distance function proposed in the

previous chapter (Equation (4.7)), we use a modified version of it:

$$\text{KTRC}\left(D_1, D_2, \mathcal{X}_t\right) = \frac{\sum_{\mathbf{x}_1,\mathbf{x}_2 \in \mathcal{X}_t} \mathbb{I}\left(\hat{y}_{D_1}(\mathbf{x}_1) > \hat{y}_{D_1}(\mathbf{x}_2) \oplus \hat{y}_{D_2}(\mathbf{x}_1) > \hat{y}_{D_2}(\mathbf{x}_2)\right)}{(|\mathcal{X}_t|-1)|\mathcal{X}_t|} \quad . \tag{5.5}$$

The difference is that we use the plug-in estimators $\hat{y}$ which allow a comparison between partially unobserved pairs.

Algorithm 5 Prune

Input: Hyperparameter space $\mathcal{X}$, observation history $\mathcal{H}$, region radius δ, fraction of the pruned space ν.
Output: Pruned hyperparameter space $\mathcal{X}^{\text{pruned}} \subseteq \mathcal{X}$.

1: **function** PRUNE
2: Estimate the most similar data sets of the new data set using Equation (5.5).
3: Estimate the set $\mathcal{X}'$ containing the $\nu \cdot |G|$ hyperparameter configurations $\mathbf{x}' \in G \subset \mathcal{X}$ with little utility using Equation (5.4).
4: $\mathcal{X}^{\text{pruned}} := \{\mathbf{x} \in \mathcal{X} \mid \text{dist}\left(\mathbf{x}, \mathbf{x}'\right) > \delta,\ \mathbf{x}' \in \mathcal{X}'\}$.
5: **return** $\mathcal{X}^{\text{pruned}} \cup \{\mathbf{x} \in \mathcal{X} \mid \text{dist}\left(\mathbf{x}, \mathbf{x}'\right) \leq \delta, \mathbf{x}' \in \mathcal{X}_t\}$
6: **end function**

Algorithm 5 summarizes the pruning function. In the first line the k most similar data sets are estimated which we know from past experiments using the KTRC distance function defined in Equation (5.5). In Line 2, the utility of hyperparameter configurations are computed using the plug-in estimators (Equation (5.4)) on a fine grid $G \subset \mathcal{X}$. The $\nu \cdot |G|$ hyperparameter configurations with little utility define regions where no improvement is predicted. Hence, the pruned hyperparameter space is defined as the set of hyperparameter configurations that are not within a δ-region of these low-utility hyperparameter configurations (Line 3). Additionally, the hyperparameter configurations that are within a δ-region of already evaluated hyperparameter configurations are added (Line 4). The intuition here is that since we have already observed an evaluation in this region, the acquisition function will not choose a hyperparameter combination close to these points for exploration but only for exploitation. Hence, no evaluations will be done by Bayesian optimization without a very likely improvement. As a distance function between hyperparameter configurations we chose one that considers indicator variables. Indicator variables indicate, for example, which algorithm was chosen. Therefore, it is obvious that the change in the loss is likely not smooth. Therefore, we define the distance function used by Algorithm 5 as

$$\text{dist}\left(\mathbf{x}, \mathbf{x}'\right) = \begin{cases} \infty & \text{if } \mathbf{x} \text{ and } \mathbf{x}' \text{ differ in an indicator variable} \\ \|\mathbf{x} - \mathbf{x}'\| & \text{otherwise} \end{cases} \quad . \tag{5.6}$$

The extended Bayesian optimization with pruning is described in Algorithm 6.

Algorithm 6 Bayesian Optimization with Pruning

Input: Configuration space $\mathcal{X}$, observation history $\mathcal{H}$, acquisition function a, initial set of configurations $\mathcal{X}^{\text{init}}$.
Output: Best configuration found.

for $\mathbf{x} \in \mathcal{X}^{\text{init}}$ **do**
  $f_* = f(\mathbf{x}_*)$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\mathbf{x}_*, f_*)\}$
end for
while not converged **do**
  Update the surrogate model $p(f_*|\mathbf{x}_*, \mathcal{H})$.
  $\mathcal{X}^{\text{pruned}} \leftarrow \text{PRUNE}(\mathcal{X})$
  $\mathbf{x}_* = \arg\max_{\mathbf{x}_* \in \mathcal{X}^{\text{pruned}}} a(\mathbf{x}_*, p(f_*|\mathbf{x}_*, \mathcal{H}))$
  $f_* = f(\mathbf{x}_*)$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\mathbf{x}_*, f_*)\}$
  if $f_* < f^{\text{min}}$ **then**
    $\mathbf{x}^{\text{min}}, f^{\text{min}} \leftarrow \mathbf{x}_*, f_*$
  end if
end while
return $\mathbf{x}^{\text{min}}$

5.3 Experimental Evaluation

In this section we will compare our proposed pruning strategy to the competitor strategies summarized in Table 5.1 on the SVM and Weka meta-data set. We compare the different optimization strategies with respect to two metrics. We report the mean of at least ten repetitions. For the strategies with random initialization (Random, I-GP, I-RF), we report the mean of 1,000 repetitions. Details about the meta-data sets and the evaluation metrics can be found in Chapter 3.

5.3.1 Hyperparameter Optimization for SVMs

As described in Section 3.2.2, we do not use all 288 hyperparameter configurations for training but only 50 per data set. The evaluation is nevertheless done on all 288 of the new data set. We choose G to contain these 288 configurations and fixed $|\mathcal{N}(D_{\text{new}})| = 2$, $\nu = 1 - |G|^{-1}$ and δ such that the two closest neighbored hyperparameter configurations of the test region are within δ-distance.

We conduct two different experiments. First, we compare a surrogate model with pruning to current state of the art optimization strategies. Once again, we stress that pruning in Bayesian

Table 5.1: Brief overview of the competitor methods. An explanation of this table is given in Section 3.2.3.

Name	Abbrev.	Surrogate Model	Meta-Knowledge	Details
Random Search	Random	None	✗	Sec. 2.2
Independent Gaussian Process	I-GP	Gaussian Process	✗	Sec. 2.7.1
Independent Random Forest	I-RF	Random Forest	✗	Sec. 2.7.1
Surrogate Collaborative Tuning	SCoT	SVMRank + Gaussian Process	✓	Sec. 2.7.2
Gaussian Process with Multi-Kernel Learning	MKL-GP	Gaussian Process	✓	Sec. 2.7.2

optimization is an orthogonal contribution. Hence, these results are actually of minor interest. Second, we compare different surrogate models with and without pruning or initialization. Pruning is a useful contribution as long as it does not worsen the optimization speed in general and accelerates it in some cases.

Figure 5.1 shows the results of the comparison of pruning to the current state of the art method. We decided to choose the Gaussian process that is *not* learned across data sets as a surrogate model because it is the most common surrogate model. Surprisingly, the pruning alone with the Gaussian process is able to outperform all the competitor strategies with respect to all three evaluation metrics.

Figure 5.2 to 5.5 show the results of different surrogate models. We distinguish four different cases: i) only the surrogate model, ii) the surrogate model with pruning, iii) the surrogate model with three steps of initialization and iv) the surrogate model with three steps of initialization and pruning. Figure 5.2 and 5.3 show the results for the plain surrogate models and the remaining three Figures show the results for the transfer surrogate models. Our expectation before the experiments were that the lift is higher i) for the experiments without initialization and ii) for the experiments with plain surrogate models that do not learn across data sets. The reason for this is simple. An initialization is a fixed policy that proposes hyperparameter configurations that have been good on average, while pruning discards regions that were not useful. Thus, pruning will also have an effect of initialization. The difference between initialization and pruning is that an initialization proposes a specific hyperparameter configuration while pruning reduces the full hyperparameter space to a set of good hyperparameter configurations. Furthermore, pruning is applied at each iteration and

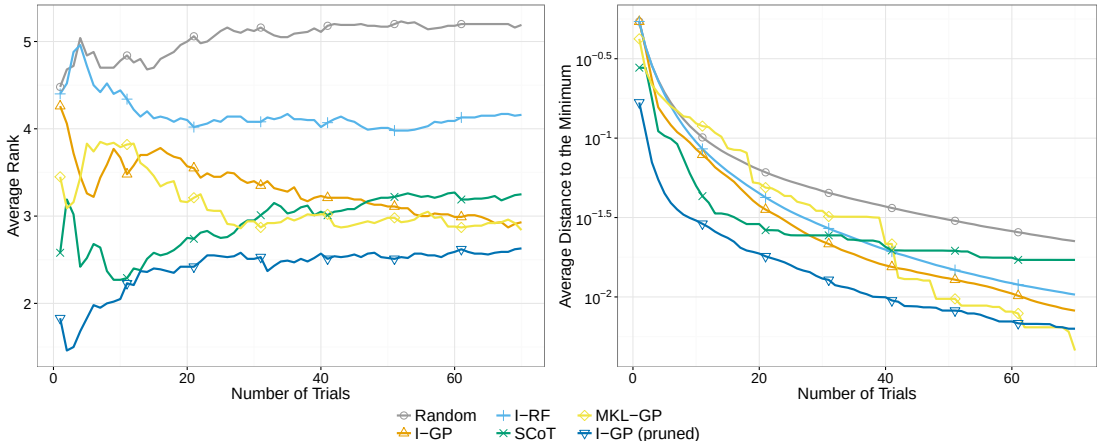

Figure 5.1: Pruning is an orthogonal contribution to Bayesian optimization. Nevertheless, we compare a pruned independent Gaussian process to many current state of the art optimization strategies without pruning.

not just for the initial iterations. Pruning is a way to transfer knowledge between data sets such that those strategies that do not use this knowledge at all benefit more and are prevented from conducting unnecessary exploration queries.

This is exactly what the results of the experiments show. The Bayesian optimization experiments with pruning have comparable good starting points like those with initialization. Compare the results of the independent Gaussian process (Figure 5.2) for the setting that only uses initialization with the one that only uses pruning. We clearly see the unnecessary exploration queries after a good start. The setting with both initialization and pruning does not suffer from this problem and thus is clearly the best strategy. This effect is weaker for the transfer surrogate models in Figure 5.5. Only for SCoT (Figure 5.4) pruning does not accelerate the hyperparameter optimization on *this* metadata set but it also does not worsen it. Table 5.2 shows the results for all evaluation metrics and surrogate models.

First, the results in the plot always converge to the same value across different optimization strategies if you allow only enough trials. Second, even a very small improvement of the performance *just* by choosing a better hyperparameter configurations is already a success, especially since this optimization is usually limited in time. This little improvement may result in significantly better results for a new model compared to the competitors or decides whether you win a research challenge or not.

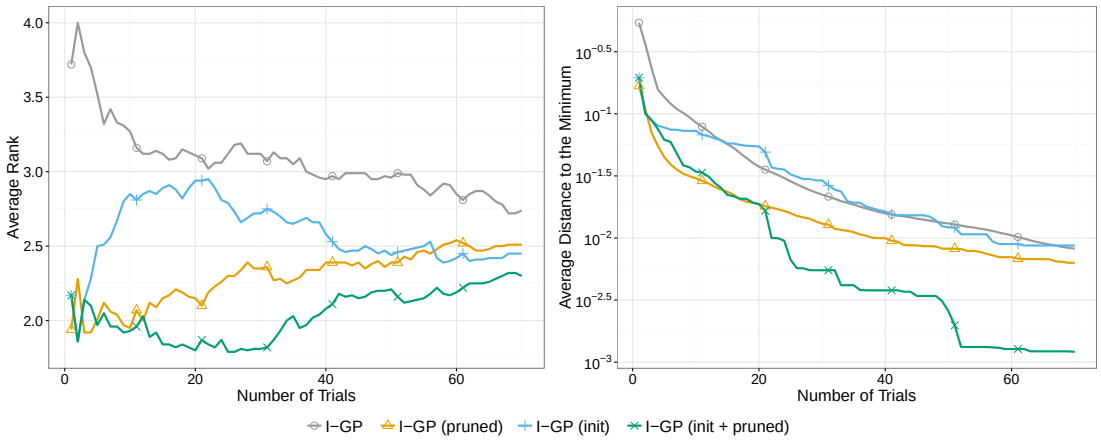

Figure 5.2: Average rank and average distance to the minimum for I-GP on the SVM meta-data set.

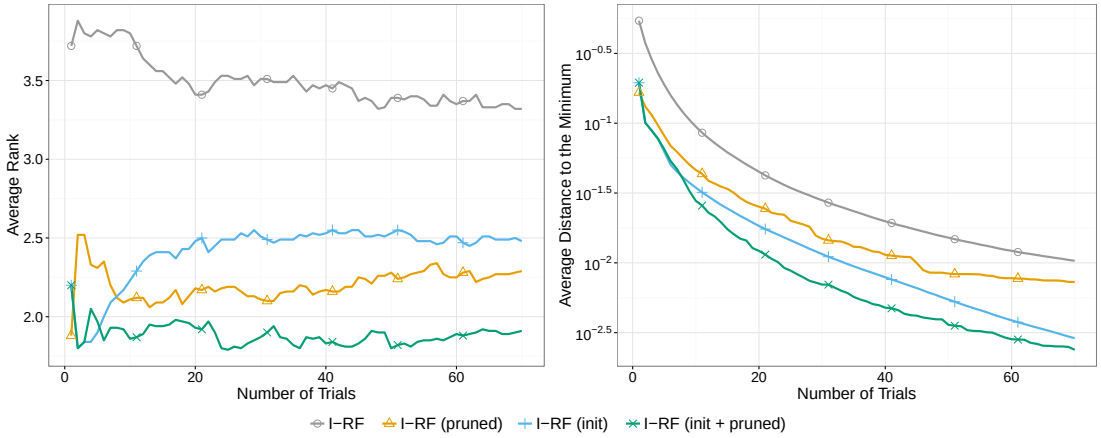

Figure 5.3: Average rank and average distance to the minimum for I-RF on the SVM meta-data set.

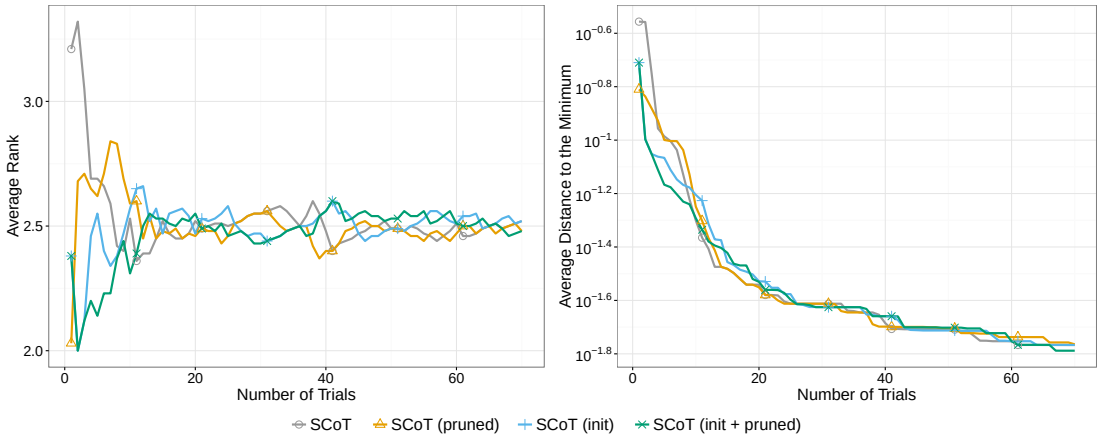

Figure 5.4: Average rank and average distance to the minimum for SCoT on the SVM meta-data set.

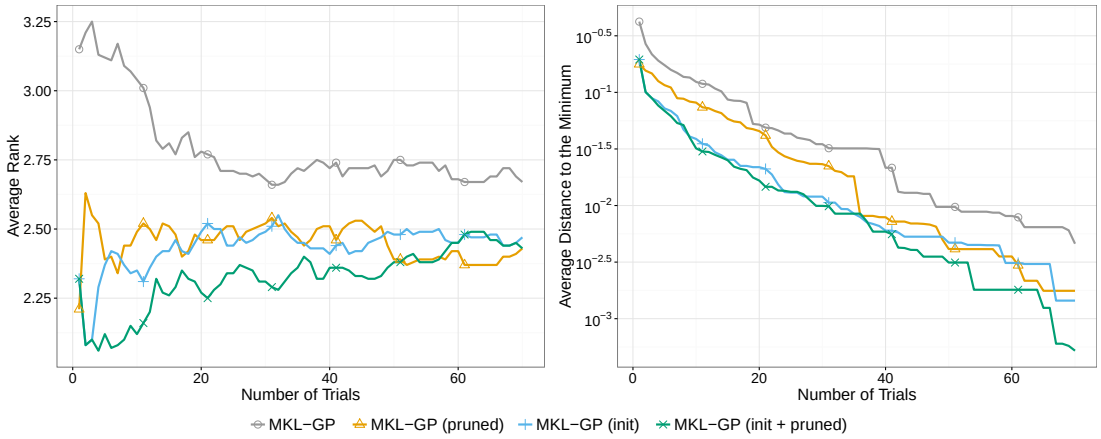

Figure 5.5: Average rank and average distance to the minimum for MKL-GP on the SVM meta-data set.

Table 5.2: Average rank and average distance to the minimum after 30 trials on the SVM meta-data set. Best results are bold.

I-GP	no pruning/init	pruned	init	init + pruned
Average Rank@30	3.12	2.35	2.72	**1.81**
ADTM@30	0.0224	0.0131	0.0291	**0.0055**
I-RF	no pruning/init	pruned	init	init + pruned
Average Rank@30	3.51	2.11	2.51	**1.87**
ADTM@30	0.0281	0.0149	0.0116	**0.0070**
SCoT	no pruning/init	pruned	init	init + pruned
Average Rank@30	2.55	2.55	2.47	**2.43**
ADTM@30	0.0244	0.0244	**0.0237**	**0.0237**
MKL-GP	no pruning/init	pruned	init	init + pruned
Average Rank@30	2.68	2.52	2.49	**2.31**
ADTM@30	0.0349	0.0232	0.0120	**0.0099**

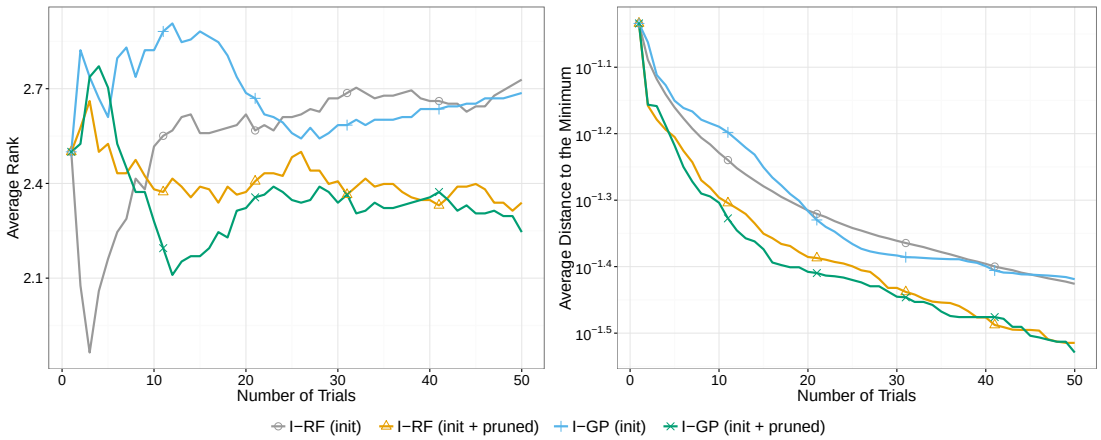

Figure 5.6: Average rank and average distance to the minimum for I-RF and I-GP on the Weka meta-data set.

5.3.2 Hyperparameter Optimization for Weka

In the last chapter, we have seen little improvement in cases where an initialization is combined with surrogate models that are learning across data sets. We expect pruning to be useful in two scenarios: if i) the dimensionality of the hyperparameter space is very high and ii) the meta-data set is too large such that surrogate models that are learning across data sets are no longer a cost-efficient alternative to evaluating the true function. Since most surrogate models are based on Gaussian processes, a further problem is storing the kernel matrix. In our next meta-data set we are using more than a million meta-instances which result into a kernel matrix of dimensions $10^6 \times 10^6$ which needs 8 TB of memory for storing it.

For the Weka meta-data set we conducted a similar experiment as for the SVM meta-data set. Due to the size of the data set, we restricted ourselves to the optimization strategies that do not learn across data sets. Previously, we have seen that an optimization strategy without both, initialization and pruning, is outperformed by a large margin by the same strategy only using pruning. Hence, we show here only the comparison between the strategy i) only using an initialization step and ii) using both initialization and pruning. Figure 5.6 concludes our experiments. As we have seen on the SVM meta-data set, the results indicate that pruning is a useful addition to Bayesian optimization by further accelerating the hyperparameter optimization.

5.4 Conclusion

We propose pruning as an orthogonal contribution to Bayesian optimization and show in elaborated experiments on two different meta-data sets that it accelerates the hyperparameter optimization in

most cases and in the worst case does not worsen it. It can be especially considered for optimization strategies that do not use information from the past for the surrogate model.

6

Hyperparameter Optimization Initialization

A typical way of accelerating a search is by means of a good initialization. We propose to transfer knowledge by using an initialization strategy for hyperparameter optimization. In contrast to the current state of the art initialization strategies, our strategy is neither limited to hyperparameter configurations that have been evaluated on previous experiments nor does it need meta-features. The initial hyperparameter configurations are derived by optimizing for a meta-loss which we formally define in this chapter. This loss depends on the hyperparameter response function of the data sets that were investigated in previous experiments. Since this function is unknown and only few observations are given, the meta-loss is not differentiable. We propose to approximate the response function by a differentiable plug-in estimator. Then, we are able to learn the initial hyperparameter configuration sequence by applying gradient-based optimization techniques.

Extensive experiments are conducted on two meta-data sets. Our initialization strategy is compared to the state of the art for initialization strategies and further methods that are able to transfer knowledge between data sets. We give empirical evidence that our work provides an improvement over the state of the art.

6.1 Introduction

As discussed in the previous chapters, hyperparameter optimization is an omnipresent and vital problem in machine learning. Among the existing strategies to find near-optimal hyperparameter configurations, Bayesian optimization [37] is the current state of the art for hyperparameter optimization. Recent work tries to transfer knowledge about the hyperparameter space from past experiments to a new data set. This is either done by using surrogate models that consider past experiments [4,72,87,61] or by using the information on past experiments to initialize the new search [24,23,26,58]. The motivation behind this approach is that subsets of the hyperparameter space that are good on few data sets are likely good candidates for others. While the first approach is limited to the application in Bayesian optimization, the second can be used for any hyperparameter optimization strategies.

In this chapter, we mathematically formalize the problem of hyperparameter optimization and derive a meta-loss for hyperparameter optimization. Since this meta-loss is yet not differentiable, we propose to use a differentiable plug-in estimator. Given this estimator, initial hyperparameter configurations can be learned by gradient-based optimization techniques that minimize the meta-loss. In contrast to existing initialization strategies, our strategy is neither limited to hyperparameter configurations that have been evaluated on previous data sets nor does it depend on meta-features. Additionally, we can show that a direct optimization for the right loss leads to better initializations and ultimately to an accelerated hyperparameter optimization.

6.2 Learning Initializations

In this section, the task of initializing hyperparameter optimization strategies is generalized and a novel approach to choose initial hyperparameter configurations is proposed. Instead of choosing from hyperparameter configurations that have been best on previous data sets [24,26], we directly learn hyperparameter configurations and thus are not limited to hyperparameter configurations that have been evaluated on previous data sets. The idea is to initialize the initial sequence of hyperparameter configurations with promising configurations and further improve them by minimizing the proposed hyperparameter loss. Since this loss is differentiable, the initial sequence of hyperparameter configurations can be learned with gradient-based optimization techniques such as stochastic gradient descent. This loss is derived by formally expressing a useful evaluation measure for hyperparameter optimization in general.

6.2.1 A Loss for Initializing the Hyperparameter Optimization

In Section 3.2.1, the evaluation measure for hyperparameter optimization in general is formalized. A good initialization should support a faster convergence of the hyperparameter search such that the evaluation measure is minimal after T steps. Although, this cannot necessarily be achieved by minimizing the same measure within the I initialization iterations, we use this loss as a proxy and optimize the initial sequence of hyperparameter configurations for the same evaluation measure. Thus, we optimize directly for ADTM (Equation (3.4)) by minimizing the meta-loss in Equation (6.1).

$$\mathcal{L}\left(\mathcal{X}_I, \mathcal{D}\right) = \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \min_{\mathbf{x} \in \mathcal{X}_I} f_D\left(\mathbf{x}\right) \tag{6.1}$$

6.2.2 Differentiable Meta-Loss

The goal in this work is to learn a set of I initial hyperparameter configurations that are not limited to the candidates evaluated on previous data sets. This sequence is learned by minimizing the meta-loss defined in Equation (6.1).

Obviously, this loss is not differentiable. The minimum function is a non-differentiable function and the function f_D is only partially observed. The minimum function can be approximated by the differentiable softmin function $\sigma : \mathbb{R}^m \to \mathbb{R}^m$ as defined in Equation (6.2). We choose $\beta = -100$ as proposed by Grabocka et al.[27] such that $\sum_{i=1}^{m} \mathbf{x}_i \sigma\left(\mathbf{x}\right)_i$ is very close to the true minimum $\min\left\{\mathbf{x}_1, \ldots, \mathbf{x}_m\right\}$.

$$\sigma\left(\left(\mathbf{x}_1, \ldots, \mathbf{x}_m\right)^T\right)_i = \frac{e^{\beta \mathbf{x}_i}}{\sum_{j=1}^{m} e^{\beta \mathbf{x}_j}} \tag{6.2}$$

The function f_D is approximated by a differentiable plug-in estimator or surrogate model $\hat{f}_D$. This can be any differentiable regression model that is able to learn from few observations on D and generalize to unseen meta-instances. Thus, the final, differentiable meta-loss is given in Equation (6.3).

$$\mathcal{L}\left(\mathcal{X}_I, \mathcal{D}\right) = \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \sum_{i=1}^{I} \sigma_{D,i} \hat{f}_D\left(\mathbf{x}_i\right) \tag{6.3}$$

where for notational convenience, the following notation is used in the remainder of the chapter,

$$\sigma_{D,i} \quad = \quad \frac{e^{\beta \hat{f}_D(\mathbf{x}_i)}}{\sum_{j=1}^{I} e^{\beta \hat{f}_D(\mathbf{x}_j)}} \quad . \tag{6.4}$$

6.2.3 Gradients for the Meta-Loss

In order to apply a gradient-based learning algorithm that minimizes the meta-loss, the gradients for this loss with respect to the initial hyperparameter configurations need to be estimated.

$$\begin{aligned} \frac{\partial}{\partial \mathbf{x}_{l,j}} \mathcal{L}\left(\mathcal{X}_I, \mathcal{D}\right) &= \frac{\partial}{\partial \mathbf{x}_{l,j}} \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \sum_{i=1}^{I} \sigma_{D,i} \hat{f}_D\left(\mathbf{x}_i\right) \\ &= \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \sigma_{D,l} \left(\frac{\partial}{\partial \mathbf{x}_{l,j}} \hat{f}_D\left(\mathbf{x}_l\right) \right) \cdot \left(\beta \left(1 - \sigma_{D,l}\right) \hat{f}_D\left(\mathbf{x}_l\right) + 1 \right) \end{aligned} \tag{6.5}$$

Hyperparameter response functions are highly non-linear functions. Hence, our plug-in estimator needs to be able to model this property. Gaussian processes have proven to be a good surrogate model for the meta-testing data, hence we decided to also use it as a surrogate model for each training data set D. Hence, using the notation from Section 2.6, the derivative for this specific prediction model is

$$\frac{\partial}{\partial \mathbf{x}_{l,j}} \mathcal{L}\left(\mathcal{X}_I, \mathcal{D}\right) = \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \sigma_{D,l} \left(\boldsymbol{\alpha}_D^T \frac{\partial}{\partial \mathbf{x}_{l,j}} \mathbf{k}_{D,*} \right) \cdot \left(\beta \left(1 - \sigma_{D,l}\right) \mathbf{k}_{D,*}^T \boldsymbol{\alpha}_D + 1 \right) \quad . \tag{6.6}$$

6.2.4 Learning Algorithm

After deriving the gradients, the final learning algorithm is presented in Algorithm 7. First, a surrogate model for each training data set is computed. Then, some initial values for $\mathbf{x}_1, \ldots, \mathbf{x}_I$ are chosen. Instead of using a more sophisticated initialization step such as k-means, a relatively simple strategy is used. A subset containing I of all $|\mathcal{D}|$ training data sets are chosen at random and the best hyperparameter configurations for these data sets are used as initial values. Afterwards, the initial solution is iteratively improved by updating the hyperparameter configurations into the negative direction of the gradient weighted by the learning rate η. For our experiments we identified $\eta = 10^{-3}$ and $E = 10^3$ epochs as useful values for our problem.

6.2.5 Adaptive Data Set Weights

So far, the influence of each training data set for finding the optimal initial hyperparameter configurations is equal. Naturally, it is expected that one data set may contain more information than others, hence, we propose to weight the influence of each data set with a similarity function that depends on one training data set D and the new data set D_{new} for which the optimal hyperparameter

Algorithm 7 Learning Hyperparameter Optimization Initialization

Input: Meta-training set $\mathcal{D}$, number of initial hyperparameter configurations I, number of epochs E, meta-learn rate η.

Output: Optimal set of hyperparameter configurations for initialization.

```
for D ∈ 𝒟 do
    Compute f̂_D(x_*) = k_{D,*}^T α_D on observed meta-instances (x_i, f_D(x_i)).
end for
Initialize 𝒳_I = {x_1, ..., x_I} with the best hyperparameter configurations of I data sets D ∈ 𝒟 chosen at random.
for e = 1 to E do
    Precompute f̂_D(x_i) and σ_D for all i = 1...I and D ∈ 𝒟.
    for i = 1 to I do
        for j = 1 to P (in parallel) do
            x_{i,j} ← x_{i,j} − η ∂/∂x_{i,j} ℒ(𝒳_I, 𝒟)
        end for
    end for
end for
return 𝒳_I
```

configuration is sought.

$$\mathcal{L}\left(\mathcal{X}_I, \mathcal{D}\right) = \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \text{sim}\left(D, D_{\text{new}}\right) \sum_{i=1}^{I} \sigma_{D,i} \hat{f}_D\left(\mathbf{x}_i\right) \tag{6.7}$$

The similarity function sim should be 1 if D is similar to D_{new} and close to 0 if they are completely different. This similarity function may depend on meta-features[24,58] but we propose to iteratively reupdate and recompute the initial hyperparameter configurations depending on the outcome of already performed evaluations $f_{D_{\text{new}}}\left(\mathbf{x}_1\right), \ldots, f_{D_{\text{new}}}\left(\mathbf{x}_t\right)$, $t < I$. The assumption is that the I initial hyperparameter configurations are not evaluated in parallel but sequentially. Then, at time step $t+1$, it is sufficient to minimize the loss in Equation (6.7) by keeping the already evaluated hyperparameter configurations $\mathcal{X}_t = \{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$ fixed and find $\mathbf{x}_{t+1}$ that minimizes the loss. The weighting of the data sets is updated using the approximated Kendall Tau Rank Correlation Coefficient as defined in Equation (5.5).

6.2.6 Comparison to State of the Art

The state of the art strategy for initializing hyperparameter optimization[24,58] can be understood as a special case of our general loss function in Equation (6.7). They optimize for the same loss but use

Table 6.1: Brief overview of the competitor methods. An explanation of this table is given in Section 3.2.3.

Name	Abbrev.	Surrogate Model	Meta-Knowledge	Details
Random Search	Random	None	✗	Sec. 2.2
Independent Gaussian Process	I-GP	Gaussian Process	✗	Sec. 2.7.1
Independent Random Forest	I-RF	Random Forest	✗	Sec. 2.7.1
Surrogate Collaborative Tuning	SCoT	SVMRank + Gaussian Process	✓	Sec. 2.7.2
Gaussian Process with Multi-Kernel Learning	MKL-GP	Gaussian Process	✓	Sec. 2.7.2

a different similarity function. While we are using a constant similarity function in Section 6.2.4 or an adaptive function in Section 6.2.5, they propose to consider only the data sets that are nearest regarding a distance function δ over the meta-features of the data set. Formally, they use

$$\text{sim}\left(D, D_{\text{new}}\right) = \begin{cases} 1 & \text{if } D \text{ is among the } I \text{ nearest data sets regarding } \delta \text{ to } D_{\text{new}} \\ 0 & \text{otherwise} \end{cases}. \tag{6.8}$$

To optimize the loss function with respect to this similarity function, no difficult optimization technique is needed. Simply taking the best hyperparameter configurations of the I nearest data sets to D_{new} is sufficient.

This choice of initial hyperparameter configurations is limited to the hyperparameter configurations that have been investigated on the training data sets. We will show that better initial hyperparameter configurations can be found if this constraint is not given.

6.3 Experimental Evaluation

In this section we will analyze the impact of our proposed initialization strategy on state of the art hyperparameter optimization methods as summarized in Table 6.1. We compare our initialization strategy to competitor initialization methods and to transfer surrogate models. In our results we report the mean of at least ten repetitions of a leave-one-data-set-out cross-validation. Details about the meta-data sets and the evaluation metrics can be found in Chapter 3.

6.3.1 Initialization Strategies

Following initialization strategies will be considered in our experiments.

No Initialization (No Init) No initialization is used. This is equivalent to a random initialization for all surrogate models that do not transfer knowledge from previous experiments. Thus, these results were repeated 1,000 times instead of only 10 times and averaged.

Random Best Initialization (RBI) This initialization is a very simple initialization. I training data sets from the set of all training data sets $\mathcal{D}$ are chosen uniformly at random. Then, the best hyperparameter configurations on these data sets are used for the initialization. Hence, this corresponds to the initialization used for Algorithm 7.

Nearest Best Initialization (NBI) This is the initialization strategy proposed by Reif et al. and Feurer et al.[24,58]. Instead of choosing I training data sets uniformly at random, they are chosen with respect to the similarity between the meta-features listed in Table 3.4. Then, like for RBI, the best hyperparameter configurations on these data sets are chosen for initialization.

Learning Initializations (LI) Learning Initializations is the strategy introduced in Section 6.2 and summarized in Algorithm 7. Its advantage is that it is directly optimized for the loss and the selected hyperparameter configurations are not limited to hyperparameter configurations that were previously observed in past experiments. For our experiment, we precomputed the values of the response function on a grid. Since the initial hyperparameter configurations found by Learning Initialization will likely not be on this grid. However, to avoid the time-consuming evaluation for the exact hyperparameter configurations, the hyperparameter configurations were chosen that are closest to the one proposed by Learning Initializations.

Adaptive Learning Initializations (aLI) Adaptive Learning Initializations is presented in Section 6.2.5. It is an extension to Learning Initializations (LI) that tries to incorporate the knowledge about the new data set that is sequentially gathered by re-weighting the influence of each training data set.

6.3.2 Experiments

Two different experiments are conducted. First, state of the art initialization strategies are compared with respect to the ADTM after I initial hyperparameter configurations. Second, the long term effect on the hyperparameter optimization is compared. Even though the initial hyperparameter

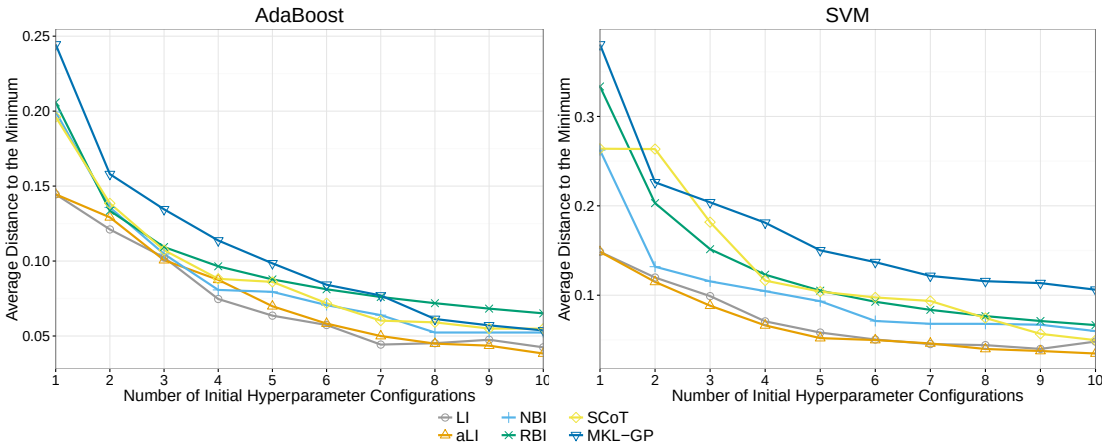

Figure 6.1: Development of the ADTM for increasing number of initial hyperparameter configurations on both meta-data sets. Our proposed strategies LI and aLI are outperforming the state of the art initialization strategies (RBI/NBI) and state of the art surrogate models that transfer knowledge from previous experiments (SCoT and MKL-GP).

configurations lead to good results after I steps, the ultimate aim is to find near-optimal hyperparameter configurations after T iterations.

Comparison to other Initialization Strategies

The ADTM on the two meta-data set for $I = 1 \ldots 10$ for different initialization strategies is shown in Figure 6.1. This experiment analyzes i) the performance of initialization strategies versus transfer surrogate models that are transferring knowledge from previous experiments (SCoT and MKL-GP), ii) the legitimacy of the use of meta-features as a measure for similarity between data sets and iii) compares our proposed initialization strategies to the current state of the art. One may argue, that in the case that a surrogate model is used to transfer knowledge from previous experiments, no further initialization strategy is needed. The experiments do not indicate that this is true. While we acknowledge that at least SCoT provides a moderate initialization sequence, it is still not able to beat the best initialization strategy. A direct comparison between RBI and NBI indicates that meta-features can be used to estimate the similarity between data sets. The use of meta-features in NBI leads to an improved initialization compared to RBI. Finally, our proposed initialization strategy LI provides a very good initialization for $I = 1$ and is consistently the best initialization. Its variation aLI does not seem to provide better results.

Comparison with Respect to the Long Term Effect

The aim of an initialization strategy is to accelerate the hyperparameter search and convergence to a close-to-optimal hyperparameter configuration. Thus, not the performance at the end of the initialization is essential but the further convergence. Therefore, a further experiment was conducted. Bayesian optimization was initialized with five hyperparameter configurations using the respective initialization strategy and then was continued using Bayesian optimization. All initialization strategies are also compared to the case where no initialization was used. The experiments are conducted for all surrogate models presented in Table 6.1.

We expect that transferring knowledge from previous experiments has more impact if this is not done by the surrogate model. This can be seen in Figure 6.2. Comparing Figure 6.1 with Figure 6.2 on the SVM meta-data set, one can see that all initialization strategies provide better results for the first initial value than the I-GP without initialization does after six iterations of Bayesian optimization. The results in Figure 6.2 and 6.3 are of special interest because the effect of the knowledge transfer is not distorted by a surrogate model that also transfers knowledge. Our proposed strategy LI outperforms any other initialization strategy on both meta-data sets and for both surrogate models. Again, aLI does not provide better results than LI. It is interesting to notice that RBI provides similar results to NBI.

The results in Figure 6.4 and 6.5 demonstrate the impact of an initialization strategy on a surrogate model that transfers already knowledge from previous experiments. Obviously, the gap between no initialization and some specific initialization shrinks. SCoT without initialization achieves similar results as SCoT with RBI or NBI but SCoT with LI consistently achieves the best results. Account should be taken of the fact that the result lines will compulsory get closer with growing number of Bayesian optimization iterations and will meet after just enough iterations. For the MKL-GP the results are similar, but here NBI is able to outperform LI on the SVM meta-data set.

So far we argued carefully that, even though surrogate models that are transferring knowledge across data sets exist, there is still need for an initialization strategy. Moreover, one can question: are these surrogate models necessary if a good initialization strategy is used? Surrogate models that learn across data sets have three big disadvantages:

1. The run-time for updating the surrogate model after each Bayesian optimization iteration is by a factor of $\mathcal{O}\left(|\mathcal{D}|^2\right)$ higher than a surrogate model that is only trained on the current data set (assuming the surrogate model is based on a Gaussian process which is the case for SCoT and MKL-GP).

2. A specific surrogate model is needed and no out of the box machine learning model such

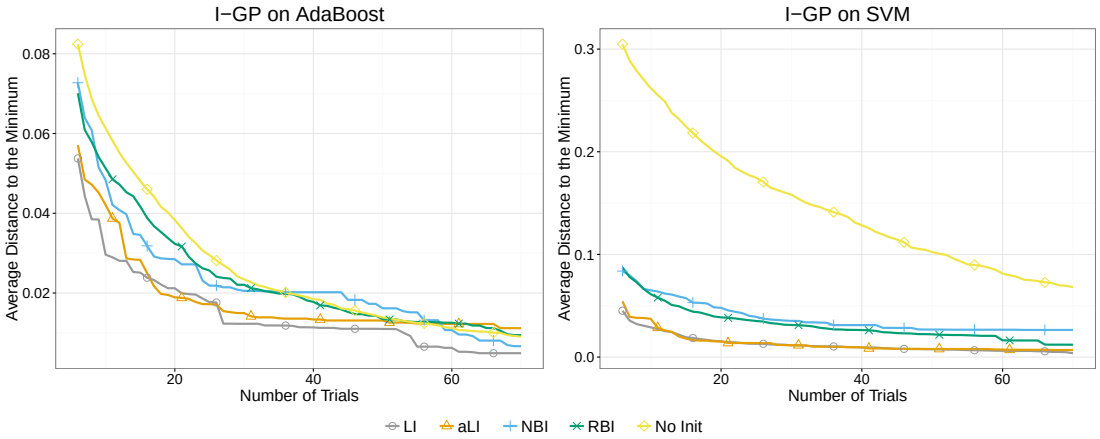

Figure 6.2: Impact of an initialization with five hyperparameter configurations on the long term optimization for I-GP (a surrogate model that does not use information from previous experiments on other data sets). Our proposed strategies LI and aLI are outperforming alternative initialization strategies on both meta-data sets.

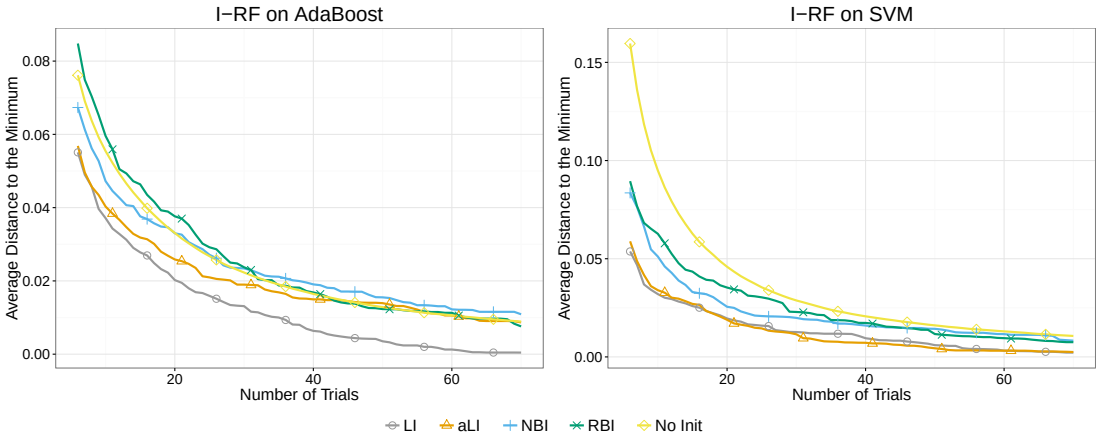

Figure 6.3: Impact of an initialization with five hyperparameter configurations on the long term optimization for I-RF (a surrogate model that does not use information from previous experiments on other data sets). Our proposed initialization strategy LI is outperforming the state of the art on both meta-data sets.

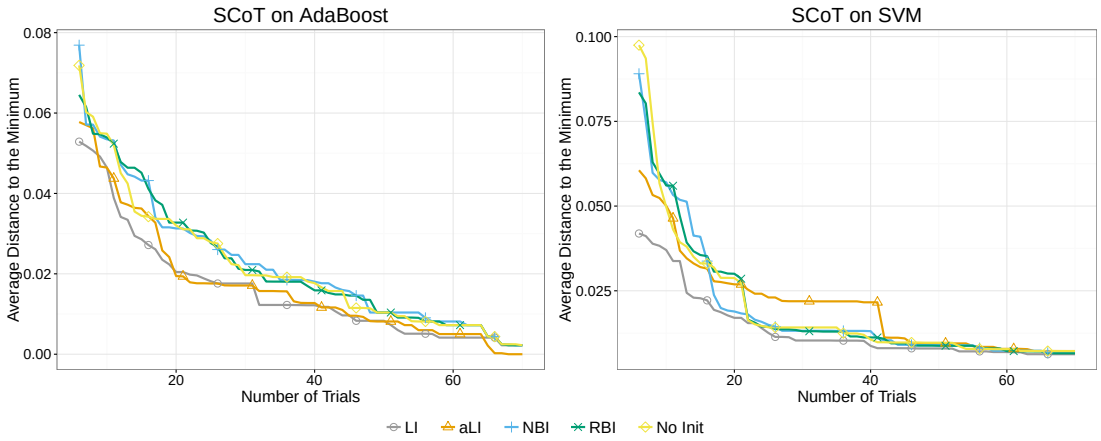

Figure 6.4: Impact of an initialization with five hyperparameter configurations on the long term optimization for SCoT (a surrogate model that transfers knowledge from previous experiments to the new data set). Our proposed initialization strategy LI is outperforming the state of the art on both meta-data sets.

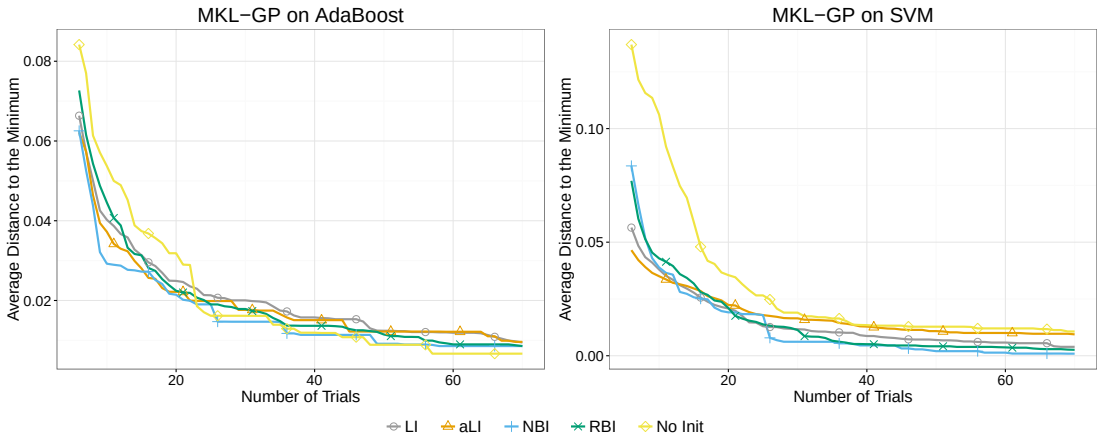

Figure 6.5: Impact of an initialization with five hyperparameter configurations on the long term optimization for MKL-GP (a surrogate model that transfers knowledge from previous experiments to the new data set). Our proposed initialization strategy LI is outperforming the state of the art on AdaBoost meta-data sets. We acknowledge that NBI provides better results for the SVM meta-data set.

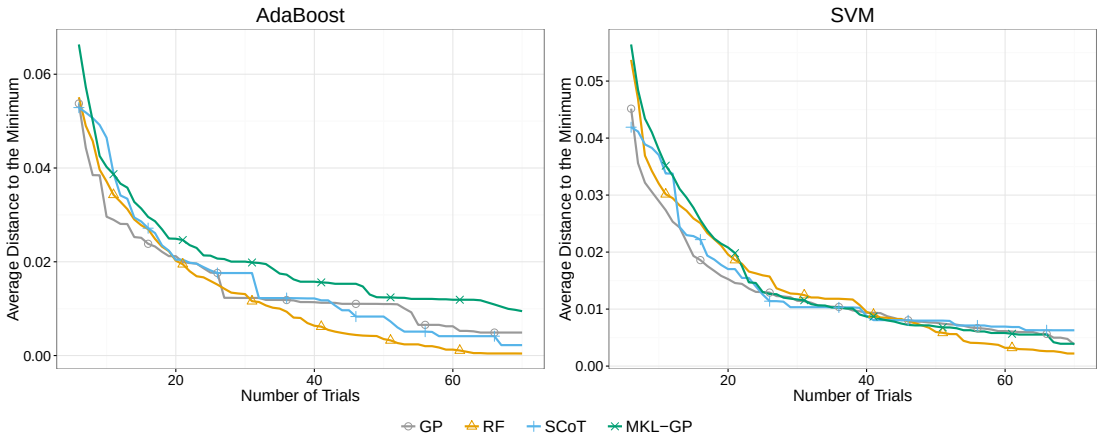

Figure 6.6: Comparison of the performance development of four different surrogate models that are all initialized with five LI hyperparameter configurations. RF is a surrogate model that does not transfer knowledge between data sets but yet performs best only due to the LI initialization strategy. Hence, current state of the art surrogate models that transfer knowledge between data sets do not seem to achieve better results if one uses a initialization strategy instead.

as a Gaussian process or random forest can be used. Additionally, hand-crafted, problem-dependent meta-features need to be used.

3. These surrogate models are specific for Bayesian optimization and cannot be used for other optimization frameworks such as the initialization strategy. Furthermore, an initialization strategy can be estimated by a single researcher and then shared with other researchers and practitioners while the specific surrogate model always also includes sharing the meta-data.

These disadvantages may be tolerable if there is an improvement in terms of hyperparameter optimization acceleration. Figure 6.6 compares both surrogate models that do not use meta-features and do not transfer knowledge (I-GP and I-RF) to those that do (SCoT and MKL-GP). All four strategies are initialized with five hyperparameter configurations by LI since this provided better results for all strategies. As the reader can see, the surrogate models that learn across data sets do not provide better results.

6.3.3 Experimental Conclusion

Concluding this section, our proposed initialization strategy LI is able to provide better initial hyperparameter configurations than the state of the art. Furthermore, it demonstrates better results with respect to the long term effect. Initialization strategies for optimization strategies that do not transfer knowledge from other experiments gain most from the initialization. But also for surrogate models that already transfer the knowledge an improvement is recognizable. Furthermore, the cur-

rent state of the art surrogate models do not seem to be necessary if a good initialization strategy is chosen. The proposed adaptive re-weighting of the data set weights (aLI) has some potential but the re-weighting as proposed by us does not show better results than a constant weighting.

6.4 Conclusion

A meta-loss for hyperparameter optimization was derived that depends on the hyperparameter response function of previously seen data sets. Since the response function is unknown and only few observations are given, the meta-loss is not differentiable. By approximating the response function with a differentiable plug-in estimator, the meta-loss becomes differentiable. That in turn enabled us to learn initial hyperparameter configurations that minimize the meta-loss. These learned initial hyperparameter configurations are not limited to configurations that have been seen before.

Our initialization learning algorithm was compared to state of the art initialization strategies and provided better initial values and furthermore has better long term effects on the hyperparameter optimization. Finally, a generalized meta-loss was presented that allows to (dynamically) weight the influence of each data set and we have shown that the current state of the art initialization strategies are optimized for a special case of this general meta-loss.

7

Two-Stage Transfer Surrogate Model

In Section 2.7, we presented different surrogate models based on Gaussian processes. They have the tempting property that they are hyperparameter-free and naturally provide certainties about their predictions. However, with growing meta-information Gaussian processes are no longer feasible as they involve the inversion of a kernel matrix of the size of the training data, which naturally limits their applicability.

Consequently, instead of learning a joint surrogate model on all of the meta-data, we propose a scalable two-stage transfer surrogate model framework. In this framework, we propose to first learn individual surrogate models on the observations of each data set and then combine all surrogates to a joint one using ensembling techniques. The final surrogate is a weighted sum of all data set specific surrogate models plus an additional surrogate that is solely learned on observations of the new data set. We present two different strategies on finding ensembling weights. The first one is based on a probabilistic product of experts approach, the second one is based on kernel regression.

In an empirical evaluation including comparisons to the current state of the art on our meta-data sets, we are able to demonstrate that our proposed approach does not only scale to large meta-data but also finds the stronger prediction models.

7.1 Scalable Two-Stage Transfer Surrogate Framework

In Section 2.6 we have discussed how to learn Gaussian processes, where the computationally most expensive step lies in the inversion of the kernel matrix K, which is of size n if we are facing n many

training instances. Given the scenario that we are in possession of large scale meta-knowledge, learning a Gaussian process becomes infeasible as inverting K can only be done in $\mathcal{O}(n^3)$. However, Gaussian processes are a decent choice as surrogate models for Bayesian optimization, as they naturally predict uncertainties and are basically hyperparameter free.

Besides the computational challenges, learning a Gaussian process on all training instances makes the strong assumption that each training instance and data set are equally important. This issue is usually addressed by adding meta-features which leads to a nontransparent and implicit representation of similarity between data sets and their influence. We want to propose a framework that tackles both issues by making Gaussian processes scalable and making the influence of each data set within the meta-data explicit.

Therefore, in order to still learn Gaussian processes, we propose to subdivide the meta-data into M many individual parts and learn a single Gaussian process independently on each of the parts plus a single Gaussian process for all the new observations that we will see during the Bayesian optimization trials. Formally, we divide our meta-data

$$\mathbf{X} = (\mathbf{X}^{(1)}, ..., \mathbf{X}^{(M)}) \qquad \mathbf{f} = (\mathbf{f}^{(1)}, ..., \mathbf{f}^{(M)}) \ , \tag{7.1}$$

in a way where all $\mathbf{X}^{(i)}$ are pairwise disjoint. However, instead of taking an arbitrary subdivision of our meta-data, we simply divide it by the data sets we have already observed. This means, for each data set D_i, we create a subset $\mathbf{X}^{(i)}$, $\mathbf{f}^{(i)}$ which contains all meta-instances of data set D_i. As a result, we have M Gaussian processes learned, one for each data set, such that for every $i = 1, \ldots, M$

$$p\left(f_* \mid \mathbf{x}_*, \mathbf{X}^{(i)}, \mathbf{f}^{(i)}, \boldsymbol{\theta}^{(i)}\right) = \mathcal{N}\left(f_* \mid m_i(\mathbf{x}_*), \sigma_i^2(\mathbf{x}_*)\right) \ . \tag{7.2}$$

As mentioned earlier, we also learn a Gaussian process for the new observations, which will be updated after every Bayesian optimization trial. We will simply use the index $M+1$ for this Gaussian process.

We derive our scalable two-stage transfer surrogate framework (TST) by combining all $M+1$ Gaussian processes into a weighted, normalized sum as sketched in Figure 7.1. We define the following mean and precision

$$m\left(\mathbf{x}_*\right) = \frac{\sum_{i=1}^{M+1} w_i\left(\mathbf{x}_*\right) m_i\left(\mathbf{x}_*\right)}{\sum_{i=1}^{M+1} w_i\left(\mathbf{x}_*\right)} \tag{7.3}$$

$$\sigma^{-2}(\mathbf{x}_*) = \sum_{i=1}^{M+1} v_i \sigma_i^{-2}\left(\mathbf{x}_*\right) \ . \tag{7.4}$$

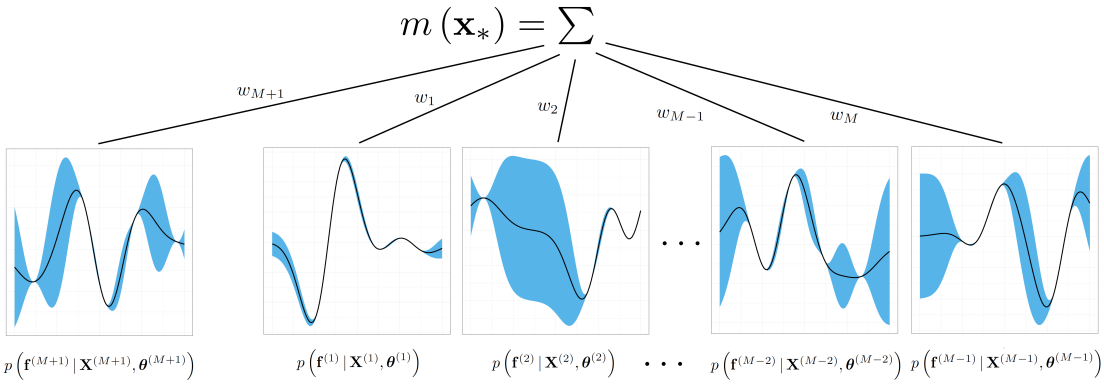

Figure 7.1: The proposed framework for our scalable transfer surrogate based on Gaussian processes. A Gaussian process is learned per data set and they are finally combined in a weighted sum.

The final framework is summarized in Algorithm 8. It consists of two different parts, at first the training of the individual processes, secondly how to combine them for prediction. As mentioned before, training involves dividing the meta-instances in M subsets, one subset for each data set on which we observed evaluations. Thus, every Gaussian process becomes the expert of the respective data set. The prediction uses these experts plus one additional expert that is estimated on the observed performances on the new data set. Based on Equations (7.3) and (7.4), the mean and uncertainty is estimated. In the following subsections, we will discuss how to derive possible options for choosing w and v which we introduced in Equations (7.3) and (7.4). Each version is a possible surrogate model that can be used in Bayesian optimization.

7.1.1 Product of Experts

The work by Schilling et al.[62] is a special case of our framework (Algorithm 8). In contrast to us, they are focusing more on the scalability aspect and less on the meta-learning aspect while we achieve both aspects with our approach. They propose the use of product of experts[33] and formally derive values for the parameters $\mathbf{w}$ and $\mathbf{v}$ as defined in our scalable two-stage transfer surrogate framework. Following the work of Hinton et al.[33], when applying the independence assumption, we can write the joint likelihood in Equation (2.56) as a product of individual likelihoods

$$p\left(f_* \mid \mathbf{x}_*, \mathbf{X}, \mathbf{f}\right) = \prod_{i=1}^{M+1} p\left(f_* \mid \mathbf{x}_*, \mathbf{X}^{(i)}, \mathbf{f}^{(i)}, \boldsymbol{\theta}^{(i)}\right) \quad , \tag{7.5}$$

which is also called a product of experts model and has been introduced by Hinton[33]. Additionally, weighting coefficients β_i have been proposed to use in the product of experts model to derive the

Algorithm 8 Scalable Gaussian Process Transfer Surrogate Framework

function TRAIN($\mathbf{X}$, $\mathbf{f}$)
  Split meta-data by data set:
    $\mathbf{X} = (\mathbf{X}^{(1)}, ..., \mathbf{X}^{(M)})$ $\mathbf{f} = (\mathbf{f}^{(1)}, ..., \mathbf{f}^{(M)})$ Equation (7.1)
  for $i = 1$ to M **do**
    Estimate $p\left(\mathbf{f}^{(i)} \mid \mathbf{X}^{(i)}, \boldsymbol{\theta}^{(i)}\right) = \mathcal{N}\left(\mathbf{f}^{(i)} \mid m_i(\mathbf{X}^{(i)}), \Sigma_i(\mathbf{X}^{(i)}, \mathbf{X}^{(i)})\right)$. Equation (2.56)
  end for
end function

function PREDICT($\mathbf{x}$, $p\left(\mathbf{f}^{(M+1)} \mid \mathbf{X}^{(M+1)}, \boldsymbol{\theta}^{(M+1)}\right)$)
  $m = \frac{\sum_{i=1}^{M+1} w_i(\mathbf{x}_*) m_i(\mathbf{x}_m)}{\sum_{i=1}^{M+1} w_i(\mathbf{x}_*)}$ Equation (7.3)
  $\sigma = \sqrt{\left(\sum_{i=1}^{M+1} v_i \sigma_i^{-2}(\mathbf{x}_*)\right)^{-1}}$ Equation (7.4)
  return (m, σ)
end function

generalized product of experts

$$p\left(f_* \mid \mathbf{x}_*, \mathbf{X}, \mathbf{f}\right) = \prod_{i=1}^{M+1} p^{\beta_i}\left(f_* \mid \mathbf{x}_*, \mathbf{X}^{(i)}, \mathbf{f}^{(i)}, \boldsymbol{\theta}^{(i)}\right) \quad , \tag{7.6}$$

where the initial formulation is obtained by setting all $\beta_i = 1$[33]. Usually, the coefficients β_i in the generalized product of experts are chosen to sum up to one.

Computing the product of all these Gaussian densities, we obtain a Gaussian distribution with the following mean and precision:

$$m\left(\mathbf{x}_*\right) = \sigma^2\left(\mathbf{x}_*\right) \sum_{i=1}^{M+1} \beta_i \sigma_i^{-2}(\mathbf{x}_*) m_i(\mathbf{x}_*) \tag{7.7}$$

$$\sigma^{-2}\left(\mathbf{x}_*\right) = \sum_{i=1}^{M+1} \beta_i \sigma_i^{-2}(\mathbf{x}_*) \quad . \tag{7.8}$$

Substituting the precision into the formula for the mean, the predicted mean resolves to

$$m\left(\mathbf{x}_*\right) = \frac{\sum_{i=1}^{M+1} \beta_i \sigma_i^{-2}(\mathbf{x}_*) m_i(\mathbf{x}_*)}{\sum_{i=1}^{M+1} \beta_i \sigma_i^{-2}(\mathbf{x}_*)} \quad , \tag{7.9}$$

which is a sum of means, weighted by the product of β_i and the individual precisions. For our experiments, we set

$$\beta_i = \frac{1}{M+1} \quad \forall i = 1, \ldots, M+1 \ , \tag{7.10}$$

which does not influence the predicted mean as the terms cancel out. However, this effectively increases the uncertainty which the general model of experts usually tends to underestimate [19]. To sum up, generalized products of experts are an instance of scalable transfer surrogates when setting

$$w_i(\mathbf{x}_*) = \beta_i \sigma_i^{-2}(\mathbf{x}_*) \tag{7.11}$$

$$v_i = \beta_i \tag{7.12}$$

as weight parameters in Algorithm 8.

7.1.2 Kernel Regression

In the previous section, we derived parameters w and v for Algorithm 8 under the assumption every data set has equal importance for the task of finding optimal hyperparameter configurations for our new data set D_{M+1}. But in order to find good hyperparameter configurations on a new data set D_{M+1}, it is very intuitive to rely stronger on the meta-knowledge of data sets that have a similar hyperparameter response surface. Hence, setting w_i to larger values for these experts to increase their influence makes a lot of sense. Assuming that we know the similarity $k(\chi_i, \chi_j)$ between two data sets D_i and D_j, where χ_i and χ_j are the data set descriptors of D_i and D_j, respectively, we proposed to set the value of w_i to the similarity between the data set D_i and the new data set D_{M+1} [84]:

$$w_i = k(\chi_i, \chi_{M+1}) \ . \tag{7.13}$$

The concrete kernel that we apply is the Epanechnikov quadratic kernel

$$k_\rho(\chi_i, \chi_j) = \delta\left(\frac{\left\|\chi_i - \chi_j\right\|_2}{\rho}\right) \ , \tag{7.14}$$

where the δ function is given by

$$\delta(t) = \begin{cases} \frac{3}{4}\left(1 - t^2\right) & \text{if } t \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{7.15}$$

and $\rho > 0$ is the bandwidth. Setting $\mathbf{w}$ like this, our scalable Gaussian process transfer surrogate framework is now equivalent to kernel regression with the Nadaraya Watson kernel-weighted average for the mean prediction. Furthermore, we propose to rely on the uncertainty of the surrogate model for the new data set only:

$$v_i = \begin{cases} 1 & i = M+1 \\ 0 & \text{otherwise} \end{cases}. \tag{7.16}$$

We would like to use the true similarity between the new data set and all other data sets but since this is not available, we will evaluate two different common techniques to approximate it. One is based on meta-features, simple, statistical or information theoretic properties that are extracted from the data set which are considered to describe a data set[4,58,67]. We use the meta-features listed in Table 3.4 and explained in Section 3.1.4.

However, using these meta-features has one drawback. They are constant which means that the knowledge of the target data set enters the model only via the target Gaussian process which is updated after every trial. Therefore, we propose an alternative using a pairwise hyperparameter performance comparison[47,82]. The idea is to select pairs $\left(\mathbf{x}_i, \mathbf{x}_j\right)$ of evaluated hyperparameter configurations on the new data set D_{M+1} and count how often D_{M+1} and another data set D_k agree on the ranking of these configurations. After evaluating t many hyperparameter configurations during Bayesian optimization on the new data set, we estimate the data set descriptors for each data set D_k as

$$\left(\chi_k\right)_{j+(i-1)t} = \begin{cases} \frac{1}{t(t-1)} & \text{if } m_k\left(\mathbf{x}_i\right) > m_k\left(\mathbf{x}_j\right) \\ 0 & \text{otherwise} \end{cases}. \tag{7.17}$$

While the value of $\mathbf{f}\left(\cdot, D_{M+1}\right)$ is known for these t hyperparameter configurations, this is not necessarily true for the data sets $D_1, \ldots, D_M$. Hence, we use the prediction of each individual expert instead.

Computing the Euclidean distance of two meta-feature vectors then yields the number of discordant pairs normalized by dividing by the number of all pairs. This is basically a distance function based on the Kendall rank correlation coefficient[41]. In this way, during the Bayesian optimization process the coefficients are adapted after each iteration, where the data sets that agree on more hyperparameter pairs with the target data set are weighted higher. This has been shown to improve the performance drastically.

Table 7.1: Brief overview of the competitor methods. An explanation of this table is given in Section 3.2.3.

Name	Abbrev.	Surrogate Model	Meta-Knowledge	Details
Random Search	Random	None	✗	Sec. 2.2
Independent Gaussian Process	I-GP	Gaussian Process	✗	Sec. 2.7.1
Independent Random Forest	I-RF	Random Forest	✗	Sec. 2.7.1
Independent Gaussian Process with Meta-Initialization	I-GP (init)	Gaussian Process	✓	Chap. 6
Independent Random Forest with Meta-Initialization	I-RF (init)	Random Forest	✓	Chap. 6
Surrogate Collaborative Tuning	SCoT	SVMRank + Gaussian Process	✓	Sec. 2.7.2
Gaussian Process with Multi-Kernel Learning	MKL-GP	Gaussian Process	✓	Sec. 2.7.2
Factorized Multilayer Perceptron	FMLP	Neural Network	✓	Sec. 2.7.2

7.2 Experimental Evaluation

7.2.1 Optimization Strategies

We summarize all optimization strategies considered in the experiments. Our competitor methods are summarized in Table 7.1. We compare them to three different variations of the scalable two-stage transfer surrogate framework. TST-M is using the meta-feature representation for the data sets, TST-R is using the pairwise ranking representation. The third version provides another baseline[62] and uses product of experts which we call TST-PoE. We introduced it as an instance of TST in Section 7.1.1.

The results reported are estimated using a leave-one-data-set-out cross-validation and are the average of ten repetitions. For strategies with random initialization (Random, I-GP, I-RF), we report the average over thousand repetitions due to the higher variance. Details about the meta-data sets

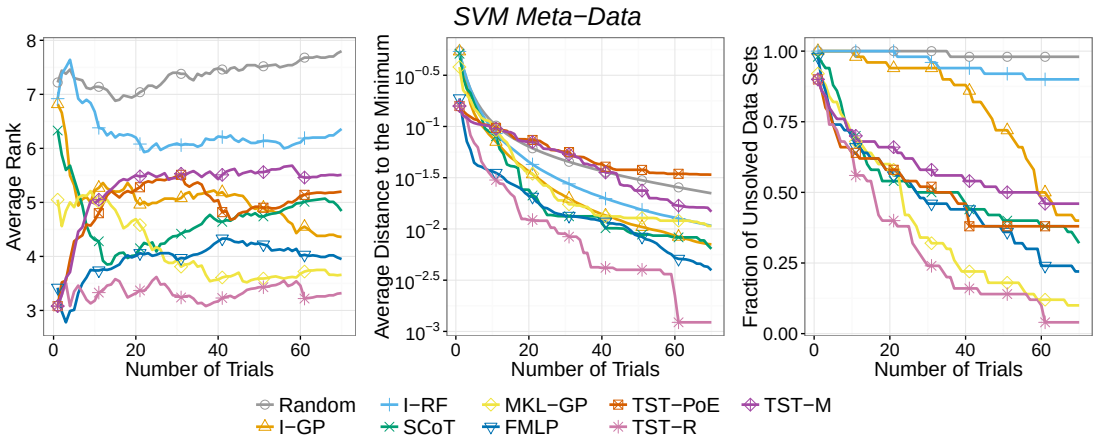

Figure 7.2: Our proposed transfer surrogate model TST-R provides the best performance with respect to both evaluation measures for the task of hyperparameter optimization. For both metrics, the smaller the better.

and the evaluation metrics can be found in Chapter 3.

7.2.2 Experiments

We compare the different hyperparameter optimization methods in two different scenarios: i) hyperparameter optimization and ii) combined algorithm selection and hyperparameter optimization. For the task of hyperparameter optimization, we optimize the hyperparameters of a support vector machine. The results are summarized in Figure 7.2. What we can see is that TST-R is outperforming the competitor methods with respect to both evaluation metrics by a large margin. TST-M has a similar good start as TST-R but its performance degenerates after few trials. Because the only difference between TST-R and TST-M is the way the data sets are described, one might argue that meta-features are less descriptive in describing a data set than the approach of pairwise rankings. We do not think that one can infer this from these results. The true reason for this behavior is that the distances for TST-R are updated after each trials and the distance to the data sets from previous experiments is increasing over time. Thus, the influence of the meta-data set vanishes and TST-R is focusing only on the knowledge about the new data set at some point of time. Contrariwise, TST-M is using a constant distance between data set based on the meta-features. While the meta-knowledge is useful especially in the beginning, TST-M keeps relying on this such that the information of the new data set is not optimally taken into account. One simple way of fixing this problem is to decay the influence of the meta-knowledge which would introduce at least one meta-hyperparameter. Because TST-R is performing well without an additional meta-hyperparameter for the decay, we do not follow this idea here.

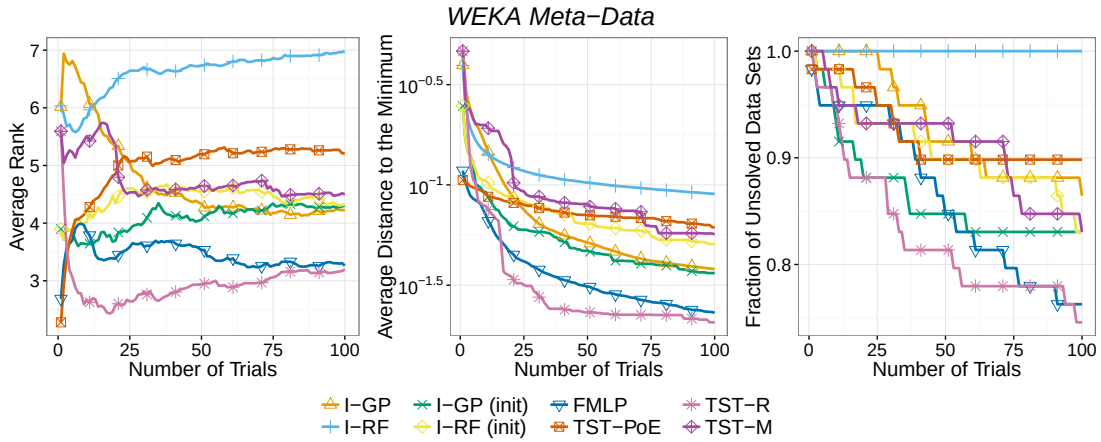

Figure 7.3: Our approach TST-R also outperforms the competitor methods for the task of combined algorithm selection and hyperparameter optimization. Surrogate models that use Gaussian processes that train over the whole meta-data are not feasible for this data set [79]. Therefore, we consider I-GP and I-RF with meta-learning initialization.

We investigate the performance of the optimization methods also for the problem of combined algorithm selection and hyperparameter optimization on our Weka meta-data set. For this experiment, we remove some methods for different reasons. We remove some weaker methods to improve the readability. Furthermore, we do not compare to methods, which are using one Gaussian process, that is trained on the complete meta-data (SCoT and MKL-GP). The reason for this is that Gaussian processes do not scale to these large meta-data sets (time and memory-wise) [79]. Our approach is learning one Gaussian process for each data set such that each model only needs to be learned on a fraction of the data and thus remains feasible. Nevertheless, we compare to FMLP, the strongest competitor from the previous experiment as well as I-GP, I-RF and TST-PoE. Furthermore, we also compare to I-GP and I-RF with five initialization steps using a strong meta-initialization technique [81]. The results summarized in Figure 7.3 are very similar to those reported in our previous experiment. TST-R again is best for both evaluation metrics but FMLP shows to be a strong competitor.

7.2.3 Scalability Experiment

As discussed in Section 2.6, Gaussian processes are computationally expensive. Training time is cubic in the number of training instances and still quadratic when updating it. Our proposed surrogate model TST makes use of Gaussian processes in a scalable way. For d data sets, where on each of these n observations of different hyperparameter configurations have been made, a traditional Gaussian process that is trained on all instances, as done by SCoT and MKL-GP, has a training time of $\mathcal{O}\left(d^3 n^3\right)$ which is a typical way of using meta-knowledge for Bayesian optimization [4,87]. We

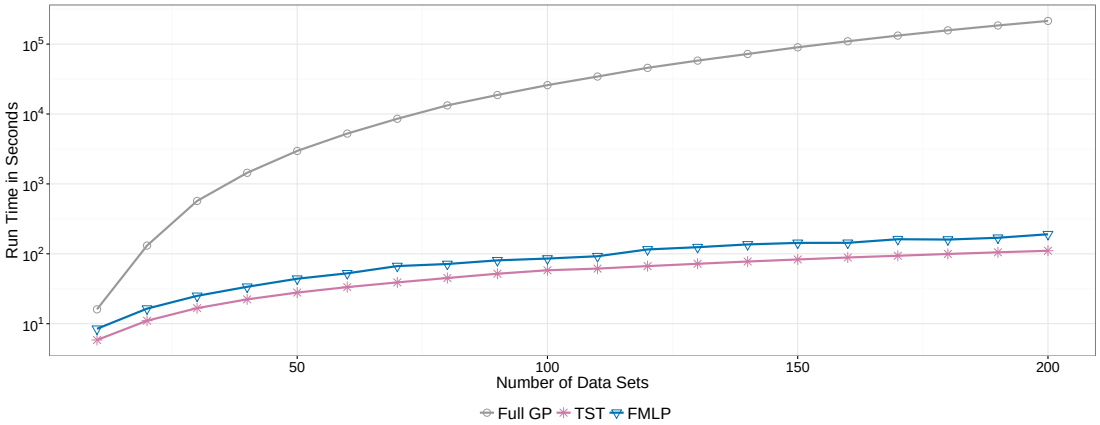

Figure 7.4: TST is clearly outperforming the state of the art that is training a single Gaussian process on the full meta-data with respect to scalability. FMLP, which is based on a neural network, has a training time that is linear in the number of data sets, similar to TST.

propose to learn for each data set an independent Gaussian process which reduces the training time to $\mathcal{O}\left(dn^3\right)$ and is no longer cubic in the number of data sets. In an empirical evaluation we show the impact of an ever-growing meta-data set.

We created an artificial meta-data set with $n = 100$ meta-instances per data set and 5 hyperparameters. We estimated the training time for a Gaussian process on the full data and TST with the number of data sets varying between 10 and 200. The results are visualized in Figure 7.4. At a point where the Full GP needs almost 60 hours of training, TST needs less than 2 minutes.

As discussed earlier, the cubic training time makes Gaussian processes unattractive for large meta-data sets. Hence, our main goal was to achieve training times for Gaussian processes that are competitive to other models such as neural networks as used by FMLP. Figure 7.4 shows that our approach needs time very similar to FMLP. We have to acknowledge that FMLP also scales linearly with an increasing number of meta-instances per data set while the training time of TST is still cubic in this dimension. To overcome this problem one can apply the idea proposed by Hinton[33] which we introduced in Section 7.1.1. Multiple Gaussian processes can be learned per data set. To achieve this, the subsets $\mathbf{X}^{(i)}$, $\mathbf{f}^{(i)}$ defined in Equation (7.1) have to be divided further. One could for example learn an individual Gaussian process for each of the three SVM kernels. If there is no natural way of dividing the data it is no problem to distribute the meta-instances in an arbitrary way. Then, we can apply the method of Section 7.1.1.

7.3 Conclusion

In this chapter, we propose a two-stage transfer surrogate for using meta-knowledge to accelerate Bayesian optimization for machine learning configurations. We propose to approximate the hyperparameter response function of each data set with an individual model. These individual models are finally combined at the second stage to estimate the score of a hyperparameter configuration. In extensive experiments on two meta-data sets, we compare our method to numerous competitor methods published recently on established machine learning conferences. We show empirically that our two-stage transfer surrogate model is able to outperform all considered competitor methods for the task of hyperparameter optimization as well as the task of combined algorithm selection and hyperparameter optimization.

8

Hyperparameter Optimization Machines

Bayesian optimization is one of the most popular methods for finding optimal hyperparameter configurations. Originally designed for black-box optimization, researchers have contributed different meta-learning approaches to speed up the optimization process. In this final chapter on meta-learning for Bayesian optimization, we propose Hyperparameter Optimization Machines, a Bayesian optimization framework that covers recent meta-learning additions. This framework gives access to adaptive hyperparameter transfer learning with plain surrogates (AHT), a new class of hyperparameter optimization strategies. The idea of AHT is as follows. A plain surrogate model, i.e. one that does not use meta-knowledge, is used for Bayesian optimization. Meta-knowledge is used within a transfer function which is combined with the acquisition function. Hence, the choice of the next hyperparameter configuration is stronger controlled by this function and decoupled by the surrogate model. The advantages are that the time- and space-consuming transfer surrogate models are replaced with plain surrogates and that we can directly optimize for a meta-loss which has not been possible with surrogate models so far. We conclude our work on meta-learning for Bayesian optimization by comparing our proposed methods and several competitor methods on two different meta-data sets. In this final comparison, we can show that AHT outperforms various instances of Bayesian optimization in the scenarios of hyperparameter optimization and algorithm selection. Finally, we discuss lessons learned and insights gained.

Bayesian optimization [37], has been proposed for black-box optimization. It was designed for finding an optimum of a function f which is expensive to evaluate. A surrogate model is used that tries to approximate f but has the advantage of being less time-consuming to evaluate. This surrogate model is combined with an acquisition function a to tackle the exploitation-exploration dilemma. Bayesian optimization has been applied to the problem of hyperparameter optimization [68] by minimizing the function f_D defined in Equation (2.4) and has proven to be very effective. Since then, researchers have adapted and specialized Bayesian optimization for the hyperparameter optimization problem.

A specific property of hyperparameter optimization beyond mere black-box optimization is that the hyperparameter response function of a specific algorithm behaves similar on similar data sets. Furthermore, some hyperparameter configurations provide decent results on average which are usually proposed as default hyperparameter configurations. Hence, information about the data sets and information about previous experiments can be used by practitioners to find good initial hyperparameter configurations. This important difference between arbitrary black-box optimization and hyperparameter optimization and the fact that meta-knowledge already helped for other hyperparameter optimization techniques [58], gave rise to various meta-learning approaches based on initialization [23,81], transfer surrogate models [4,61,72,87] and pruning [79] and has empirically proven its usefulness. Finally, the sequential hyperparameter evaluation was adapted to be capable of searching in parallel to exploit the hardware available in our days [68].

This is our concluding chapter on meta-learning for Bayesian optimization. Therefore, we want to propose hyperparameter optimization machines as a generalization of Bayesian optimization which covers the recent contributions in the context of configuration optimization for machine learning applications. Algorithm 9 outlines our aforementioned generalization. Like most Bayesian optimization approaches, it consists of a surrogate model and an acquisition function a. Iteratively, the surrogate model is updated using the observation history $\mathcal{H}$ and the next hyperparameter configuration is selected (Line 3 and 4). In contrast to the state of the art, this is done by using a linear combination of the acquisition function a and a new component, the transfer function $\mathfrak{T}$. We consider the negative value of the acquisition function a because a higher value means better values and we want to minimize the combined term. As will be detailed in the following, this covers Bayesian optimization with initialization as well as our proposed optimization strategy proposed in Section 8.2. The parameter $\boldsymbol{\alpha}$ is a meta-hyperparameter that controls the influence of transferred meta-knowledge and the knowledge gathered about the new data set so far. A reasonable $\boldsymbol{\alpha}$ is one that is close to 0 for small $t \in \mathbb{N}$ and then increases over time to 1. Thus, meta-knowledge plays a major role for the selection of the first hyperparameter configurations and becomes irrelevant as

soon as enough knowledge is gathered about the new data set. After selecting the most promising hyperparameter configuration, it is evaluated and the result is added to $\mathcal{H}$. This is repeated until convergence.

Algorithm 9 Hyperparameter Optimization Machines

Input: Configuration space $\mathcal{X}$, observation history $\mathcal{H}$, acquisition function a, transfer function $\mathfrak{T}$, acquisition function a, trade-off parameter $\boldsymbol{\alpha}$.
Output: Best configuration found.

1: $\mathcal{X}_0 \leftarrow \emptyset, f^{\min} \leftarrow \infty$
2: **while** not converged **do**
3: Update surrogate model (Update)
4: $\mathbf{x}_* \leftarrow \arg\min_{\mathbf{x}_* \in \mathcal{X}} (1 - \boldsymbol{\alpha}_t)\, \mathfrak{T}(\mathbf{x}_*, \mathcal{X}_{t-1}) - \boldsymbol{\alpha}_t\, a(\mathbf{x}_*, p(f_*|\mathbf{x}_*, \mathcal{H}))$ (Predict)
5: $\mathcal{X}_t \leftarrow \mathcal{X}_{t-1} \cup \{\mathbf{x}_*\}$
6: $f_* \leftarrow f(\mathbf{x}_*)$
7: $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\mathbf{x}_*, f_*)\}$
8: **if** $f_* < f^{\min}$ **then**
9: $\mathbf{x}^{\min}, f^{\min} \leftarrow \mathbf{x}_*, f_*$
10: **end if**
11: **end while**
12: **return** $\mathbf{x}^{\min}$

8.1.1 Relationship between HOM and Bayesian Optimization

In the last section we introduced hyperparameter optimization machines (HOM) with the goal to generalize across the contributions that have been done to Bayesian optimization for configuration optimization for machine learning. Hence, we will now map most relevant hyperparameter optimization methods related to Bayesian optimization, as defined in Algorithm 1, to HOM.

Bayesian optimization itself is obviously a specialization of HOM in the case that $\boldsymbol{\alpha} = 1$ such that the transfer function $\mathfrak{T}$ will not be considered or for arbitrary $\boldsymbol{\alpha}$ if $\mathfrak{T}$ is a constant function. In the following, we will ignore that $\mathfrak{T}$ can be a constant function and assume that it somehow reflects the meta-knowledge, meaning that $\mathfrak{T}(\mathcal{X}_t)$ is lower if $\mathcal{X}_t \subset \mathcal{X}$ contains hyperparameter configurations that performed well in previous experiments and vice versa. It is important to notice, that HOM is not an instance of Bayesian optimization with a specific acquisition function. The expression in Line 4 of Algorithm 9 and an acquisition function have in common that they acquire the next hyperparameter configuration but the acquisition function of Bayesian optimization depends only on the hyperparameter configuration and its predicted performance but is independent of the time and previously selected hyperparameter configurations.

The work that is proposing a specific surrogate model for Bayesian optimization[4,35,61,68,72,87] is hence also an instance of HOM. Choosing an appropriate transfer function, Bayesian optimization with I initialization steps[23,81] is a special case of HOM where $\boldsymbol{\alpha}_t = 0$ if $t \leq I$ and 1 otherwise.

Finally, even random search[7] and grid search can be considered as an instance of HOM with a very specific acquisition function and with no need for a surrogate model.

8.2 Adaptive Hyperparameter Transfer Learning with Plain Surrogates

So far two different ways of exploiting meta-knowledge by Bayesian optimization are common. One option is to use an initialization[23,81] and combine it with *plain surrogate models*, models that are learned only on observations of the current data set. The advantage in this case is that it does not need any additional run time during the optimization process. Otherwise, the initialization sequence is static and does not consider the gained knowledge about the response function $f_{D_{\text{new}}}$. In contrast, *transfer surrogate models*[4,61,72,87] (machine learning models that are learned on the observations on the current data and on observations of past experiments on other data sets) adaptively consider the meta-knowledge but they are costly in terms of space and time. Finally, applying meta-knowledge by initialization or transfer surrogate models achieves similar performances[81].

In this section we propose a new instance of hyperparameter optimization machines which we call *Adaptive Hyperparameter Transfer Learning with Plain Surrogates* (AHT). The idea is to make use of the newly introduced transfer function $\mathfrak{T}$ and combine it with plain surrogates while letting $\boldsymbol{\alpha}$ adopt arbitrary values between 0 and 1. This will lead to a new hyperparameter optimization strategy that tries to combine the advantages of both, initialization and transfer surrogate models, and reduce their drawbacks. In this scenario, $\boldsymbol{\alpha}$ is a meta-hyperparameter that is chosen using cross-validation on the meta-training data set. In the following sections, we will derive a transfer function $\mathfrak{T}$ that tries to minimize a meta-loss and theoretically investigate the asymptotic space and time requirements compared to plain and transfer surrogates, respectively.

AHT is also our final proposition of meta-target driven optimization methods and hence we will discuss and compare it to our previous methods that have a similar motivation. In Chapter 4 we present with SMFO our first meta-target driven optimization method. It solely chooses the next hyperparameter configuration based on the meta-knowledge. Hence, it can be considered as an instance of hyperparameter optimization machines that is only using the transfer function. It provides very good results but has some important disadvantages. It requires that the same hyperparameter configurations have been evaluated on all data sets. Furthermore, it is not able to propose the evaluation of hyperparameter configuration which do not appear in the meta-data set.

In Chapter 6 we solved some of these issues. Learning surrogate models for each data set relaxes

the constraint of evaluating the same hyperparameter configurations on all data sets and that only hyperparameter configurations appearing in the meta-data set can be proposed. But the method presented in Chapter 6 is only an initialization strategy and hence does not support the full optimization process.

AHT overcomes all aforementioned problems. Furthermore, it combines the feedback on the new data set and can be applied during the full optimization process. It marks the end of our developing process that resulted into several intermediate solutions and finally leads to a convincing solution.

8.2.1 Evaluating Hyperparameter Configurations based on Meta-Knowledge

So far the evaluation of a hyperparameter configuration is based on an acquisition function that takes the predicted value and uncertainty into account. During this selection, meta-information can be used indirectly with transfer surrogate models. HOMs allow to introduce our new approach AHT that uses plain surrogates and still accelerates the optimization process as well as decreases the needed memory. The use of meta-knowledge is solely based on the transfer function $\mathfrak{T}$. Then Line 4 of Algorithm 9 combines the knowledge about the current data set and the knowledge about past experiments. Yet, it is unclear what properties are required for a good transfer function. We define two requirements. First, hyperparameter configurations that performed well on previous data set should be rated higher than others. Second, with increasing information about the new data set, the influence of the meta-information should vanish. The second requirement can also be achieved by choosing $\boldsymbol{\alpha}$ accordingly but this would inflate the number of meta-hyperparameters for the HOM which we want to avoid.

Since our ultimate goal is to minimize the DTM on the new data set, we use the ADTM on the previous data sets as a proxy for rating the hyperparameter configurations. Thus, we set the transfer function to be the meta-loss defined in Equation (6.1):

$$\mathfrak{T}\left(\mathbf{x}, \mathcal{X}_{t-1}\right) = \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \min_{\mathbf{x}' \in \mathcal{X}_{t-1} \cup \{\mathbf{x}\}} f_D\left(\mathbf{x}'\right) \quad , \tag{8.1}$$

where $\mathcal{X}_{t-1}$ is the set of evaluated hyperparameter configurations after $t-1$ trials. The hyperparameter configuration $\mathbf{x}$ that minimizes Equation (8.1) is the one that has reduced the misclassification rate the most on previous data sets if the hyperparameter configurations in $\mathcal{X}_{t-1}$ have been tried already.

One problem remains. The functions f_D are only partially observed, hence Equation (8.1) cannot be computed for arbitrary hyperparameter configurations $\mathbf{x}$. To achieve this, we replace f_D with plug-

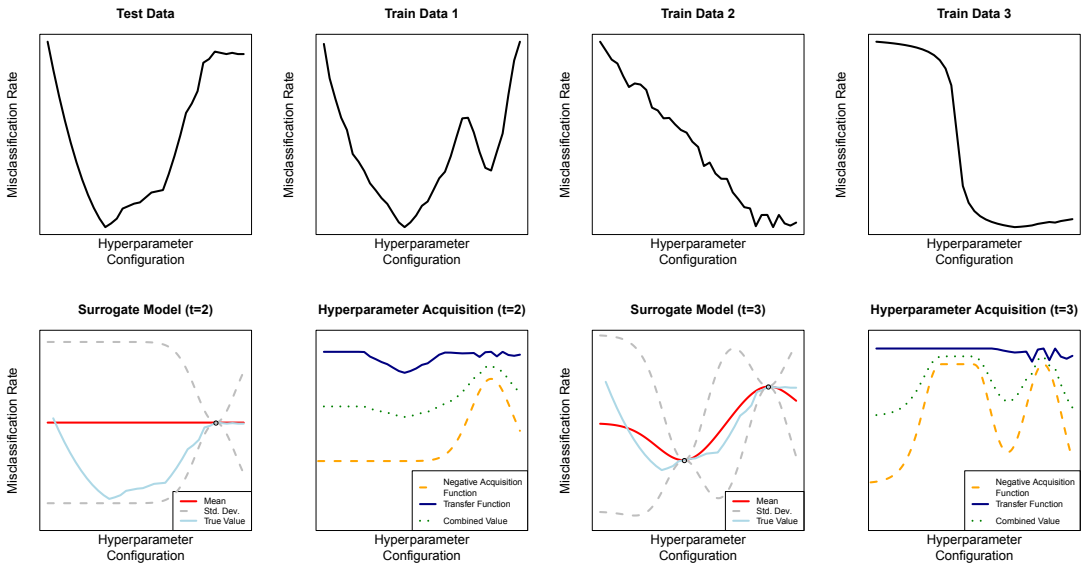

Figure 8.1: First row: Hyperparameter response functions of the current data set where we want to find the best hyperparameter configurations and of three data sets which have been investigated before (meta-knowledge). Second row: Sequential process of AHT. One can clearly see the positive impact of the transfer function on the hyperparameter configuration selection in unexplored areas. In all plots: the lower the better.

in estimators $\hat{f}_D$ that are trained on the available observations of data set D and approximate the true f_D. These observations are part of the meta-data set and hence do not involve any further evaluations of f_D.

The effect of the transfer function for $0 < \boldsymbol{\alpha}_t < 1$ will be twofold. First, it will serve as some kind of soft initialization where likely good hyperparameters are preferred by taking into account the little knowledge that has been gathered on the new data set. Second, it fulfills the criterion that the influence of the meta-knowledge is no longer used as soon as enough trials on the new data set have been performed. This will be achieved by the minimization term in Equation (8.1) that will ensure that the transfer function loses ground over time on the hyperparameter configuration selection decision. The resulting side effect is that we will choose the same value for all $\boldsymbol{\alpha}_t$.

We will explain and illustrate the impact of our proposed combination of the acquisition function with the transfer function in Figure 8.1. In the first row of Figure 8.1 the misclassification rate of a classifier with a one-dimensional hyperparameter space for four different data sets are shown. The task is now to estimate the hyperparameter configuration for the test data (upper left plot) with smallest error using the information gathered on the other three data sets.

For illustration purposes we start at $t = 2$ that means after already evaluating one hyperparameter configuration. We are using a Gaussian process as a surrogate model and expected improvement[37]

as acquisition function. As transfer function we use the function that we derived in Equation (8.1). The surrogate model got updated using the single observation (Figure 8.1; "Surrogate Model (t=2)") and the values of the acquisition function and the transfer function can be estimated (Figure 8.1; "Hyperparameter Acquisition (t=2)"). The standard Bayesian optimization is depending only on the acquisition function (dashed **orange** line) for selecting the next hyperparameter configuration and not directly on the meta-knowledge. Hence, it would choose a hyperparameter configuration at the left border of the hyperparameter search space. Computing the transfer function (solid **blue** line), we can see that the transfer function has lower values in areas where we can expect lower (that means better) function values. Even though data sets 1 and 2 indicate that the right region of the hyperparameter space is good for some data, the transfer function has high values here. The reason for this is that we already evaluated a hyperparameter configuration in that region such that the expected improvement in these regions is small.

Finally, the addition of the transfer function to the Hyperparameter Optimization Machine framework allows us not only to find a balance between exploration and exploitation on the current data set but also on the usage of meta-knowledge by adding some weight on regions that have been good on other data sets (dotted **green** line). Based on the smallest value of the combination of acquisition and transfer function, the next hyperparameter configuration is chosen. Again the surrogate model is updated and the next hyperparameter configuration can be estimated. In this simple example with low-dimensional hyperparameter configurations and little meta-knowledge, one can see already now that the transfer function loses influence over time. As explained before, this is a desired effect because at some point the knowledge on the current data set is sufficient.

This simple example motivates the improvement of AHT over Bayesian optimization without meta-knowledge. If meta-knowledge is used, there are so far two options. One is to use initializations. Compared to AHT this will lead to a fixed number of initial trials no matter if we already know that this is a bad hyperparameter region or not. AHT will not make this mistake since it is using the information of previous trials. The advantage of AHT over transfer surrogates cannot be shown in a simple one-dimensional example but we will try to explain it here and prove it empirically in the course of the chapter. As we have seen in the example, AHT is using the meta-knowledge in a way that the meta-knowledge is losing its influence over time. Transfer surrogates can handle this only to a certain degree. This will be an important issue for transfer surrogates if they are applied to a data set where hyperparameters behave completely different to the majority of data sets in the meta-data set. At any point of time the transfer surrogate is more biased to hyperparameter configurations that have been good on the meta-data set while AHT will at some point ignore the meta-data completely and rely on the data collected of the new data set only.

Table 8.1: Comparison of time and space requirements. The memory requirements of Gaussian processes (GP) with transfer function is only linear in the number of data sets d and the update time is independent of the size of the meta-data set assuming that in each previous experiment n observations are gathered and on the new data set t are gathered so far.

	Plain GP[68,23,81]	Transfer GP[4,72,87]	AHT
Training (offline)	-	$\mathcal{O}(d^3n^3)$	$\mathcal{O}(dn^3)$
Update (online)	$\mathcal{O}(t^2)$	$\mathcal{O}(t^2 + d^2n^2 + dnt)$	$\mathcal{O}(t^2)$
Prediction (online)	$\mathcal{O}(t)$	$\mathcal{O}(t + dn)$	$\mathcal{O}(t + dn)$
Space (online)	$\mathcal{O}(t^2)$	$\mathcal{O}(t^2 + d^2n^2 + dnt)$	$\mathcal{O}(t^2 + dn^2)$

8.2.2 Space and Time Requirements

In the following we discuss the time and space complexity of the three different approaches of using meta-knowledge in HOM: 1) initialization combined with a plain surrogate, 2) transfer surrogate model and 3) adaptive hyperparameter transfer learning with plain surrogates (AHT; Section 8.2). This discussion is led for the case that the surrogate model is based on a Gaussian process (GP), the most widespread approach[4,72,87].

Assuming the meta-knowledge of d data sets is available and for each of these data sets, for simplicity, n observations are available while t is the number of observations of the new, unknown data set. We distinguish between the three most time-consuming operations in the HOM. One is the *training* operation, this includes all operations that can be done before the actual optimization process starts. For transfer surrogate models this includes estimating the parameters of the surrogate model on the meta-data, for initialization techniques estimating the initial hyperparameter sequence and our approach will estimate the plug-in estimators during this step. In comparison to the other operations, this can be done offline and hence the time needed for this operation is of less interest for us. The second operation is *update* (see Algorithm 9). This operation has to be done once for each evaluation of a hyperparameter configuration. It updates the surrogate model such that the newly gathered information is considered. The last operation is *prediction.* This is the operation that evaluates the quality of a single hyperparameter configuration. Table 8.1 summarizes the space and time complexity for the different operations. The transfer surrogate models[4,72,87] need more time for offline and online computations than the combination of a Gaussian process with transfer function, respectively. Gaussian processes need to store the kernel matrix. Hence, the space complexity of transfer surrogates is quadratic in the number of meta-data sets instead of only linear. Obviously, the plain Gaussian process is beneficial in terms of space and time complexity. No offline training is needed in cases without initialization[68] but investing time to use the meta-knowledge pays off in general[23,81]. AHT provides the same complexity as our scalable approach TST proposed in Chapter 7 because both approaches use a Gaussian process per data set. The big difference is how these

Gaussian processes are used to employ meta-knowledge. TST is a typical transfer surrogate model while AHT uses the Gaussian processes only to approximate the response surfaces to query arbitrary values.

8.3 Experimental Evaluation

In our concluding evaluation, we compare all approaches covered in this thesis on two different meta-data sets. First, we compare the methods in the scenario of hyperparameter optimization only. This is carried out on a meta-data set generated on 50 different data sets with the LIBSVM library[12]. This smaller meta-data set allows to include transfer surrogate models that are based on Gaussian processes into our comparison. Finally, we apply our method in the scenario of combined algorithm selection and hyperparameter optimization on a meta-data set generated by using 19 different classifiers of Weka[29] on 59 different data sets.

8.3.1 Optimization Strategies

In our final empirical evaluation, we will compare representatives of the following five types of hyperparameter optimization machines. Those that do not use a surrogate model at all, those that use a plain surrogate model and no meta-knowledge, those that use an initialization to employ meta-knowledge combined with plain surrogates, and finally our proposed method that makes use of the transfer function. These methods are summarized in Table 8.2.

Our proposed method adaptive hyperparameter transfer learning with plain surrogates (AHT) is an instance of hyperparameter optimization machines proposed in this chapter. In the experiments, we will consider AHT with two different plain surrogates: a Gaussian process (AHT-GP) and a random forest (AHT-RF). We estimated the meta-hyperparameter α with leave-one-data-set-out cross-validation on the grid $0.1, \ldots, 0.9$. For the SVM meta-data set the best value for AHT-GP is $\alpha = 0.5$, for the Weka meta-data set it is $\alpha = 0.8$. The the optimal value on the Weka meta-data for AHT-RF is $\alpha = 0.5$. Details about the meta-data sets and the evaluation metrics can be found in Chapter 3.

8.3.2 Hyperparameter Configuration Optimization

The task of hyperparameter optimization is to find the best hyperparameter configuration for a given algorithm. This is the typical scenario for researchers tuning baselines, their own new algorithm or for practitioners that have already decided which algorithm is best for their problem. This problem is putatively easier because the search space is smaller. Nevertheless, it is an important problem in practice.

Table 8.2: Brief overview of the competitor methods. An explanation of this table is given in Section 3.2.3.

Name	Abbrev.	Surrogate Model	Meta-Knowledge	Details
Random Search	Random	None	✗	Sec. 2.2
Independent Gaussian Process	I-GP	Gaussian Process	✗	Sec. 2.7.1
Independent Random Forest	I-RF	Random Forest	✗	Sec. 2.7.1
Independent Gaussian Process with Meta-Initialization	I-GP (init)	Gaussian Process	✓	Chap. 6
Independent Random Forest with Meta-Initialization	I-RF (init)	Random Forest	✓	Chap. 6
Surrogate Collaborative Tuning	SCoT	SVMRank + Gaussian Process	✓	Sec. 2.7.2
Gaussian Process with Multi-Kernel Learning	MKL-GP	Gaussian Process	✓	Sec. 2.7.2
Factorized Multilayer Perceptron	FMLP	Neural Network	✓	Sec. 2.7.2
Two-Stage Transfer Surrogate	TST	Gaussian Process	✓	Chap. 7

Table 8.3: AUC-ADTM of the optimization strategies on the SVM meta-data set. Number in brackets indicate the ranking across the strategies.

Random	I-GP	I-RF	SCoT	MKL-GP
4.892 (9)	3.146 (7)	3.870 (8)	2.938 (5)	3.095 (6)
FMLP	**I-GP (init)**	**TST**	**AHT**	
1.453 (3)	1.633 (4)	1.237 (2)	1.220 (1)	

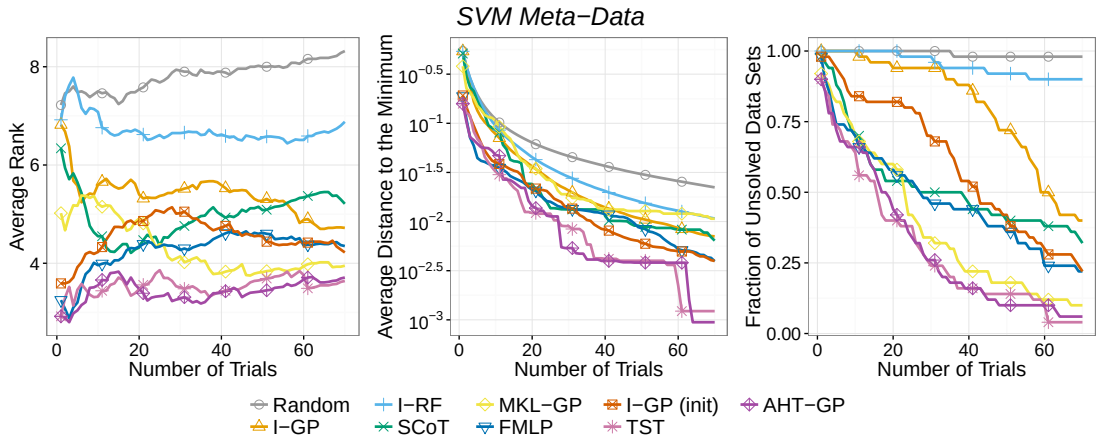

Figure 8.2: Our proposed method AHT outperforms seven competitor methods with respect to all three evaluation metrics.

MKL-GP tends to provide good results on most of the data sets and is able to find the best hyperparameter configuration in few trials (see Figure 8.2 right) but fails to find any good hyperparameter configuration on few data sets which leads to a comparable bad ADTM which is inferior to I-GP with initialization (see Figure 8.2 middle). Remarkable are also the good results of I-GP using an initialization, being very competitive considering the simplicity. TST and AHT outperform the competitor methods with respect to all three evaluation metrics on the SVM meta-data set. The improvement over other methods within the first ten trials is small but increases later on. The reason for this is that AHT is based mainly on the meta-knowledge as the competitor methods. At the point where the meta-knowledge can no longer be exploited for guiding the search, AHT's special mechanism is used to expand its leading position. TST has a mechanism that works very similar but is not that principled. The influence of the meta-knowledge is dependent on the number of configuration pair disagreements. Since this is naturally growing with the number of trials, the impact of the meta-knowledge vanishes over time. In practice, this seems to lead to comparable performance.

Furthermore, we compare all optimizers with respect to the AUC-ADTM metric as defined in Equation (4.1) in Table 8.3. These results confirm the previous discussion and nicely summarize

Table 8.4: AUC-ADTM of the optimization strategies on the Weka meta-data set. Number in brackets indicate the ranking across the strategies.

I-GP	I-RF	FMLP	I-GP (init)
12.824 (6)	27.221 (8)	7.214 (3)	10.817 (5)
I-RF (init)	**TST**	**AHT-GP**	**AHT-RF**
15.277 (7)	7.951 (4)	6.724 (2)	5.454 (1)

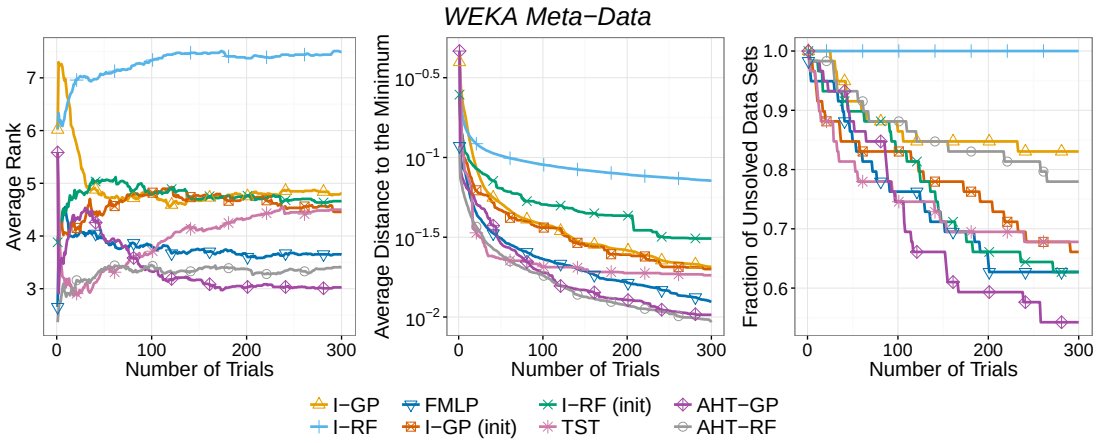

Figure 8.3: AHT with two different surrogate models achieves the best ADTM on the Weka meta-data set but the combination with a Gaussian process leads to finding optimal hyperparameter configurations in more cases.

Figure 8.2.

8.3.3 Combined Algorithm Selection and Hyperparameter Optimization

In this section we want to empirically investigate the performance of the different hyperparameter optimization strategies in the scenario of combined algorithm selection and hyperparameter optimization. This problem leads to larger meta-data sets (about 1.4 million meta-instances) which did not allow us to commit these experiments for the transfer surrogates that are based on Gaussian processes. Nevertheless, we still compare to the transfer surrogate models FMLP and TST for which we have shown in the previous chapter that they are outperforming the omitted competitor methods.

In the previous experiments we always combined AHT with a Gaussian process. Since it was previously reported that the random forest as a surrogate model provides better results for problems with high-dimensional and discrete hyperparameter spaces[22] which is the case on the Weka meta-data set (more than 60% are indicator variables), we also provide results for AHT combined with a random forest. For us, the most promising advantage of a random forest is the shorter training time

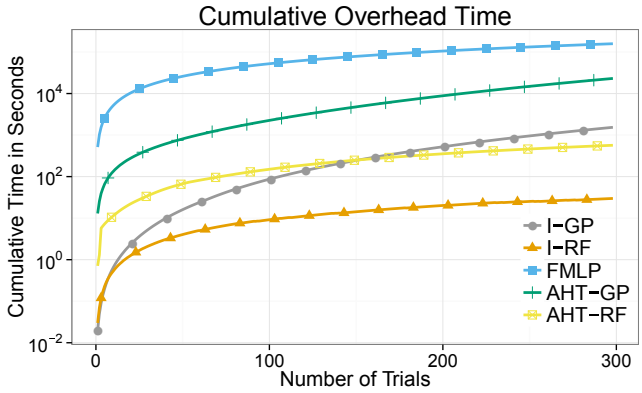

Figure 8.4: Strategies based on transfer surrogates are the slowest among all investigated methods. AHT provides the best performance for a reasonable time overhead.

compared to a Gaussian process. We also committed these experiments on the SVM meta-data set but the results were worse than the combination with a Gaussian process and thus we omitted them to avoid overcrowded figures.

Figure 8.3 summarizes the results. Surprisingly, initialization (I-GP (init), I-RF (init)) did not provide good results for this meta-data set. This is another indication that stresses that the soft and adaptive initialization effect of AHT is better than a hard initialization. Hence, the transfer surrogate models provide better results than an initialization. While TST had comparable results to AHT on the SVM meta-data set, this is not the case for the Weka meta-data set. It provides very good results in the beginning but then is not able to improve further such that FMLP surpasses it. AHT provides the best results. The combination of AHT with a Gaussian process (AHT-GP) achieves similar results with respect to the ADTM compared to AHT-RF but finds the optimum on many data sets faster than AHT-RF.

The results with respect to AUC-ADTM in Table 8.4 confirm our visual interpretation of Figure fig:DSAA2016-result-weka. The two AHT variants perform best, followed by TST. FMLP is providing better results than I-GP and I-RF with initialization in contrast to the results on the SVM meta-data set.

8.3.4 Run Time

Finally, we provided different methods for reducing the time overhead introduced by the different optimization strategies in this thesis. With TST and AHT we propose two strong methods for configuration optimization which are able to scale to large meta-data sets. Figure 8.4 summarizes our

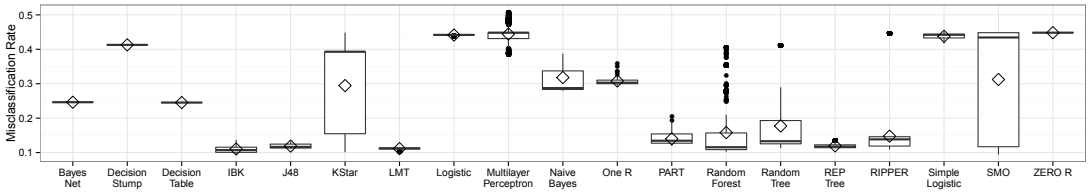

Figure 8.5: Distribution over all hyperparameter configurations for different algorithms of the data set *banana*.

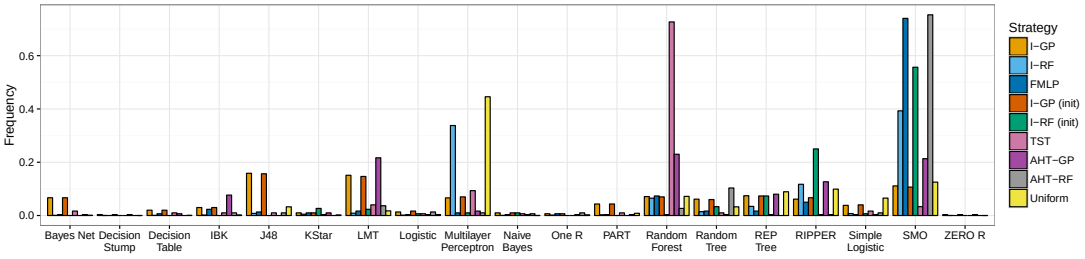

Figure 8.6: Selection frequency of evaluating the performance of a hyperparameter configuration for a specific hyperparameter configuration. If the value is higher than the uniform distribution, this algorithm was preferred by the optimization strategy.

run time results. Unsurprisingly, strategies relying on plain surrogate models (I-GP, I-RF) are the fastest. We have seen that combining plain surrogates with an initialization (that does not result in an time overhead during the optimization) achieved good results for hyperparameter optimization but they showed less convincing results in the setting of combined algorithm selection and hyperparameter optimization. Transfer surrogates (FMLP) are by orders of magnitudes the slowest approach but have found good hyperparameter configurations in both scenarios. Finally, we achieved our goal of combining the advantages of both approaches. AHT provides the best results in both scenarios but has less time overhead than transfer surrogates for finding good hyperparameter configurations on average. TST achieves the same run time as AHT-GP but provides worse configurations in our experiments.

8.3.5 Case Study

For an in-depth understanding of how the different configuration optimization strategies work, we select one data set for a deeper analysis. We select the *banana* data set because none of the optimization strategies was able to find the optimal hyperparameter configurations within 300 trials. Figure 8.5 shows the hyperparameter performance distribution for the different algorithms giving first insight why this data set is actually that difficult to optimize for. There are many different algorithms achieving small misclassification rates and hence it is difficult to narrow down the search

to just few algorithms. Figure 8.6 gives insight with what frequency each optimization strategy has selected a specific algorithm for evaluating a hyperparameter configuration. Our Weka meta-data set is limited to hyperparameter configurations that we have evaluated beforehand to make this experiment possible. Because the different algorithms have a different number of hyperparameters, the number of test meta-instances varies between the different algorithms. The uniform distribution shows the fraction of test meta-instances per algorithm and hence can be used as an indication whether an optimization strategy prefers an algorithm or not. If the value of the uniform distribution is higher than the value of the optimization strategy, the optimization strategy does not believe that the optimum can be here and vice versa. Thus, it can be seen that the optimization strategies are capable of identifying that a multilayer perceptron does not achieve good performance on this specific data set while k-nearest neighbors (IBK) does.

8.4 Conclusion

In this final chapter on meta-learning for Bayesian optimization, we propose hyperparameter optimization machines as a generalization of several Bayesian optimization approaches which includes current meta-learning extensions for the use in the hyperparameter optimization context. This generalization allows us to focus on the new hyperparameter optimization strategy AHT which uses meta-knowledge in an adaptive fashion and combines it with time- and space-efficient plain surrogate models. In experiments on two different meta-data sets for the problem of hyperparameter optimization as well as combined algorithm selection and hyperparameter optimization, the advantage of AHT compared to various other hyperparameter optimization strategies is shown empirically. We are able to show that AHT produces less time-overhead for the optimization than most other transfer surrogates by outperforming all competitor methods. However, we acknowledge that plain surrogates using an initialization are still the method with least overhead but this approach does not achieve good results in the scenario of combined algorithm selection and hyperparameter optimization.

9 Conclusion

We want to finish this part of the thesis on transfer-learning for Bayesian optimization for machine learning configurations by summarizing our findings and works. We gained three key insights about meta-knowledge in Bayesian optimization.

1. Meta-knowledge provides useful information that can be used in the context of Bayesian optimization for machine learning.
2. There is an unknown time at which information gained on the new data set is provides more information than meta-knowledge.
3. Optimizing for the correct meta-loss is beneficial.

Insight 1 have been the motivation for all our proposed methods and it usefulness can be confirmed in Chapter 4 to 8. The pruning approach proposed in Chapter 5 profits additionally from insight 3. In the beginning all configurations are discarded that are considered bad with respect to the improvement on the meta-knowledge. Similarly, the initialization technique presented in Chapter 6 employs insight 3 by directly optimizing for a meta-loss. One may argue that it also employs insight 2 because it does not use meta-knowledge after the initialization. But insight 2 only tells us that meta-knowledge becomes less important than the information gained over the new data set and not that it becomes irrelevant. Furthermore, the point at which meta-knowledge becomes less important changes from data set to data set such that a fixed initialization length is non-optimal and does not fully exploit insight 2.

Typical transfer surrogate models, such as SCoT and FMLP, use insight 1 but only partially insight 2. The meta-knowledge is never really explicitly discarded but it is tried to reduce its influence by adding new meta-instances from the new data set. TST, our surrogate model proposed in Chapter 7, introduces a heuristic to partially use insight 2. The influence of the meta-knowledge and the new data set is combined with a decaying weight for the meta-knowledge influence. Thus, it will reach a time where no meta-knowledge is used any more. Similarly to the aforementioned initialization strategy, it tries to fully use insight 2 but does not do it in a principled way.

Finally, in this chapter we presented AHT. AHT is our final proposition that makes use of all three insights. Configurations are chosen with respect to the improvement on the meta-knowledge and the expected improvement on the new data set. The impact of the meta-knowledge is determined on how much improvement is still left. In the beginning, we fully rely on it but later its impact is marginally and in the extreme case no longer existing. Our experiments have shown that this is the best methods and hence confirms our insights.

I consider TST and AHT the most promising approaches for future work in the field of Bayesian optimization. For this reason, we proposed a novel method, not presented in this thesis, which combines the ideas of AHT and TST[86]. The meta-knowledge is combined as in TST but it is considered in the acquisition function and not the surrogate model as by AHT.

Furthermore, the idea of learning initializations presented in Chapter 6 is an elegant method which allows to transfer knowledge from previous experiments to new ones which is not limited to Bayesian optimization. This makes it a more general approach and widely applicable.

Part III

Applied Bayesian Optimization

10

Distributed Hyperparameter Optimization & Applications

The goal of this thesis is to improve techniques that support machine learners by automating machine learning. While we focus in the previous chapters mainly on lab experiments, we focus stronger on comparison to human experts in this and the next chapter. In this chapter we will discuss an implementation for distributed hyperparameter optimization developed by me in the past years. It proposes a server/client architecture where the server is gathering the results of the hyperparameter optimization and controls which client is evaluating which configuration. The clients are responsible for the computational expensive part, i.e. the training of machine learning models for given configurations. I applied this tool to two different ECML-PKDD challenges. In one of them I used it to support me, in the other it was working fully autonomously. The results were astonishing, allowing me to place first and third in prestigious challenges on one of the most important European machine learning conferences.

10.1 easyOpt

We will discuss in this section how *easyOpt*, my distributed machine learning configuration optimization tool works. It is based on a server that is managing the whole optimization process and clients, that are evaluating different configurations. The communication between servers and clients is realized with Java Remote Method Invocation.

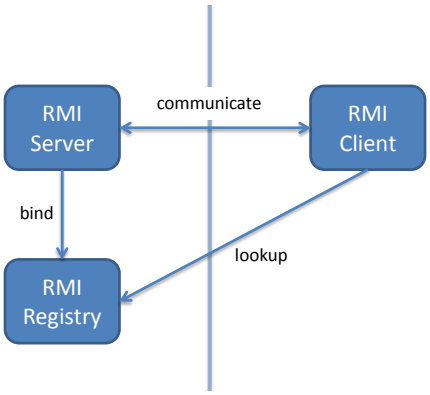

Figure 10.1: The server first registers at the RMI registry. Clients are then able to access references to the remote objects. This allows to finally invoke the remote methods.

10.1.1 Java Remote Method Invocation

The Java Remote Method Invocation (RMI) system allows a Java object to invoke methods of remote Java objects running on a remote Java virtual machine. An RMI application consists of a server and a client program. The server makes references of objects publicly accessible. This is done by providing stubs. A stub acts as a proxy for the remote project and implements the same set of remote interfaces that the remote object implements. It is obtained by the clients and used to invoke remote methods. RMI provides mechanisms for transparent communications between server and clients in this distributed object application. It enables to locate remote objects via the RMI registry, conducts the communication between remote objects and is loading class definitions for remote objects.

10.1.2 Detailed Description of easyOpt

We kept the interface to the optimization tool as simple as possible to ease its use. Figure 10.2 presents the interface in a class diagram. It contains two main classes. The class EasyOpt provides abstract methods to start a new optimization process as well as joining one by communicating with the RMI registry. The interface HyperparameterOptimization provides functions for communications between server and client. The only communications needed is to ask for the next configuration to evaluate and report the results to the server.

Figure 10.3 presents a scenario for a typical collaborative optimization process. The server program will call one of the *start** methods to bind the remote object's stub in the RMI registry and hence, allow clients to join the process. Clients can now call the method *joinOptimization* to lookup the stub in the RMI registry. Then, they have access to the methods of a HyperparameterOptimization instance which allows to participate in the optimization process. They ask for a configuration

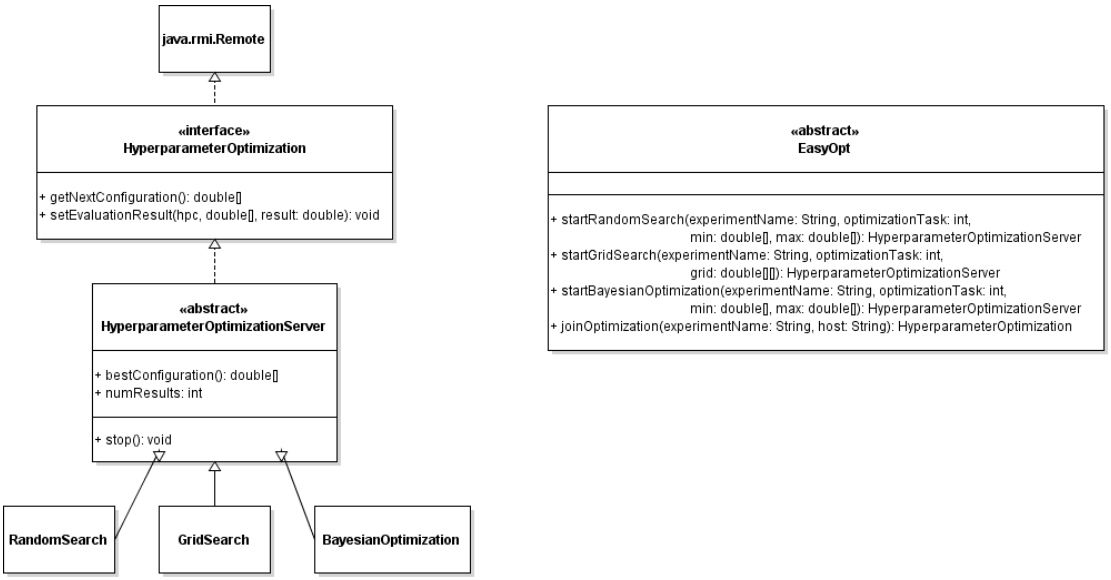

Figure 10.2: The library contains two main classes. The abstract class EasyOpt manages the communication with the RMI registry, the class implementing HyperparameterOptimization manages the communication between server and client.

to evaluate, run the program with the received configuration and finally report the result. Using Bayesian optimization, whenever the server receives a result, the surrogate model is updated. Is the end of the optimization reached, the server informs asking clients such that they can finish their processes. As soon as the last answers are received, the server finally unbinds and exits.

The tool provides three different optimization techniques, i.e. grid search, random search and Bayesian optimization. The reader may wonder why it is useful to run grid or random search in a centralized fashion. One reason is that all results are gathered at a central position and can thus be used to e.g. rerun the experiment on the full data with the best configuration. Another reason is that data sets are not loaded more than necessary. Typically, compute jobs are created for each configuration. They are enqueued in a compute cluster and as soon as they are scheduled, they load the data and evaluate their configuration. Finally, they free the memory, report the result and finish. A job processing the next configuration is started then. The centralized approach saves here time because it is not needed to reload the data again. This is especially useful for large data sets.

10.1.3 Code Example

As explained before, the aim of easyOpt is too keep it as simple as possible. To show that its usage is indeed very simple, we provide a short example that demonstrates how to optimize the hyperparameters of a logistic regression with Bayesian optimization.

The server needs to start the optimization by providing a unique experiment name, in this case

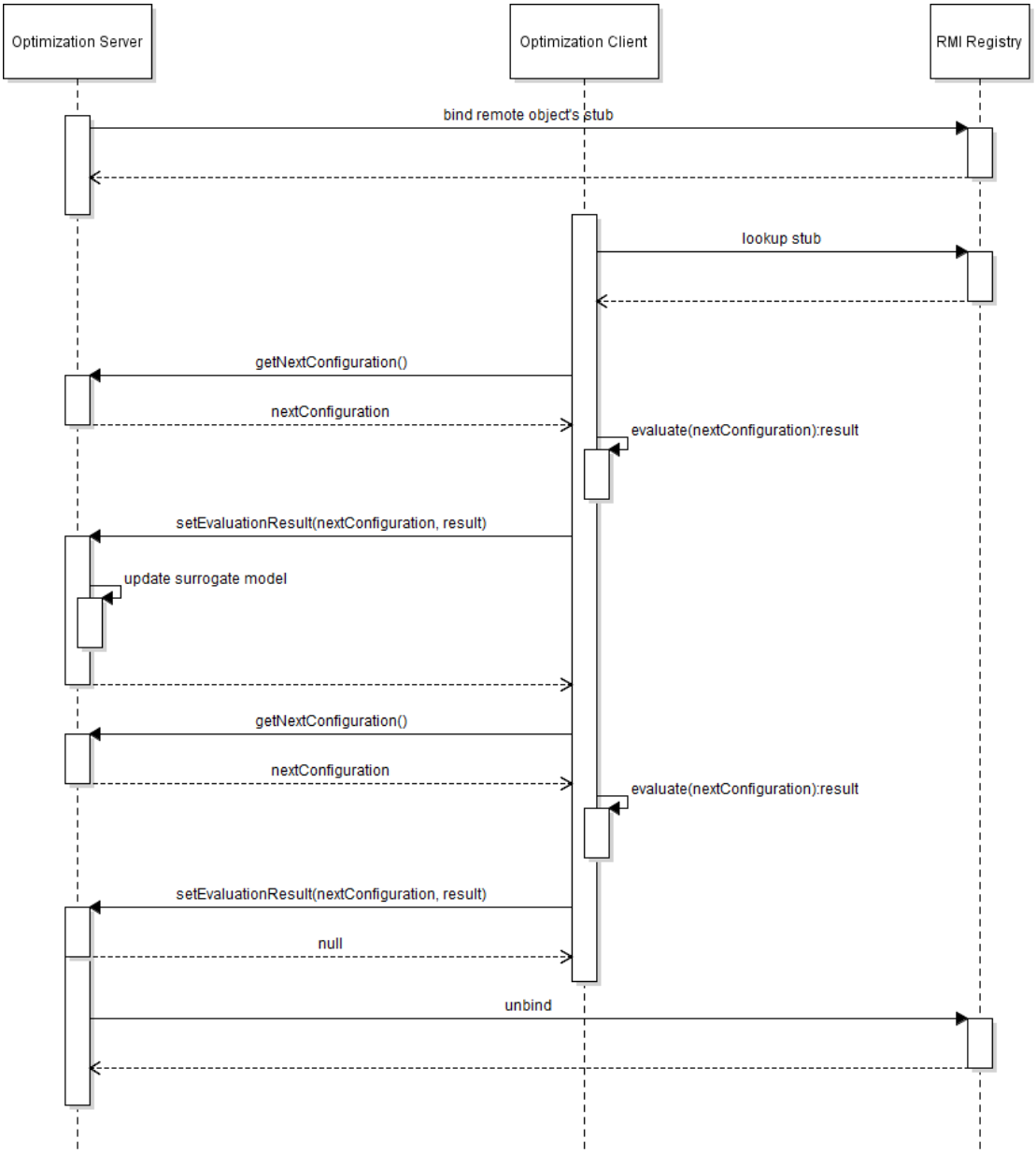

Figure 10.3: The server first binds the remote object's stub to the RMI registry. Now, clients can lookup the stub and join the optimization. Each client is now sequentially asking for a configuration, evaluates it and reports the result. As soon as the optimization is finished, the server waits until all results are gathered. Then it unbinds the stub from the RMI registry and finishes the process.

"mnist". Depending on the hyperparameters, the boundaries need to be set. In our case, the learning rate, regularization constant and the number of stochastic gradient descent epochs are the hyperparameters. The types of the hyperparameter needs to be set. We distinguish between floats and integers. Finally, it is important whether we are interested in the maximum or minimum. The clients need to be aware of how hyperparameters are encoded on the server's side.

The implementation of the client looks a little bit more complex. Based on the unique experiment name, the client can lookup the stub given the host name. In a while loop hyperparameter configurations are evaluated until the server signals that the optimization is done.

```java
// Server.java

import org.easyopt.EasyOpt;
import org.easyopt.HyperparameterOptimizationServer;

public class Server
{
  public static void main(String[] args) throws InterruptedException
  {
    int numTrials = 100;
    String experimentName = "mnist";

    double[] min = { -5, 0, 50 };
    double[] max = { 0, 1, 500 };
    int[] hyperparameterType = { 1, 1, 0 };
    HyperparameterOptimizationServer server =
        EasyOpt.startRandomSearch(experimentName, EasyOpt.MINIMIZE, min,
        max, hyperparameterType);

    // The server waits until all results are gathered. Clients are already
        freeing resources.
    while(server.getNumResults() < numTrials))
    {
      Thread.sleep(1000);
    }
    double[] bestConfiguration = server.getBestConfiguration();
    server.stop();
  }
}
```

```
// Client.java

import org.easyopt.EasyOpt;
import org.easyopt.HyperparameterOptimization;
import org.easyopt.classification.LogisticRegression;
import org.easyopt.util.Instances;

public class Client
{
  public static void main(String[] args)
  {
    String experimentName = "mnist";
    String host = "127.0.0.1";

    // Load the data
    Instances train = new Instances(new File("data/mnist.scale"));
    Instances valid = new Instances(new File("data/mnist.scale.t"));

    // Join the optimization
    HyperparameterOptimization optimizer =
        EasyOpt.joinOptimization(experimentName, host);

    double[] nextConfiguration;
    while((nextConfiguration = optimizer.getNextConfiguration()) != null)
    {
      // Parse the next hyperparameter configuration to evaluate
      double learnRate = Math.pow(10, nextConfiguration[0]);
      double regularization = nextConfiguration[1];
      int numEpochs = nextConfiguration[2];

      // Evaluate the hyperparameter configuration
      LogisticRegression lr = new LogisticRegression(learnRate,
          regularization, numEpochs);
      lr.train(train);
```

```
      optimizer.setEvaluationResult(nextConfiguration, lr.getError(valid));
    }
  }
}
```

10.2 Experimental Evaluation

We compared our optimization library to human machine learning experts by participating in two different ECML-PKDD Discovery Challenges in 2016. For the challenge on Bank Card Usage Analysis, we used the library to support the machine learning experts. For the challenge on Network Traffic Classification, we fully relied on the library. In the first challenge we placed first, in the second third.

10.2.1 ECML-PKDD Discovery Challenge 2016 on Bank Card Usage Analysis

The goal of one the ECML-PKDD Discovery Challenge on Bank Card Usage Analysis was to predict the behaviour of customers of the Hungarian bank *otpbank*. The task was to predict for every bank branch the number of visits for a set of customers. For this task, anonymized customer and bank information as well as customer activities were provided. Table 10.1 summarizes the features available for each customer, Table 10.2 summarizes the information available about the customer activities. The only information about the bank branches is their location (Table 10.3).

A labeled data set for 2014 was made available which can be used for supervised machine learning to predict the targets for a *disjoint* set of customers for 2015. Activity data for the test customers is provided only for the first six months in 2015. Bank branch visit activities are not provided. In total the data set contains 191,238 customers for 2014 and 191,237 for 2015. 8,174,244 customer activities are provided for 2014, 4,187,128 for the first half of 2015, excluding branch visit activities. The total number of bank branches is 323.

The evaluation measure for this challenge was defined as the average of cosine@1 and cosine@5 for every customer c, where

$$\text{cosine@k} = \frac{\sum_{i=1}^{k} y_{c,i}\hat{y}_{c,i}}{\sqrt{\sum_{i=1}^{b} y_{c,i}^2}\sqrt{\sum_{i=1}^{k} \hat{y}_{c,i}^2}} \tag{10.1}$$

with $y_{c,i}$ being the number of times the customer c has visited bank branch i and $\hat{y}_{c,i}$ the prediction, respectively. There are b different branches in total.

Table 10.1: Customer data available for train and test customers.

Features	Description
USER_ID	Unique user id
AGE_CAT	Age category in 2014. a = -35, b = 36-65, c = 65+
LOC_CAT	Location category of the user. a = capital, b = city, c = village
INC_CAT	Income category. possible values are a = low, b = medium, c = high, d = no income
GEN	Gender. 1 = male, 0 = female
LOC_GEO	Geo info of user address is rounded to 100m
C201*-	Binary columns for each month. If True, the user has at least one credit card.
W201*-	Binary columns for each month. If True, the user is categorized as "wealthy" in the system of the bank.

Table 10.2: Activity time series. The label can be extracted by counting the number of times CHANNEL is "b". No activities with CHANNEL "b" are given for the test data.

Features	Description
USER_ID	Unique user ID
POI_ID	Unique shop ID
CHANNEL	Type of activity. p = pos, n = webshop, b = branch
DATE	Date of activity
TIME_CAT	Time rounded to a = 05-11h, b = 12-18h, c = 19 -04h
LOC_CAT	Event location category. a = capital, b = city, c = village
MC_CAT	Anonymized market category groups. Types are indexed from a ... j
CARD_CAT	Credit vs. debit card. c = credit card, d = debit card
AMT_CAT	Amount of money spent in three categories. a=low, b=medium, c=high
GEO	Geolocation information of the event

Table 10.3: Only the geolocation about the bank branches is provided.

Features	Description
POI_ID	Unique id
GEO	Geolocation information

Problem Identification

We make the assumption that there is no relation between the number of visits of a customer among branches. This enables us to tackle b different regression tasks for each of the b branches. Independently, we train a regression model for each branch that predicts for each customer how often she will visit the branch based on past information for the branch. This is a classical example for count data and hence, we tackled this task as a Poisson regression problem. As part of the evaluation, we need to select five bank branches for which we want to make predictions. We simply choose the five bank branches with highest predicted number of visits which is the best way to achieve a good score in case the predictor performs reasonable. We use gradient boosted decision trees[15] as the prediction model.

Data Preprocessing

For the feature and hyperparameter selection we split the labeled data set into a training data set D_{train} and a validation data set D_{valid} such that the performance on D_{valid} will reflect the performance on the hidden test data. The task is to infer from some customers and their activities in 2014 the behaviour of a disjoint set of customers in 2015. Only customer information as well as the customer's activities of the first half of 2015 (excluding branch visits) is given for the test customers. Thus, we decide to split the given labeled data set by customers, selecting 80% for D_{train} and the remaining 20% for D_{valid} uniformly at random. Only the first six months of activities of the validation customers (excluding branch visits) is provided for validation purposes. The only problem here is that we are actually predicting from data from 2014 for customers in 2014 but there is no way to overcome this problem.

Very basic information of the customers is available including age, location, income and gender. While gender is by nature binary, the other features are already binned into three categories. We employ this information as features after transforming them via one-hot encoding. Furthermore, the internal classification of a bank whether the customer is considered as wealthy or not is given for each month. We distinguish customers of following five categories: customers that have been classified as 1) wealthy in all observed months, 2) not wealthy in all observed months, 3) first wealthy and then changed to not wealthy, 4) first not wealthy and then changed to wealthy, 5) those who changed their classification more than once. Applying one-hot encoding, we add this information as features. Finally, the information in what month the customer possesses a credit card of the bank is provided. Analogously to the five categories of the wealthy classification, we create categories for the credit card time-series information.

Besides using basic customer features, we want to use the information of the customer's activities.

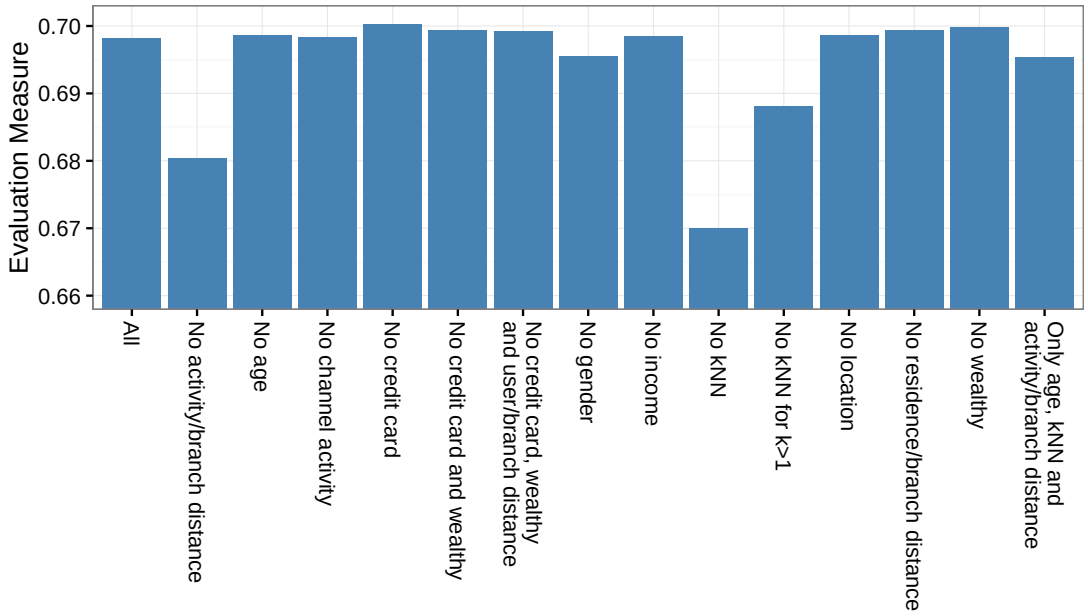

Figure 10.4: Intermediate feature backward selection results. Location-aware features provide huge improvements.

The only feature we use is the number of activities per channel committed by the customer.

Location information about activities, bank branches and customers provide one of the most impactful features. One feature we use is the distance between the residence of the customer and a bank branch which is a quite obvious choice. Digging into the data, we see that there are many customers using bank branches very far away from their residence. We try to cover this by also adding the maximum, minimum, mean and median distance between a bank branch and the customer's activities. Finally, we add k-nearest-neighbors predictions for $k = 2^0, 2^1, \ldots, 2^{10}$ using the Euclidean distance between the residence of customers as the distance function. These features follow the simple assumption that customers that live nearby visit the same bank branches. Figure 10.4 provides insight into our intermediate feature selection experiments and clearly shows the importance of the location-aware features. Based on this experiment, we use all features but the credit card information. Figure 10.5 shows the relative frequency of a specific feature being taken as a splitting variable. Again, this shows the importance of location-aware features.

Hyperparameter Optimization

We used easyOpt to find the hyperparameter configurations of our predictor. Figure 10.6 presents the progress of the optimization process that was conducted in parallel on 100 cores for our own train/validation split as well as results on the public leaderboard.

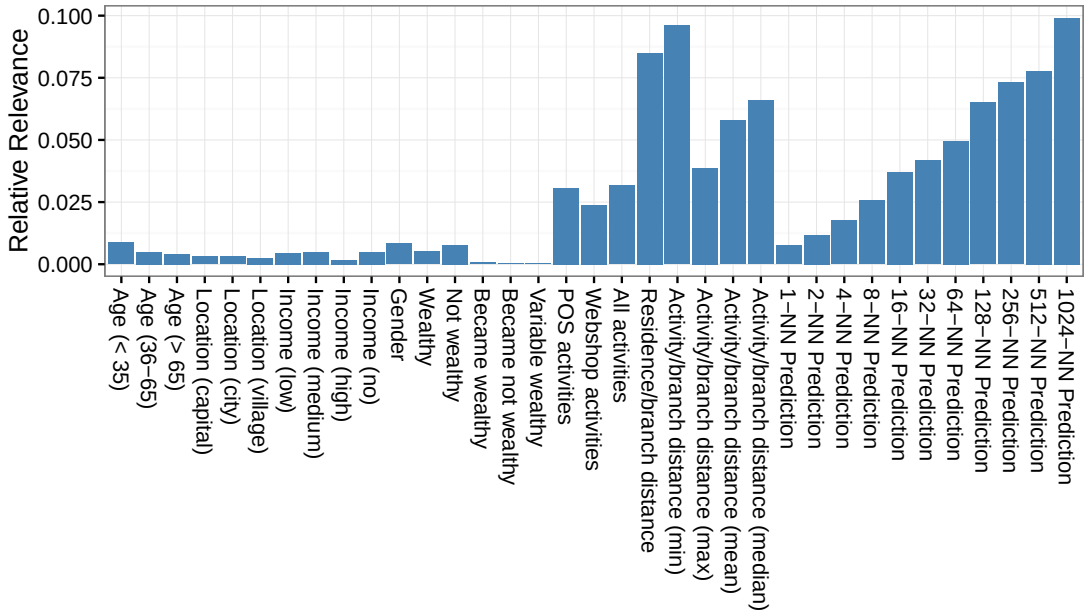

Figure 10.5: This plot visualizes the relative relevance of all features used. The higher the score, the more often the feature was used for building a tree. Location-aware features prove to be highly predictive.

One of our best single models achieved a cosine@1 score of 0.676 and a cosine@5 score of 0.728 leading to an overall score of 0.702 on our validation split. The performance on the test set is with 0.687 much smaller. A possible reason might be temporal effects because the predictions for the test customers are for 2015 but we learn on data from 2014.

The results on the public and private leaderboard of the top five performing teams is provided in Table 10.4. We achieved the best position on both boards.

Table 10.4: The performance of the top five teams for both tasks on the public and private leaderboard.

Public Leaderboard	
Team	Score
1. Wistuba	0.68659
2. Ya	0.68512
3. Cosine Vinny	0.67436
4. Outliers	0.65607
5. seed71	0.65287

Private Leaderboard	
Team	Score
1. Wistuba	0.68812
2. Ya	0.68583
3. Cosine Vinny	0.67503
4. Outliers	0.65684
5. HoliGeOe	0.65310

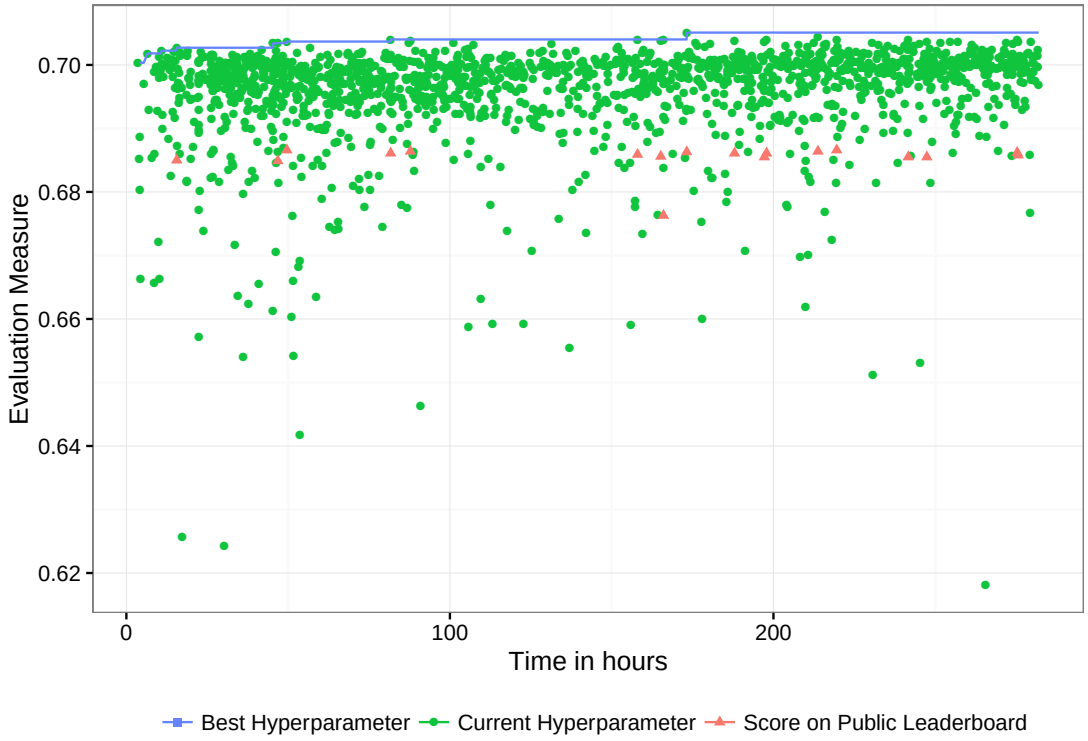

Figure 10.6: Searching for a good hyperparameter configuration with 100 cores in parallel. The public leaderboard score is shown for some of the best hyperparameter configurations on our validation set.

Table 10.5: Details on how the data sets have been created.

Dataset	Train	Validation	Test
Interval	0% – 20%	40% – 60%	80% – 100%
Start Time	2016-02-14 23:00:01	2016-02-15 10:43:44	2016-02-15 15:27:07
End Time	2016-02-15 8:22:35	2016-02-15 13:06:29	2016-02-15 23:00:00
Instances	761179	761179	761180

10.2.2 ECML-PKDD Discovery Challenge 2016 on Network Traffic Classification

For the ECML-PKDD Discovery Challenge 2016 on Network Traffic Classification a data set was generated by using a probe to monitor and measure network activities. The type of measurements are listed in Table 10.7 and 10.8. The task of the challenge was then to predict the type of application based on a transmission in the network.

The data has been collected for an entire day and split chronologically. The available labeled data (train and validation) has been recorded in the night, morning and noon. The test data has been recorded in the afternoon and evening. Data between the splits has been removed to avoid direct temporal dependencies between instances. Details about the splits are provided in Table 10.5. Applications have been separated into 20 different categories. This includes the category "Unknown Application" which contains applications with very few instances. This category is not considered for evaluation.

For this challenge we fully relied on easyOpt, we did not even preprocess the data manually. We selected gradient boosted decision trees as the classification model and defined the range of the hyperparameters. Then, easyOpt was used for about eight days on 100 cores to find the best configuration using the predefined train and validation split. The best configuration was used to retrain the model on train and validation and predict for test. Among 25 participants we placed third right behind the teams of the University of Trento and IBM Research. The detailed results are provided in Table 10.6, results of the hyperparameter search are presented in Figure 10.7.

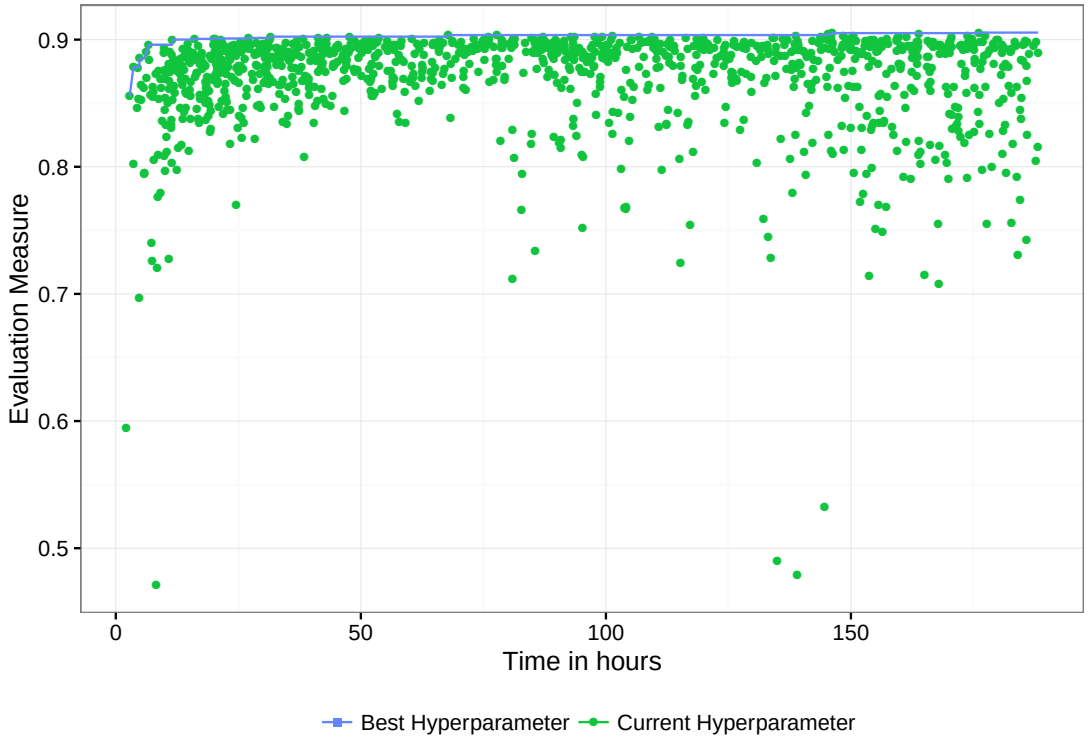

Figure 10.7: Searching for a good hyperparameter configuration with 100 cores in parallel for the Network Traffic Classification Challenge.

Table 10.6: Results of all participating teams with respect to Micro/Macro Precision/Recall/F1. The Macro-F1 measure was used to determine the winner.

Team name	Mic.-P.	Mic.-R.	Mic.-F1	Mac.-P.	Mac.-R.	Mac.-F1
1. UNITN-CogNet	0.9862	0.9745	0.9803	0.9448	0.8521	0.8961
2. IBM-CogNet	0.9777	0.9795	0.9786	0.8965	0.8831	0.8897
3. Wistuba	0.9878	0.973	0.9803	0.9306	0.8509	0.8889
4. FIIT_STU	0.9874	0.9723	0.9798	0.9221	0.8487	0.8839
5. colastrong	0.9882	0.9729	0.9805	0.9279	0.8431	0.8835
6. WekaOne	0.9881	0.9691	0.9785	0.9327	0.8342	0.8807
7. TREELOGIC	0.9875	0.9691	0.9782	0.928	0.8354	0.8793
8. Tubthumpers	0.9866	0.9715	0.9790	0.9131	0.8459	0.8782
9. UPMC_Team	0.9874	0.9715	0.9794	0.918	0.8372	0.8757
10. unocanda	0.9854	0.9130	0.9478	0.9631	0.7923	0.8694
11. Zarmeen	0.9829	0.9101	0.9451	0.9586	0.7920	0.8674
12. Ranger in R	0.9878	0.9630	0.9753	0.9238	0.8120	0.8643
13. sonam19	0.9865	0.9631	0.9747	0.9224	0.8120	0.8637
14. TelematicUDC	0.9859	0.8935	0.9374	0.9754	0.7699	0.8606
15. DeepDiggers	0.9833	0.9085	0.9444	0.9476	0.7873	0.8600
16. DRL-UNITN-Cognet	0.9901	0.9572	0.9734	0.9280	0.7973	0.8577
17. BaCuDan	0.9713	0.9693	0.9703	0.8600	0.8546	0.8573
18. RushGW	0.9869	0.9596	0.9731	0.9211	0.7967	0.8544
19. netoniq	0.9782	0.9036	0.9394	0.9556	0.7675	0.8513
20. CybElt	0.9827	0.9624	0.9724	0.9121	0.7967	0.8505
21. MaiNTM	0.9783	0.8865	0.9301	0.9387	0.7650	0.8430
22. LSI-UFU	0.9843	0.8945	0.9372	0.9294	0.7572	0.8345
23. ETSISI_UPM	0.9657	0.8858	0.9240	0.9218	0.7582	0.8320
24. RocketScience	0.8185	0.8156	0.8170	0.7205	0.6210	0.6670
25. ICT_UNIFESP	0.3307	0.9614	0.4921	0.3218	0.8607	0.4685

Table 10.7: List of all features including a description as provided by the challenge organizers (Part 1).

Features	Description
cli_pl_header	HTTP client response header size
cli_pl_body	HTTP client response payload size
cli_cont_len	HTTP client declared content length (in the header field)
srv_pl_header	HTTP server response header size
srv_pl_body	HTTP server response payload size
srv_cont_len	HTTP server declared content length (in the header field)
aggregated_sessions	number of requests aggregated into one entry
bytes	Number of bytes transmitted from the client and server comprising the TCP stack header
net_samples	— used internally
tcp_frag	Number of fragmented packets
tcp_pkts	Number of server transmitted packets
tcp_retr	Number of retransmitted packets
tcp_ooo	Number of out of order packets
cli_tcp_pkts	Number of server transmitted packets (Client)
cli_tcp_ooo	Number of out of order packets (Client)
cli_tcp_retr	Number of retransmitted packets (Client)
cli_tcp_frag	Number of fragmented packets (Client)
cli_tcp_empty	How many empty TCP packets have been transmitted (Client)
cli_win_change	How many times the client receive window has been changed
cli_win_zero	How many times the client receive window has been closed
cli_tcp_full	How many packets with full payload have been transmitted (Client)
cli_tcp_tot_bytes	Client TCP total bytes
cli_pl_tot	Client total payload
cli_pl_change	How many times the payload has been changed (Client)

Table 10.8: List of all features including a description as provided by the challenge organizers (Part 2).

Features	Description
srv_tcp_pkts	Number of server transmitted packets (Server)
srv_tcp_ooo	Number of out of order packets (Server)
srv_tcp_retr	Number of retransmitted packets (Server)
srv_tcp_frag	Number of fragmented packets (Server)
srv_tcp_empty	How many empty TCP packets have been transmitted (Server)
srv_win_change	How many times the server receive window has been changed
srv_win_zero	How many times the server receive window has been closed
srv_tcp_full	How many packets with full payload have been transmitted (Server)
srv_tcp_tot_bytes	Server TCP total bytes
srv_pl_tot	Server total payload
srv_pl_change	How many times the payload has been changed (Server)
srv_tcp_win	Last server TCP receive window size
srv_tx_time	Server data transmission time
cli_tcp_win	Last client TCP receive window size
client_latency	Estimated packet delay between client and probe
application_latency	Calculated application response time
cli_tx_time	Client data transmission time
load_time	Roundtrip time since the client request starts up to all server response data are received from client: = application_latency+cli_tx_time+srv_tx_time
server_latency	Estimated packet delay between server and probe
proxy	Flag to identify if it has been used a proxy
sp_healthscore	The healthscore specifies a value between 0 and 10, where 0 represents a low load and a high ability to process requests and 10 represents a high load and that the server is throttling requests to maintain adequate throughput
sp_req_duration	Time elapsed to elaborate the response by the server
sp_is_lat	IS latency
sp_error	If the protocol server rejects the request because the current processing load on the server exceeds its capacity, the protocol server includes a SharePointError header set to 2 in the response. If the protocol server renders an error page to the client for any other reason, the protocol server includes a SharePointError header set to zero in the response
throughput	Bytes/load_time

11

Automatic Frankensteining

Automating machine learning by providing techniques that autonomously find the best algorithm, hyperparameter configuration and preprocessing is helpful for both researchers and practitioners. Therefore, it is not surprising that automated machine learning has become a very interesting field of research. While current research is mainly focusing on finding good pairs of algorithms and hyperparameter configurations, we will present an approach that automates the process of creating a top performing ensemble of several layers, different algorithms and hyperparameter configurations. These kinds of ensembles are called jokingly Frankenstein ensembles and proved their benefit on versatile data sets in many machine learning challenges. We compare our approach Automatic Frankensteining with the current state of the art for automated machine learning on 80 different data sets and can show that it outperforms them on the majority using the same training time. Furthermore, we compare Automatic Frankensteining on a large scale data set to more than 3,500 machine learning expert teams and are able to outperform more than 3,000 of them within 12 CPU hours.

11.1 Introduction

Algorithm selection and hyperparameter optimization is an omnipresent problem for data science practitioners and researchers. Nevertheless, they are usually not concerned about which algorithm or which hyperparameter configuration is selected but they want to have an accurate prediction model. Hence, it is not surprising that many main machine learning players such as Amazon, IBM,

Google and SAS offer commercial tools that at least partially automate this process.

Automating machine learning also attracted a lot of attention of machine learning researchers in the past years. Various studies were able to show that automatic hyperparameter optimization is able to outperform human experts[6,56]. Furthermore, the concept was extended to select preprocessing, algorithm and hyperparameter configurations altogether. The two most famous publicly available tools that offer these features are Auto-WEKA[75] and auto-sklearn[23]. Often, the tools for automating machine learning are extended by methods from meta-learning to transfer knowledge from observed data sets to new ones to initialize the search[81].

The current research focuses strongly on the question which algorithm/hyperparameter configuration combination is best. But in fact the true task that needs to be tackled is to find the strongest prediction model which of course does not need to be estimated from a single algorithm but might be an ensemble of many of them.

While detailed machine learning solutions for real problems created by companies are often confidential, the many competitions hosted by platforms such as Kaggle, DrivenData and CodaLab give great insight how high performance systems for company data can be created. The top performing solutions are often based on ensembles with many layers and various types of preprocessing, hyperparameter configurations and algorithms. These solutions are often very complex and hence this kind of ensembling is jokingly called Frankensteining named after the novel by Mary Shelly[66] in which the scientist Frankenstein creates an ugly artificial life form of many different components. Inspired by these solutions, we try to improve the current state of the art for automated machine learning by proposing a way that will find solutions like these Frankenstein ensembles that make intensive use of diverse algorithms and multi-layer ensembling. Hence, we propose *Automatic Frankensteining*, an automatic approach to generate deep stacked ensembles. While some research already identified that an average ensemble improves the performance[43,23,48], no one has looked into the direction of deep ensembles that consist of several layers. Thus, we are the first to propose a machine learning approach that autonomously identifies a good set of hybrid base learners for a stacked ensemble as well as the right combiner for all base learners. We compare our approach on 80 different classification data sets from the UCI repository to the current state of the art approaches Auto-WEKA and auto-sklearn and show that Automatic Frankensteining is outperforming its competitors on the large majority of data sets using the same CPU time. Finally, we compare our approach to the performance of more than 3,500 human machine learning expert teams on a large scale business data set. Therefore, to the best of our knowledge, we are the first to compare automated machine learning to this very large amount of human experts. Additionally, the solution found by Automatic Frankensteining is outperforming more than 3,000 of the human experts on this task within 12 hours CPU time while each human expert team had 62 days.

11.2 Related Work

Automating machine learning has become a hot topic in the past years. Snoek et al.[68] were among the first to show that Bayesian optimization can be used to automate hyperparameter optimization. Soon afterwards, this idea was extended to the problem of selecting algorithms and preprocessing techniques[75]. Various approaches to make use of meta-knowledge either through initialization[23] or transfer surrogate models[4] have been proposed to accelerate the search for good models. Lately, first tries to make use of ensembling techniques have been proposed. Feurer et al.[23] identified that the many prediction models estimated during Bayesian optimization can be used afterwards to create a weighted ensemble. Lacoste et al.[43] proposes a Bayesian approach. They combine those models that perform best on a randomly selected subset of the validation data. Finally, Levesque[48] proposes to use Bayesian optimization directly to estimate which prediction model is the best candidate to be added to the ensemble. Thornton et al.[75] do not focus on the problem of creating strong ensembles but they make use of more advanced ensembling techniques by considering the structure of the ensemble as further hyperparameters.

In contrast to the state of the art, we propose two innovations. First, we learn strong prediction models per algorithm instead of an overall strong model to obtain strong base learners for the ensemble. Second, we create deep ensembles instead of flat ensembles automatically. Therefore, we use Bayesian optimization to directly estimate strong base learners for ensembling techniques such as stacking instead of finding models that are good in general and create an ensemble as a by-product.

11.3 Background

In the next section we will formalize the problem and describe the basics of automating machine learning and ensembles.

11.3.1 Problem Definition

In supervised machine learning one tries to learn a strong prediction model based on a labeled data set D with n instances and m predictors $X \in \mathbb{R}^{n \times m}$ with corresponding labels $y \in \mathbb{R}^n$ that minimizes a given loss function $\mathcal{L}$ for future data instances. This raises the questions which is the best machine learning algorithm of a set of possible algorithms $A_1, \ldots, A_k$ for a problem where a machine learning algorithm A uses the labeled training data to estimate a prediction model $\hat{y}$. Most machine learning algorithms also have parameters that are not learned during the training phase but need to be set upfront. A typical example is the trade-off parameter C of a linear support vector machine. These parameters are usually called hyperparameters. Since the resulting model is usually sensitive to the

chosen hyperparameter configuration, we actually want to know which algorithm combined with what hyperparameter configuration leads to the model that achieves the smallest loss.

Let us formalize the different parts of machine learning to ease talking about them. A machine learning algorithm A is a function $A : \mathbb{R}^{n \times m} \times \mathcal{Y}^n \times \Lambda \rightarrow \mathcal{M}$ where Λ is the hyperparameter space, $\mathcal{Y}$ the target space and $\mathcal{M}$ the space of all prediction models. The result of this function is a prediction model $\hat{y} : \mathbb{R}^m \rightarrow \mathcal{Y}$ that allows to make predictions for new instances. The target space $\mathcal{Y}$ depends on the problem. For regression problems $\mathcal{Y} = \mathbb{R}^n$ and for classification problems $\mathcal{Y} = \mathbb{R}^{n \times c}$ for a set of classes $\mathcal{C} = \{1, \ldots, c\}$. For the groundtruth $\mathrm{Y} \in \mathcal{Y}$ of a classification problem usually yields that $y_{i,j} = 1$ if instance i belongs to class j and $y_{i,j} = 0$ otherwise.

We will focus on classification problems as a specific problem instance in this chapter. Typical classification losses are the classification error

$$\mathcal{L}\left(\mathrm{Y}, \hat{\mathrm{Y}}\right) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\left(\arg\max_{j \in \mathcal{C}} y_{i,j} \neq \arg\max_{j \in \mathcal{C}} \hat{y}_{i,j}\right) \tag{11.1}$$

and the logistic loss

$$\mathcal{L}\left(\mathrm{Y}, \hat{\mathrm{Y}}\right) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{c} y_{i,j} \log\left(\hat{y}_{i,j}\right) \quad , \tag{11.2}$$

where Y is the groundtruth and $\hat{\mathrm{Y}}$ the prediction.

The hyperparameter space Λ depends on the algorithm A. In the aforementioned case of a linear support vector machine with the only hyperparameter being the trade-off C, the hyperparameter space would equal $\mathbb{R}^+$, however, hyperparameter spaces are often multi-dimensional.

11.3.2 Ensembling Techniques

An ensemble of classifiers is a combination of classifiers $h_1, \ldots, h_e$. The simplest way of combining them is by *voting* i.e. that each classifier has an equal vote and the majority decides which class is predicted. If the probability score is predicted, one can combine the prediction by averaging the score over all classifiers.

$$\hat{y}(x) = \frac{1}{e} \sum_{i=1}^{e} h_i(x) \tag{11.3}$$

Instead of using an unweighted voting/averaging, one can also assign weights to the classifiers. This can be done manually or also learned on a hold-out data set[10]. Important for an ensemble is that its members are accurate by itself (better than random) and diverse which means they are stronger than the other members on specific parts of the data[30]. A more advanced technique to combine classifiers is *stacking*. In the case of stacking, the data set needs to be split into disjoint sets D_1 and

D_2. The ensemble members will be trained on D_1 and predictions are estimated for all $(x, y) \in D_2$ and vice versa. After obtaining the meta-instances

$$\left(\begin{pmatrix} h_1(x) & \cdots & h_e(x) \end{pmatrix}^{\mathrm{T}}, y \right) \quad , \tag{11.4}$$

a combiner algorithm is learned on these meta-instances which can be any classifier or even another ensemble.

11.4 Automatic Frankensteining

In the Automatic Frankensteining framework we distinguish between two different components. The *model selection component* is responsible for training models and finding near-optimal hyperparameter configurations. The *ensembling component* combines the models trained in the preceding model selection component to further boost the prediction performance and reduce the input space for the subsequent model selection component.

11.4.1 Model Selection Component

The task of the model selection component is to create useful meta-features or to provide the final predictions based on some given input. On an abstract level, the model selection component can be considered as a function $g : \mathcal{X} \to \mathcal{Y}^*$ that converts a data set representation X_2 to an intermediate representation $g(X_2)$. Therefore, it needs to be trained previously on a data set D_1 with predictors X_1 and corresponding labels y_1. This conversion is done by training machine learning algorithms $A_1, \ldots, A_k$ on D_1 that create models $\hat{y} : \mathcal{X} \to \mathcal{Y}$. Using these models to create predictions $\left(\hat{y}_1(X_2), \ldots, \hat{y}_l(X_2)\right)^T$ we obtain a new representation.

Example 11.4.1. *Assume that only one model is used to generate a new representation for some predictors X_2. Then one can e.g. use a decision tree A_1 with a chosen hyperparameter configuration $\boldsymbol{\lambda}_1$ and train it on the data set D_1. A prediction model $\hat{y}$ is obtained which can be used to make predictions for X_2, i.e. $\hat{y}(X_2)$. These predictions are used as the new representation for X_2.*

As previously explained in Section 11.3.2, good meta-features are created by a diverse set of machine learning algorithms. Here, diversity rather than prediction performance is important. Nevertheless, very weak or useless prediction models will also lead to bad meta-features. Hence, it is vital to select appropriate hyperparameter configurations for the machine learning algorithms. The model selection component will solve both issues. Having access to k machine learning algorithms $A_1, \ldots, A_k$, it will automatically search for good hyperparameter configurations for each of them.

Algorithm 10 Training the Model Selection Component

Input: Hyperparameter spaces $\Lambda_{1,\ldots,k}$, observation histories $\mathcal{H}_{1,\ldots,k}$, surrogate models $\Psi_{1,\ldots,k}$, acquisition function a, time limit t, train and validation data sets $D_{\text{train}} = (X_{\text{train}}, y_{\text{train}}), D_{\text{valid}} = (X_{\text{valid}}, y_{\text{valid}})$.

Output: Meta-features $\hat{\text{Y}}$ for D_{valid}, loss L for column-wise predictions on D_{valid}.

$\text{L} \leftarrow (), \hat{\text{Y}} \leftarrow ()$
while time limit t is not reached **do**
  Choose $j \in \{1, \ldots k\}$ proportional to $\left|\Lambda_j\right|$ at random.
  Fit Ψ_j to $\mathcal{H}_j$
  $\boldsymbol{\lambda} \leftarrow \arg\max_{\boldsymbol{\lambda} \in \Lambda_j} a\left(\boldsymbol{\lambda}, \Psi_j, f_j^{\min}\right)$
  if predicted run-time for $A_j(X_{\text{train}}, y_{\text{train}}, \boldsymbol{\lambda})$ does not exceed the remaining run-time **then**
    $\hat{y} \leftarrow A_j(X_{\text{train}}, y_{\text{train}}, \boldsymbol{\lambda})$
    $f_j(\boldsymbol{\lambda}) \leftarrow \mathcal{L}(y_{\text{valid}}, \hat{y}(X_{\text{valid}}))$
    $\mathcal{H}_j \leftarrow \mathcal{H}_j \cup \left\{\left(\boldsymbol{\lambda}, f_j(\boldsymbol{\lambda})\right)\right\}$
    $\hat{\text{Y}} \leftarrow \left(\begin{array}{cc} \hat{\text{Y}} & \hat{y}(X_{\text{valid}}) \end{array}\right)$
    $\text{L} \leftarrow \left(\begin{array}{cc} \text{L} & f_j(\boldsymbol{\lambda}) \end{array}\right)$
    if $f_j(\boldsymbol{\lambda}) < f_j^{\min}$ **then**
      $f_j^{\min} \leftarrow f_j(\boldsymbol{\lambda})$
    end if
  end if
end while
return $\hat{\text{Y}}$, L

In contrast to the state of the art for automatic model selection and hyperparameter optimization, which is interested in finding *the* best model with *the* best hyperparameter configuration, we are interested in finding predictive meta-features. Thus, we need to find for each algorithm a hyperparameter configuration that leads to a good prediction model itself. Hence, the hyperparameter search differs from the current state of the art approaches. While the state of the art is applying a global hyperparameter optimization over the whole algorithm and hyperparameter space, we apply a hyperparameter optimization per algorithm. Because the hyperparameter dimensionality between algorithms might differ a lot, as a logistic regression has only one to two hyperparameters while a neural network might have a dozen, each of the k individual hyperparameter searches is continued with probability proportional to the number of hyperparameters. For the automatic hyperparameter search, we make use of Bayesian optimization explained in Section 2.3.

Algorithm 10 summarizes the training procedure of the model selection component. Given a set of k learning algorithms $A_1, \ldots, A_k$, a training and validation data set D_{train} and D_{valid}, k-many hyperparameter searches are executed in parallel for a given time limit t. Whenever a CPU finished its task, a new task for it is estimated as follows. An algorithm A_j is selected proportional to the dimensionality of the hyperparameter search. Following Bayesian optimization, the surrogate model Ψ_j is updated and the hyperparameter configuration for the chosen algorithm A_j is selected that maximizes the acquisition function. If the predicted run-time of evaluating A_j with the selected hyperparameter configuration $\boldsymbol{\lambda}$ exceeds the remaining run-time, we continue with another algorithm. For predicting the run-time we use the approach proposed by Hutter et al.[36]. We run our k algorithms on different data sets of different sizes. Then, we use the number of instances, the number of predictors and the hyperparameter configuration to predict the run-time using a random forest. This will ensure that training the model selection component will finish within the specified time frame. In the case that no further algorithm can finish in the remaining time, we stop training the model selection component early.

After training the model selection component, it will provide predictions for D_{valid} for each model learned and its corresponding loss. This output will be used for training the following ensembling component. Furthermore, the model selection component now represents a function $g : \mathbb{R}^{n \times p} \to \mathbb{R}^{n \times q}$ that can map some predictors $X \in \mathbb{R}^{n \times p}$ to the meta-feature space. Therefore, the q prediction models trained during the training phase of the model selection component are used to make predictions

$$\hat{\mathrm{Y}} = \begin{pmatrix} \hat{y}_1(X) & \cdots & \hat{y}_q(X) \end{pmatrix} \tag{11.5}$$

which act as our meta-features.

11.4.2 Ensembling Component

Before using the output of the model selection component as final predictions or meta-features, the dimensionality needs to be reduced. One option is to use the validation performance to select only the predictions of the best model or the best model of each algorithm. This has the advantage of using strong prediction models from diverse algorithms but the disadvantage that many estimated models are not considered at all. Another option is to average the predictions or average them by algorithm. This usually leads to a lift in the prediction but will not work in our case. Here, we face the problem that we learned some models with bad hyperparameter configurations that led to very bad or even constant models that will deteriorate the overall prediction. Therefore, we propose to

Algorithm 11 Bagged Ensemble

Input: Number of bags b, fraction of models in a bag r, number of combined models in a bag s, groundtruth y_{valid}, predictions $\hat{\mathrm{Y}} \in \mathbb{R}^{n \times q}$.
Output: Prediction of the ensemble.

for $i \leftarrow 1$ to b **do**
  $Q \subset \{1, \ldots, q\}$ s.t. $|Q| = rq$
  $\hat{y}_i \leftarrow 0 \in \mathbb{R}^n$
  for $j \leftarrow 1$ to s **do**
    $m_{\text{best}} \leftarrow 0,\ l_{\text{best}} \leftarrow \infty$
    for $m \in Q$ **do**
      $l \leftarrow \mathcal{L}\left(y_{\text{valid}}, \frac{1}{j}\left(\hat{y}_i + \hat{\mathrm{Y}}_{\cdot,m}\right)\right)$
      if $l < l_{\text{best}}$ **then**
        $m_{\text{best}} \leftarrow j,\ l_{\text{best}} \leftarrow l$
      end if
    end for
    $\hat{y}_i \leftarrow \hat{y}_i + \hat{\mathrm{Y}}_{\cdot,m_{\text{best}}}$
  end for
end for
return $\frac{1}{b \cdot s} \sum_{i=1}^{b} \hat{y}_i$

use a weighted ensemble to combine different models, i.e. the columns of $\hat{\mathrm{Y}}$. We employ the bagged ensembling technique with replacement by Caruana et al.[10] to create b bags. Each bag chooses at random with probability r whether to consider a model or not. Then, s models are greedily selected and combined. The final prediction is the average prediction of all bags. This bagged approach will avoid overfitting on the meta-level. This way of ensembling might look complicated but in fact it is nothing but a weighted average of all models. The weight of the model is given by how often it was chosen. Algorithm 11 summarizes the component. During the training phase the weights are

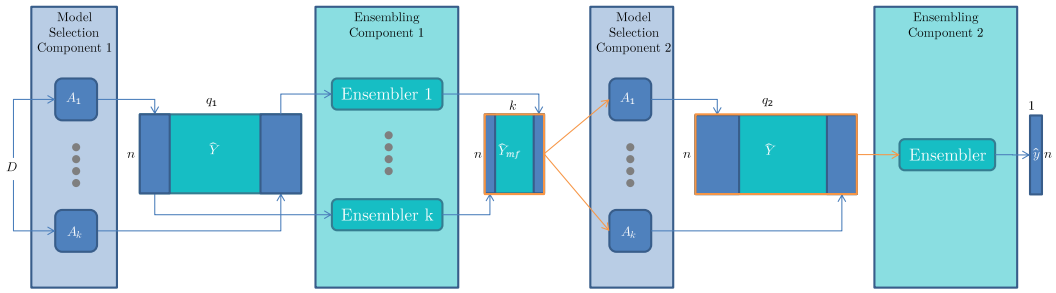

Figure 11.1: The final framework consists of two main layers. The first layer learns models on the original data. The estimated models are then ensembled by algorithm family. The resulting predictions lead to our meta-features that are used in the second main layer. Again, models are trained, this time on the meta-features. Finally, all models are ensembled to a single prediction vector.

estimated on some validation set. For prediction, these weights are used to combine the predictions of the model selection component. Since most weights are zero, many models can be discarded after the training procedure of the ensembling component has finished. This saves memory and reduces the prediction time.

11.4.3 Final Framework

In this section we will describe how the components described in the previous sections are combined to the final framework. Furthermore, we will describe on what part of the data the specific components have been trained and evaluated.

Figure 11.1 sketches our framework. While it looks complicated, it is the most simple version of our proposed method of Automatic Frankensteining. In fact, it is very similar to a stacked ensemble.

First, the given training data is split into two disjoint parts D_{train} and D_{blend} where D_{blend} contains 10% of the data. D_{train} is used in the first model selection component. The component is evaluating algorithm/hyperparameter configurations in a three-fold cross-validation. Thus, we are learning for each algorithm/hyperparameter configuration tuple three models, one for each fold. The following ensembling component also evaluates the weighted ensembles with a three-fold cross-validation with $b = 1, s = 30$ and $r = 1$. At this point, we can generate meta-features for D_{train} by using out-of-fold-predictions such that label information is not leaked. Furthermore, we can generate meta-features for arbitrary data by averaging the predictions of each of the three prediction models learned on a fold. We avoid retraining a model on the complete data to save time. Now, the training for the first two components is completed and they will not be updated any more.

Then, the last two components are trained. Using the first two components, we generate the meta-features for D_{blend} that is used as the training data for the last two components. D_{train} will be

used at this point for evaluating the performance of the models. Actually, we trained the second model selection component twice. Once, only on the meta-features and once on the meta-features plus the original features. This allows to use possible interactions between features and meta-features. All models are finally combined with a big weighted ensemble with $b = 10$, $s = 30$ and $r = 0.5$.

Now, the automatically created ensemble can be used to predict for a new test data set D_{test}. The first model selection component computes the meta-features that are then processed by the ensemble component. These meta-features now are used for the second model selection component as features and their output is combined again by another ensembling component to the final prediction vector. For this step only predictions are done or combined.

We restrict us to following classifiers: naive Bayes (Gaussian and Bernoulli), logistic regression, support vector machine with radial basis kernel, k-nearest neighbors, extra randomized trees, gradient boosting classifier[54] and gradient boosting machine[15].

The first half of the available training time is used to train the first two components, the second half for the last two components.

11.5 Experimental Section

Our experimental section is divided into two parts. In the first part we focus on comparing our approach to the current state of the art for automatic machine learning. In the second part we focus on comparing against human machine learning experts.

11.5.1 Comparison to Other Approaches

We will compare our approach against Auto-WEKA[75] and auto-sklearn[23] using the authors' implementation and recommended settings.

Auto-WEKA is an addition to WEKA[29] which makes use of the different preprocessing techniques and algorithms offered by WEKA. Since stacking and other ensembles are also algorithms in WEKA, Auto-WEKA can make use of ensembles as well. In contrast to our approach, it is not steered to create an ensemble and thus it can happen that the final solution found by Auto-WEKA is not an ensemble. At its core, Auto-WEKA's search is controlled by Bayesian optimization.

auto-sklearn uses Bayesian optimization to find good algorithms, hyperparameter configurations and preprocessing techniques provided by scikit-learn[54]. After finishing the search, auto-sklearn uses a weighted ensemble[10] to combine estimated prediction models. It has also the feature

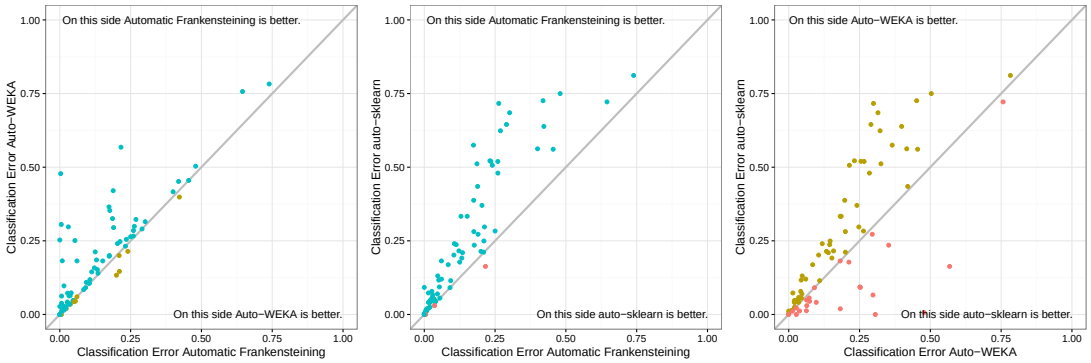

Figure 11.2: Our approach Automatic Frankensteining is only beaten on 11 data sets by Auto-WEKA and only 3 by auto-sklearn and hence provides the better solution for the majority of data sets.

to use meta-knowledge to initialize[23] the search i.e. that it first evaluates those algorithm/hyperparameter configuration pairs that have been good on many other data sets. We disabled this feature to avoid a distortion of the results for two reasons. First, Auto-WEKA does not contain this feature and second, we perform our evaluation on UCI data sets which are likely used to estimate the initialization sequence and thus auto-sklearn would make use of test information.

We chose 80 different classification data sets from the UCI repository. If these data sets already provided a train/test split, we merged them. We shuffled the instances and split them into 80% training data and 20% test data that is used for evaluation purposes only. Our evaluation measure is the classification error for which all methods are optimized directly. Each method got 30 minutes time to create the best predictions.

Figure 11.2 shows the classification error on the test split where each point represents one data set. The classification error for each approach can be read from the axis. Points on the diagonal indicate ties, points on the one or the other side indicate wins for the named method. Our approach Automatic Frankensteining finds in 55 out of 80 cases a better prediction model than Auto-WEKA and has 14 ties. In comparison to auto-sklearn the difference is even stronger. Automatic Frankensteining finds for 74 data sets a stronger model and only in 3 cases auto-sklearn provides the better solution. The average rank of Automatic Frankensteining is 1.28, for Auto-WEKA 2.06 and for auto-sklearn 2.66. Comparing Auto-WEKA and auto-sklearn head to head, Auto-WEKA finds on average the better prediction models. It finds in 56 cases the better solution, only in 21 cases auto-sklearn is the better option

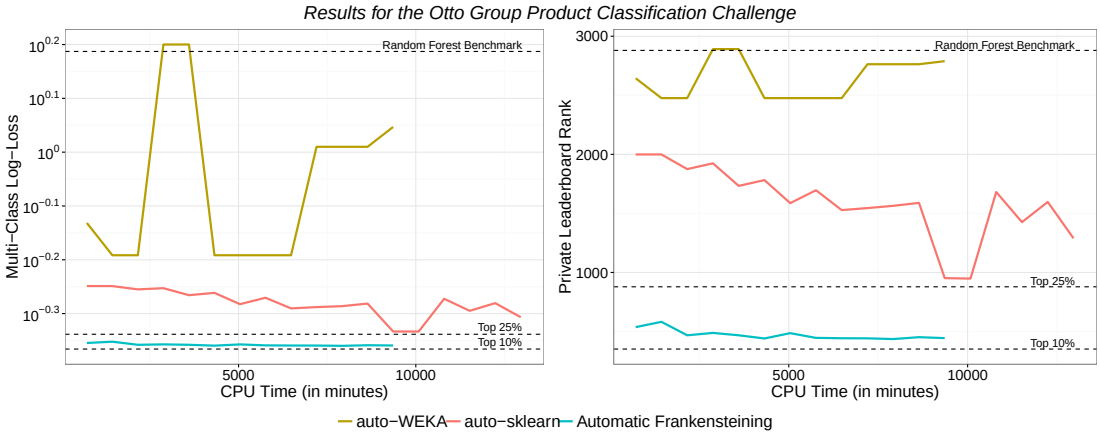

Figure 11.3: Automatic Frankensteining (bottom) achieves within hours already a very small loss on the private leaderboard, outperforming the automated machine learning baselines WEKA (top) and auto-sklearn (middle) as well as the majority of human participants. The dotted lines indicate the performane of the Random Forest Benchmark, the top 25% and top 10% of the participants.

11.5.2 Comparison to Human Experts

To compare our approach against a wide range of machine learning experts, we applied Automatic Frankensteining on the Otto Group Product Classification Challenge*, one of the most popular Kaggle challenges. In this challenge, the Otto Group, one of the world's biggest e-commerce companies, asked participants to decide to which of their main categories a product described by 93 features belongs. The challenge data contains business data from more than 20 countries of the Otto Group for more than 200,000 products. The performance was measured based on the multi-class logistic loss as defined in Equation (11.2). More than 3,500 teams participated in this challenge and tried for 62 days to be among the top three to claim their share of the $10,000 price money. The organizers provided a Random Forest Benchmark baseline which was able to achieve a score of 1.5387 on the private leaderboard, the best solution achieved a score of 0.38243.

The labels for the leaderboards were never made publicly available but it is still possible to submit your predictions and let Kaggle evaluate it for you. This means the following for our experiments. First, we are limited by the number of submissions and evalutions because they involve time-consuming interactions with the Kaggle interface. Hence, we were restricted to just few evaluations. Second, since the label is still hidden, it is impossible that our results are distorted by any means because the split and evaluation are maintained by a third party. This also guarantees a high degree of reproducability.

The main focus in this experiment is the comparison of Automatic Frankensteining to human

*https://www.kaggle.com/c/otto-group-product-classification-challenge

experts. Nevertheless, it is also interesting to see how the current state of the art for automated machine learning performs. Unfortunately, Auto-WEKA only allows to optimize for the classification error and not for arbitrary metrics. We still report the results of Auto-WEKA but this is the reason why we see rather bad and strange results by Auto-WEKA. Automatic Frankensteining and auto-sklearn directly optimize for the right loss. We decided to give each approach 12, 24, 36, ..., 156 hours and submit the predictions after this time to the Kaggle platform. Since we saw a dramatic improvement by auto-sklearn after 156 hours, we also investigated whether auto-sklearn can improve even more if more time is provided. Figure 11.3 presents the results with respect to the multi-class log-loss and the rank on the private leaderboard for the predictions estimated after the given CPU time. As explained before, Auto-WEKA shows some strange behaviour. But since the log-loss correlates with the classification error, Auto-WEKA is able to beat at least the baseline provided by the challenge organizers. Automatic Frankensteining achieves already after 12 hours a very good result and can improve if more computational run-time is provided. After running for 2 days on the data set, Automatic Frankensteining has converged and does not show any further improvement. It is not able to reach the top 10% of the participants and reached a score just under rank 400. Yet, this is quite an impressive result considering that the better performing human experts likely had to spend much more time to achieve the same result. Furthermore, Automatic Frankensteining is able to demonstrate for a further data set that it delivers the better predictions than auto-sklearn.

We also investigated the performance of the individual components. Each single model created by Automatic Frankensteining provided worse predictions than the final ensemble. So we can confirm the results found by the human experts during this challenge, that only a Frankenstein ensemble can provide the most powerful predictions.

11.6 Conclusions

We proposed Automatic Frankensteining, an automatic way of learning ensembles with many different algorithms with several layers. In a comparison on 80 different data sets, Automatic Frankensteining was able to outperform the current state of the art for automatic machine learning for the large majority of the data sets. Furthermore, in an additional experiment we compared Automatic Frankensteining on a large scale business data set with more than 3,500 human machine learning expert teams and were able to achieve a better score than more than 3,000 teams within just 12 hours CPU time. In the future we will further improve our method to finally reach the level of very strong experts. Therefore, we have to focus strongly on components such as automatic feature engineering which currently give human experts a not negligible advantage over our approach.

Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.

Andrew Ng

12 Conclusion

Here we will only conclude the third part of the thesis, for a conclusion on the second part please see Chapter 9.

In this part of the thesis we evaluated Bayesian optimization in comparison to human experts on larger data sets. We investigated how good fully automated machine learning performs compared to human experts. We saw very good results but it still does not achieve the level of top machine learning experts. Nevertheless, it is already now a very useful tool for machine learning experts to ease their daily life for optimizing hyperparameters or get a quick prototype.

Part III of this thesis only focuses on selecting algorithms and hyperparameter configurations based on observations on a single data set. This is in strong contrast to Part II which focuses on transferring knowledge observations from other data sets to a new one. Obviously, it would be very interesting to apply the methods of Part II in particular to the automatic ensemble creation presented in Chapter 11.

I did not apply transfer learning in Chapter 10 because I strongly focused on the challenge and no meta-knowledge for the used prediction model was available. It did not seem beneficial to spend resources in creating required meta-knowledge and instead I spend all resources directly on the target data set.

In Chapter 11 our main focus was on proposing a novel method for automatic ensemble creation. However, applying transfer learning is straight forward for the first model selection component. I am convinced that this will lead to similar results in less time as it was demonstrated multiple times in Part II of this thesis. An interesting research question is whether full ensemble architectures can

be transferred from one problem to the other. However, this is a more complicated question and out-of-scope.

Interest in automating machine learning is growing and I am sure we will see major breakthroughs in the coming years. There are a couple of things that are currently very difficult to accomplish automatically. Feature engineering is one of these things. Feature engineering lacks a formal definition but is essential for applied machine learning. I define feature engineering as the task of creating or extending the instance representation by using creativity, common knowledge or domain expertise. Related to this task are preprocessing and representation learning[5] which can be both automated. Preprocessing involves techniques such as scaling or extracting e.g. TF-IDF score from text data. Representation learning is the automatic transformation of features from the raw data. One can distinguish two categories, supervised and unsupervised representation learning. Typical examples for supervised representation learning are neural networks, examples for unsupervised representation learning are clustering, matrix factorization or auto-encoders.

To give an example for feature engineering, let us remind the features we engineered in Chapter 10 for the ECML-PKDD Discovery Challenge 2016 on Bank Card Usage Analysis. We said that customers living close to each other will likely visit the same bank branches and thus, added the nearest-neighbor predictions with respect to residence distance as additional features. This is an excellent example that shows that coming up with this feature is not really difficult for a human being. Otherwise, this is almost an impossible task for the machine and a time-consuming task for the human being. Even though this feature can be described and implemented in hindsight easily, more work, code and time is needed to proof that a feature is actually useful for the problem. Furthermore, it may even happen that adding features, which are useful at its own, do not enrich the representation in combination with other features because their information is already captured by the other features. Few work exists that try to engineer features automatically[3] but the big breakthrough is not achieved yet.

I consider automatic feature engineering as one of the remaining big challenges of automated machine learning. Additionally, I think it is also the topic that is most helpful in improving the solutions provided by fully automated machine learning.

Another interesting aspect is the automated creation of neural network architectures. Due to the high interest in Deep Learning, this is likely to become a hot topic in the upcoming years.

A

Look-Up Tables for SMFO

Look-up tables for hyperparameter optimization generated by Average Surrogate Model-free Optimization as described in Chapter 4. Table A.1 to A.3 contain the look-up table for Adaboost, Table A.4 to A.12 contain the look-up table for libSVM.

Table A.1: Best ranking of hyperparameters for multiboost according to A-SMFO introduced in Chapter 4 (Part 1).

Rank	Number of Iterations	Number of Product Terms
1	10000	4
2	100	2
3	10000	10
4	1000	3
5	10	2
6	5000	20
7	2000	2
8	500	4
9	1000	5
10	50	2
11	1000	15
12	10000	15
13	5000	7
14	2000	3
15	2	2
16	10000	5
17	10000	7
18	200	4
19	200	2
20	10000	3
21	2000	5
22	50	3
23	100	20
24	200	3
25	50	4
26	10000	20
27	2000	10
28	5000	4
29	20	3
30	100	3
31	1000	2
32	1000	7
33	1000	20
34	500	15
35	10	3
36	20	4

Table A.2: Best ranking of hyperparameters for multiboost according to A-SMFO introduced in Chapter 4 (Part 2).

Rank	Number of Iterations	Number of Product Terms
37	5000	2
38	5000	15
39	5000	5
40	100	7
41	200	5
42	500	10
43	2000	15
44	100	5
45	5000	3
46	1000	4
47	2	4
48	1000	30
49	20	15
50	50	20
51	5	3
52	20	2
53	5	2
54	10000	2
55	2000	7
56	500	2
57	200	7
58	2	3
59	10000	30
60	10	15
61	2000	4
62	50	30
63	10	7
64	2000	20
65	5	4
66	50	7
67	20	5
68	100	4
69	500	5
70	10	5
71	2	30
72	5000	10

Table A.3: Best ranking of hyperparameters for multiboost according to A-SMFO introduced in Chapter 4 (Part 3).

Rank	Number of Iterations	Number of Product Terms
73	100	15
74	500	3
75	500	7
76	200	10
77	20	10
78	1000	10
79	2000	30
80	200	20
81	2	20
82	200	15
83	10	4
84	5	7
85	2	7
86	50	5
87	5000	30
88	200	30
89	2	15
90	5	5
91	2	10
92	100	30
93	2	5
94	100	10
95	5	15
96	500	30
97	50	10
98	5	30
99	20	7
100	5	20
101	500	20
102	10	20
103	50	15
104	5	10
105	10	30
106	10	10
107	20	20
108	20	30

Table A.4: Best ranking of hyperparameters for libSVM according to A-SMFO introduced in Chapter 4 (Part 1).

Rank	Kernel	C	Degree	Gamma
1	RBF	8	-	0.1
2	RBF	16	-	5
3	RBF	8	-	0.01
4	RBF	16	-	0.5
5	Linear	4	-	-
6	RBF	64	-	10
7	Polynomial	8	6	-
8	RBF	16	-	100
9	Polynomial	32	2	-
10	RBF	64	-	1
11	Polynomial	16	9	-
12	RBF	4	-	0.05
13	Polynomial	2	4	-
14	RBF	4	-	2
15	RBF	0.5	-	0.1
16	RBF	64	-	0.5
17	RBF	4	-	5
18	Linear	0.5	-	-
19	RBF	16	-	0.05
20	RBF	16	-	0.1
21	Linear	1	-	-
22	Polynomial	64	7	-
23	RBF	64	-	2
24	Polynomial	32	5	-
25	Polynomial	4	9	-
26	RBF	0.03125	-	1000
27	RBF	2	-	1
28	RBF	64	-	5
29	Polynomial	4	2	-
30	RBF	32	-	0.0001
31	RBF	1	-	100
32	RBF	32	-	0.5
33	RBF	4	-	0.5
34	Linear	0.03125	-	-
35	RBF	2	-	0.5

Table A.5: Best ranking of hyperparameters for libSVM according to A-SMFO introduced in Chapter 4 (Part 2).

Rank	Kernel	C	Degree	Gamma
36	RBF	8	-	100
37	RBF	64	-	0.1
38	RBF	8	-	5
39	RBF	0.5	-	0.01
40	Polynomial	64	9	-
41	Polynomial	1	2	-
42	RBF	16	-	20
43	Polynomial	64	2	-
44	RBF	1	-	10
45	Polynomial	16	10	-
46	Linear	32	-	-
47	Polynomial	4	4	-
48	RBF	16	-	1
49	RBF	64	-	50
50	Linear	8	-	-
51	Polynomial	16	7	-
52	Polynomial	4	5	-
53	RBF	8	-	0.5
54	RBF	16	-	2
55	Polynomial	4	8	-
56	RBF	64	-	0.01
57	RBF	0.5	-	0.5
58	RBF	0.5	-	1000
59	Linear	16	-	-
60	RBF	64	-	0.05
61	RBF	0.0625	-	0.05
62	RBF	32	-	5
63	Polynomial	32	6	-
64	RBF	1	-	0.05
65	RBF	64	-	20
66	Linear	0.125	-	-
67	RBF	4	-	0.01
68	RBF	2	-	100
69	RBF	4	-	0.1
70	RBF	0.5	-	0.05

Table A.6: Best ranking of hyperparameters for libSVM according to A-SMFO introduced in Chapter 4 (Part 3).

Rank	Kernel	C	Degree	Gamma
71	RBF	64	-	0.0001
72	RBF	8	-	2
73	RBF	32	-	100
74	RBF	16	-	0.01
75	RBF	2	-	5
76	RBF	32	-	1
77	Linear	2	-	-
78	Polynomial	16	4	-
79	Polynomial	16	5	-
80	RBF	32	-	0.1
81	RBF	0.125	-	100
82	RBF	2	-	0.05
83	Polynomial	4	3	-
84	RBF	2	-	2
85	RBF	32	-	20
86	RBF	32	-	0.05
87	Polynomial	32	10	-
88	RBF	0.25	-	0.01
89	RBF	1	-	1
90	RBF	32	-	2
91	Polynomial	16	2	-
92	RBF	0.03125	-	5
93	RBF	32	-	10
94	RBF	2	-	10
95	RBF	16	-	0.001
96	Polynomial	64	8	-
97	RBF	1	-	0.1
98	RBF	8	-	10
99	Linear	0.0625	-	-
100	RBF	4	-	20
101	RBF	4	-	50
102	RBF	0.25	-	1000
103	Polynomial	64	5	-
104	RBF	8	-	1
105	RBF	4	-	100

Table A.7: Best ranking of hyperparameters for libSVM according to A-SMFO introduced in Chapter 4 (Part 4).

Rank	Kernel	C	Degree	Gamma
106	Polynomial	4	10	-
107	RBF	8	-	0.05
108	RBF	32	-	0.01
109	Polynomial	64	3	-
110	Polynomial	64	10	-
111	RBF	0.03125	-	0.1
112	RBF	1	-	0.5
113	RBF	32	-	0.001
114	RBF	16	-	10
115	RBF	64	-	100
116	RBF	8	-	20
117	Polynomial	4	7	-
118	Linear	0.25	-	-
119	Polynomial	32	3	-
120	Polynomial	2	5	-
121	Polynomial	2	3	-
122	RBF	1	-	2
123	RBF	0.5	-	100
124	RBF	8	-	50
125	RBF	1	-	5
126	Polynomial	16	8	-
127	Linear	64	-	-
128	RBF	4	-	1
129	Polynomial	1	3	-
130	Polynomial	8	4	-
131	Polynomial	8	3	-
132	Polynomial	32	9	-
133	Polynomial	8	5	-
134	RBF	2	-	0.1
135	RBF	0.5	-	50
136	Polynomial	2	10	-
137	Polynomial	64	6	-
138	Polynomial	8	2	-
139	RBF	2	-	0.01
140	RBF	0.25	-	0.5

Table A.8: Best ranking of hyperparameters for libSVM according to A-SMFO introduced in Chapter 4 (Part 5).

Rank	Kernel	C	Degree	Gamma
141	RBF	0.25	-	2
142	RBF	32	-	50
143	RBF	0.0625	-	10
144	RBF	0.0625	-	50
145	Polynomial	2	9	-
146	RBF	4	-	10
147	Polynomial	8	7	-
148	RBF	4	-	0.001
149	Polynomial	1	5	-
150	Polynomial	32	4	-
151	RBF	0.5	-	10
152	RBF	2	-	50
153	RBF	16	-	50
154	Polynomial	8	10	-
155	Polynomial	32	8	-
156	Polynomial	64	4	-
157	RBF	2	-	0.001
158	Polynomial	1	4	-
159	Polynomial	16	3	-
160	RBF	1	-	1000
161	RBF	1	-	0.01
162	Polynomial	1	7	-
163	RBF	1	-	0.001
164	Polynomial	4	6	-
165	RBF	0.5	-	20
166	RBF	0.125	-	50
167	RBF	2	-	20
168	Polynomial	2	8	-
169	RBF	1	-	50
170	RBF	8	-	0.001
171	Polynomial	1	8	-
172	Polynomial	16	6	-
173	RBF	0.125	-	10
174	Polynomial	8	8	-
175	RBF	0.25	-	50

Table A.9: Best ranking of hyperparameters for libSVM according to A-SMFO introduced in Chapter 4 (Part 6).

Rank	Kernel	C	Degree	Gamma
176	RBF	1	-	20
177	Polynomial	2	2	-
178	Polynomial	1	9	-
179	RBF	0.0625	-	1
180	RBF	16	-	0.0001
181	RBF	0.5	-	5
182	RBF	0.125	-	1000
183	Polynomial	0.5	2	-
184	Polynomial	32	7	-
185	Polynomial	8	9	-
186	Polynomial	1	10	-
187	RBF	0.125	-	5
188	RBF	8	-	0.0001
189	RBF	0.5	-	2
190	RBF	0.0625	-	1000
191	Polynomial	1	6	-
192	RBF	0.03125	-	0.5
193	RBF	64	-	0.001
194	RBF	2	-	1000
195	RBF	0.25	-	20
196	RBF	0.125	-	0.01
197	RBF	4	-	0.0001
198	RBF	0.0625	-	5
199	Polynomial	2	6	-
200	RBF	32	-	1000
201	Polynomial	2	7	-
202	RBF	0.0625	-	2
203	RBF	0.03125	-	10
204	RBF	64	-	1000
205	Polynomial	0.0625	4	-
206	RBF	0.125	-	0.5
207	RBF	8	-	1000
208	RBF	16	-	1000
209	RBF	0.25	-	0.1
210	RBF	0.25	-	100

Table A.10: Best ranking of hyperparameters for libSVM according to A-SMFO introduced in Chapter 4 (Part 7).

Rank	Kernel	C	Degree	Gamma
211	RBF	4	-	1000
212	RBF	0.125	-	2
213	RBF	0.25	-	0.05
214	RBF	0.5	-	1
215	RBF	0.125	-	0.1
216	RBF	0.25	-	1
217	RBF	0.125	-	1
218	RBF	2	-	0.0001
219	RBF	0.03125	-	1
220	RBF	0.0625	-	100
221	RBF	0.25	-	10
222	RBF	0.0625	-	0.5
223	Polynomial	0.5	6	-
224	RBF	0.03125	-	100
225	RBF	0.0625	-	0.1
226	Polynomial	0.5	3	-
227	Polynomial	0.125	4	-
228	Polynomial	0.5	9	-
229	RBF	0.25	-	5
230	Polynomial	0.25	2	-
231	Polynomial	0.5	10	-
232	Polynomial	0.5	4	-
233	RBF	0.125	-	0.05
234	RBF	0.5	-	0.001
235	RBF	0.0625	-	0.01
236	RBF	0.03125	-	50
237	Polynomial	0.5	8	-
238	Polynomial	0.25	3	-
239	Polynomial	0.5	5	-
240	RBF	0.03125	-	0.05
241	Polynomial	0.5	7	-
242	RBF	0.125	-	20
243	RBF	0.03125	-	2
244	RBF	0.0625	-	20
245	Polynomial	0.125	2	-

Table A.11: Best ranking of hyperparameters for libSVM according to A-SMFO introduced in Chapter 4 (Part 8).

Rank	Kernel	C	Degree	Gamma
246	RBF	0.25	-	0.001
247	Polynomial	0.0625	2	-
248	RBF	0.03125	-	0.01
249	Polynomial	0.25	4	-
250	RBF	0.03125	-	20
251	Polynomial	0.125	3	-
252	Polynomial	0.03125	2	-
253	Polynomial	0.25	10	-
254	Polynomial	0.0625	3	-
255	Polynomial	0.25	8	-
256	Polynomial	0.25	9	-
257	Polynomial	0.25	7	-
258	Polynomial	0.03125	3	-
259	Polynomial	0.25	6	-
260	Polynomial	0.25	5	-
261	RBF	1	-	0.0001
262	RBF	0.125	-	0.001
263	Polynomial	0.125	6	-
264	Polynomial	0.125	5	-
265	Polynomial	0.125	7	-
266	RBF	0.5	-	0.0001
267	Polynomial	0.125	8	-
268	Polynomial	0.125	9	-
269	Polynomial	0.125	10	-
270	RBF	0.0625	-	0.001
271	Polynomial	0.0625	6	-
272	RBF	0.25	-	0.0001
273	Polynomial	0.0625	5	-
274	Polynomial	0.03125	5	-
275	Polynomial	0.03125	6	-
276	Polynomial	0.0625	7	-
277	RBF	0.125	-	0.0001
278	Polynomial	0.0625	8	-
279	Polynomial	0.0625	9	-
280	Polynomial	0.03125	4	-

Table A.12: Best ranking of hyperparameters for libSVM according to A-SMFO introduced in Chapter 4 (Part 9).

Rank	Kernel	C	Degree	Gamma
281	Polynomial	0.0625	10	-
282	RBF	0.0625	-	0.0001
283	Polynomial	0.03125	7	-
284	RBF	0.03125	-	0.001
285	Polynomial	0.03125	8	-
286	RBF	0.03125	-	0.0001
287	Polynomial	0.03125	9	-
288	Polynomial	0.03125	10	-

References

[1] Abdulrahman, S. M., Brazdil, P., van Rijn, J. N., & Vanschoren, J. (2018). Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine Learning*, 107(1), 79–108.

[2] Adankon, M. M. & Cheriet, M. (2009). Model selection for the LS-SVM. application to handwriting recognition. *Pattern Recognition*, 42(12), 3264–3270.

[3] Anderson, M. R., Antenucci, D., Bittorf, V., Burgess, M., Cafarella, M. J., Kumar, A., Niu, F., Park, Y., Ré, C., & Zhang, C. (2013). Brainwash: A data system for feature engineering. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*.

[4] Bardenet, R., Brendel, M., Kégl, B., & Sebag, M. (2013). Collaborative hyperparameter tuning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013* (pp. 199–207).

[5] Bengio, Y., Courville, A. C., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8), 1798–1828.

[6] Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.* (pp. 2546–2554).

[7] Bergstra, J. & Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13, 281–305.

[8] Brazdil, P., Giraud-Carrier, C. G., Soares, C., & Vilalta, R. (2009). *Metalearning - Applications to Data Mining*. Cognitive Technologies. Springer.

[9] Brazdil, P. & Soares, C. (2000). A comparison of ranking methods for classification algorithm selection. In *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings* (pp. 63–74).

[10] Caruana, R., Niculescu-Mizil, A., Crew, G., & Ksikes, A. (2004). Ensemble selection from libraries of models. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*.

[11] Cawley, G. (2001). Model selection for support vector machines via adaptive step-size tabu search. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, pp. 434-437, Prague, Czech Republic, April 2001*. (pp. 434–437). Prague, Czech Republic.

[12] Chang, C.-C. & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2, 27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[13] Chapelle, O., Vapnik, V., & Bengio, Y. (2002a). Model selection for small sample regression. *Machine Learning*, 48(1-3), 9–23.

[14] Chapelle, O., Vapnik, V., Bousquet, O., & Mukherjee, S. (2002b). Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3), 131–159.

[15] Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, San Francisco, CA, USA - August 13 - 17, 2016*.

[16] Cox, D. D. & John, S. (1997). Sdo: A statistical method for global optimization. In *in Multidisciplinary Design Optimization: State-of-the-Art* (pp. 315–329).

[17] David-Tabibi, O. & Netanyahu, N. S. (2002). Verified null-move pruning. *ICGA Journal*, 25(3), 153–161.

[18] de Souza, B. F., de Carvalho, A. C. P. L. F., Calvo, R., & Ishii, R. P. (2006). Multiclass SVM model selection using particle swarm optimization. In *6th International Conference on Hybrid Intelligent Systems (HIS 2006), 13-15 December 2006, Auckland, New Zealand* (pp.3).

[19] Deisenroth, M. P. & Ng, J. W. (2015). Distributed gaussian processes. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015* (pp. 1481–1490).

[20] Do, C. B., Foo, C., & Ng, A. Y. (2007). Efficient multiple hyperparameter learning for log-linear models. In *Advances in Neural Information Processing Systems 20, Proceedings of the*

Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007 (pp. 377–384).

[21] Domhan, T., Springenberg, J. T., & Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015* (pp. 3460–3468).

[22] Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., & Leyton-Brown, K. (2013). Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*.

[23] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada* (pp. 2962–2970).

[24] Feurer, M., Springenberg, J. T., & Hutter, F. (2014). Using meta-learning to initialize bayesian optimization of hyperparameters. In *ECAI workshop on Metalearning and Algorithm Selection (MetaSel)* (pp. 3–10).

[25] Friedrichs, F. & Igel, C. (2005). Evolutionary tuning of multiple svm parameters. *Neurocomput.*, 64, 107–117.

[26] Gomes, T. A. F., Prudêncio, R. B. C., Soares, C., Rossi, A. L. D., & Carvalho, A. C. P. L. F. (2012). Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, 75(1), 3–13.

[27] Grabocka, J., Schilling, N., Wistuba, M., & Schmidt-Thieme, L. (2014). Learning time-series shapelets. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014* (pp. 392–401).

[28] Guo, X. C., Yang, J. H., Wu, C. G., Wang, C. Y., & Liang, Y. C. (2008). A novel ls-svms hyper-parameter selection based on particle swarm optimization. *Neurocomput.*, 71(16-18), 3211–3215.

[29] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1), 10–18.

[30] Hansen, L. K. & Salamon, P. (1990). Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10), 993–1001.

[31] Hennig, P. & Schuler, C. J. (2012). Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13, 1809–1837.

[32] Hernández-Lobato, J. M., Hoffman, M. W., & Ghahramani, Z. (2014). Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada* (pp. 918–926).

[33] Hinton, G. E. (1999). Products of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks*, volume 1 (pp. 1–6).

[34] Hoffman, M. D., Brochu, E., & de Freitas, N. (2011). Portfolio allocation for bayesian optimization. In *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011* (pp. 327–336).

[35] Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05 (pp. 507–523). Berlin, Heidelberg: Springer-Verlag.

[36] Hutter, F., Xu, L., Hoos, H. H., & Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artif. Intell.*, 206, 79–111.

[37] Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *J. of Global Optimization*, 13(4), 455–492.

[38] Kapoor, A., Qi, Y. A., Ahn, H., & Picard, R. W. (2005). Hyperparameter and kernel learning for graph based semi-supervised classification. In *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]* (pp. 627–634).

[39] Keerthi, S. S., Sindhwani, V., & Chapelle, O. (2006). An efficient method for gradient-based adaptation of hyperparameters in SVM models. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006* (pp. 673–680).

[40] Kégl, B. & Busa-Fekete, R. (2009). Boosting products of base classifiers. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09 (pp. 497–504). New York, NY, USA: ACM.

[41] Kendall, M. G. (1938). A New Measure of Rank Correlation. *Biometrika*, 30(1/2), 81–93.

[42] Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1), 97–106.

[43] Lacoste, A., Larochelle, H., Marchand, M., & Laviolette, F. (2014). Sequential model-based ensemble optimization. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014* (pp. 440–448).

[44] Lai, T. & Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1), 4–22.

[45] Land, A. H. & Doig, A. G. (1960). An Automatic Method for Solving Discrete Programming Problems. *Econometrica*, 28, 497–520.

[46] Lawler, E. L. & Wood, D. E. (1966). Branch-And-Bound Methods: A Survey. *Operations Research*, 14(4), 699–719.

[47] Leite, R., Brazdil, P., & Vanschoren, J. (2012). Selecting classification algorithms with active testing. In *Proceedings of the 8th International Conference on Machine Learning and Data Mining in Pattern Recognition*, MLDM'12 (pp. 117–131). Berlin, Heidelberg: Springer-Verlag.

[48] Levesque, J., Gagné, C., & Sabourin, R. (2016). Bayesian hyperparameter optimization for ensemble learning. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, UAI 2016, June 25-29, 2016, New York City, NY, USA*.

[49] MacKay, D. J. C. (1992). Information-based objective functions for active data selection. *Neural Computation*, 4(4), 590–604.

[50] Masada, T., Fukagawa, D., Takasu, A., Hamada, T., Shibata, Y., & Oguri, K. (2009). Dynamic hyperparameter optimization for bayesian topical trend analysis. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009* (pp. 1831–1834).

[51] McQuarrie, A. D. & Tsai, C.-L. (1998). *Regression and time series model selection*. World Scientific.

[52] Metzen, J. H. (2016). Minimum regret search for single- and multi-task optimization. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016* (pp. 192–200).

[53] Mockus, J., Tiešis, V., & Žilinskas, A. (1978). The application of bayesian methods for seeking the extremum. *Towards Global Optimization*, 2, 117–129.

[54] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

[55] Pfahringer, B., Bensusan, H., & Giraud-Carrier, C. (2000). Meta-learning by landmarking various learning algorithms. In *In Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 743–750).: Morgan Kaufmann.

[56] Pinto, N., Doukhan, D., DiCarlo, J. J., & Cox, D. D. (2009). A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Computational Biology*, 5(11), e1000579. PMID: 19956750.

[57] Rasmussen, C. E. & Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

[58] Reif, M., Shafait, F., & Dengel, A. (2012). Meta-learning for evolutionary parameter optimization of classifiers. *Machine Learning*, 87(3), 357–380.

[59] Rendle, S. (2010). Factorization machines. In *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010* (pp. 995–1000).

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

[61] Schilling, N., Wistuba, M., Drumond, L., & Schmidt-Thieme, L. (2015). Hyperparameter optimization with factorized multilayer perceptrons. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II* (pp. 87–103).

[62] Schilling, N., Wistuba, M., & Schmidt-Thieme, L. (2016). Scalable hyperparameter optimization with products of gaussian process experts. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I* (pp. 33–48).

[63] Schneider, P., Biehl, M., & Hammer, B. (2010). Hyperparameter learning in probabilistic prototype-based models. *Neurocomputing*, 73(7-9), 1117–1124.

[64] Seeger, M. W. (2006). Cross-validation optimization for large scale hierarchical classification kernel methods. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006* (pp. 1233–1240).

[65] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & de Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1), 148–175.

[66] Shelley, M. W. (1818). *Frankenstein; or, The Modern Prometheus*. Lackington, Hughes, Harding, Mavor & Jones.

[67] Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1), 6:1–6:25.

[68] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.* (pp. 2960–2968).

[69] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M. M. A., Prabhat, & Adams, R. P. (2015). Scalable bayesian optimization using deep neural networks. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015* (pp. 2171–2180).

[70] Srinivas, N., Krause, A., Seeger, M., & Kakade, S. M. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. In J. Fürnkranz & T. Joachims (Eds.), *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (pp. 1015–1022).: Omnipress.

[71] Sun, Q. & Pfahringer, B. (2013). Pairwise meta-rules for better meta-learning-based algorithm ranking. *Machine Learning*, 93(1), 141–161.

[72] Swersky, K., Snoek, J., & Adams, R. P. (2013). Multi-task bayesian optimization. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.* (pp. 2004–2012).

[73] Swersky, K., Snoek, J., & Adams, R. P. (2014). Freeze-thaw bayesian optimization.

[74] Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4), 285.

[75] Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). 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*, KDD '13 (pp. 847–855). New York, NY, USA: ACM.

[76] Vilalta, R. & Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artif. Intell. Rev.*, 18(2), 77–95.

[77] Villemonteix, J., Vazquez, E., & Walter, E. (2009). An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4), 509–534.

[78] Wistuba, M., Duong-Trung, N., Schilling, N., & Schmidt-Thieme, L. (2016a). Bank card usage prediction exploiting geolocation information. *CoRR*, abs/1610.03996.

[79] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2015a). Hyperparameter search space pruning - A new component for sequential model-based hyperparameter optimization. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II* (pp. 104–119).

[80] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2015b). Learning data set similarities for hyperparameter optimization initializations. In *Proceedings of the 2015 International Workshop on Meta-Learning and Algorithm Selection co-located with European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2015 (ECMLPKDD 2015), Porto, Portugal, September 7th, 2015.* (pp. 15–26).

[81] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2015c). Learning hyperparameter optimization initializations. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Campus des Cordeliers, Paris, France, October 19-21, 2015* (pp. 1–10).

[82] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2015d). Sequential model-free hyperparameter tuning. In *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015* (pp. 1033–1038).

[83] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2016b). Hyperparameter optimization machines. In *2016 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2016, Montreal, QC, Canada, October 17-19, 2016* (pp. 41–50).

[84] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2016c). Two-stage transfer surrogate model for automatic hyperparameter optimization. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I* (pp. 199–214).

[85] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2017). Automatic Frankensteining: Creating complex ensembles autonomously. In *Proceedings of the 2017 SIAM International Conference on Data Mining, Houston, Texas, USA, April 27-29, 2017.* (pp. 741–749).

[86] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2018). Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1), 43–78.

[87] Yogatama, D. & Mann, G. (2014). Efficient transfer learning method for automatic hyperparameter tuning. In *International Conference on Artificial Intelligence and Statistics (AISTATS 2014)*.

Index