technische universität dortmund

fakultät statistik

Multi-objective Analysis of Machine Learning Algorithms Using Model-based Optimization Techniques

Dissertation

by

M.Sc. Data Science
DANIEL HORN

in partial fulfillment of
the requirements for the degree of
Doktor der Naturwissenschaften

Submitted:	Dortmund, January 2019
Primary referee:	Prof. Dr. Claus Weihs
Secondary referee:	JProf. Dr. Andreas Groll
Commission chairperson:	Prof. Dr. Jörg Rahnenführer
Assessor:	Dr. Michel Lang
Day of the oral examination:	20th February 2019

Preambel

This dissertation summarizes my work during the last five years, from October 2013 to December 2018 and covers all of my publications originating from this period. Since it is cumulative, only brief summaries of the contributed articles are given. Exhaustive results are mostly omitted and the original articles are referred instead. This text arranges the publications into the big picture and gives introductions to their respective research fields, starting from the absolute basics. For more in-depth details I recommend to read the respective articles themselves. Apart from the introduction and the conclusion, each chapter of this work concludes with a section on the contributed material. After a simple enumeration of the contributed papers and software libraries, a short description on how they arose follows.

Since this dissertation covers the fields of optimization, machine learning and algorithm selection, it was not possible to use a consistent notation throughout all chapters. Notation is only locally valid and can change when advancing to another research field. However, I tried to use some letters consistently during the entire work. X always corresponds to an influential parameter, while Y corresponds to a target variable and f to the functional connection between X and Y. Variations of these are used to describe the observations and parameter spaces. Table 1 gives an overview to this notation and can be used as an look-up table.

This work would not have been possible without the help of many people. At first, I want to thank Prof. Claus Weihs. I still remember this afternoon in January in 2013. I was about to start working on my master thesis and just made up my mind that I would like to stay at the university afterwards as an Ph.D. student. After his lecture in the course classification methods, I told him

$X^{(i)}$	i-th influential parameter	$Y\,/\,Y^{(i)}$	(i-th) target variable
$\mathscr{X}$	parameter space	$\mathscr{Y}$	objective space
$\mathscr{X}^{(i)}$	space of the i-th parameter	$\mathscr{Y}^{(i)}$	space of the i-th objective
$\boldsymbol{x}$	observation vector	$y/\boldsymbol{y}$	corresponding target value(s)
$\boldsymbol{x}_j$	optional numbering of $\boldsymbol{x}$	$y_j/\boldsymbol{y}_j$	optional numbering of $y/\boldsymbol{y}$
$x^{(i)}$	observation of the i-th parameter	$y^{(i)}$	observation of the i-th objective
$f : \mathscr{X} \to \mathscr{Y}$	functional relation of the $X^{(i)}$ and Y	$\hat{f}$	estimator / model for f

Table 1: *Notation used consistently through the entire work.*

about my decision and asked if I could start working at his chair. He just answered with the counterquestion: "Okay, when would you like to start?", and afterwards I was employed. Claus Weihs had trust in my competence and gave me the opportunity to follow my own ideas from the first day on. I am very grateful for this.

At second, I want to thank Bernd Bischl. Although Claus Weihs is the official supervisor of my dissertation, Bernd surely was my mentor, at least during its first years. Most of the things I have learned during the last year of my master's degree and in the first years of my graduation, I have learned from or because of Bernd. Since he left to Munich halfway through my graduation, it was complicated to continue the good collaboration in the subsequent years. Nevertheless, without Bernd many of the contributed work would not have been possible.

Moreover, I have to thank all remaining co-authors of my contributed publications. Tobias Wagner, especially for introducing me to many people in the multi-objective optimization community. Aydın Demircioğlu and Tobias Glasmacher for their support in the project SVMs for large data sets. Jakob Richter, Jakob Bossek, Janek Thomas and Michel Lang as the remaining members of the `mlrMBO` team. Martin Zaefferer for his work on hierarchical Kriging kernel, who had a hard time remotivating me after some first frustrating results. My student workers Karin Schork and Rosa Pink, who supported me in many ways. Swetlana Herbrandt and Nadja Bauer, as well as the remaining members of the chair for computational statistics, for all the informative discussions during the last five years. Andreas Groll, for agreeing to co-review this dissertation.

A special thank has to go to Rosa Pink, once again, for proof reading this text. I cannot count the hours we spend on discussing single sentences, even words, until this work was finally finished. Additional thanks go to Claus Weihs, Andreas Groll, Michael Kirchhoff, Jennifer Neuhaus-Stern, Marie-Louise Vosteen and Steffen Maletz for finding many mistakes and giving helpful comments.

And, at last, my biggest thank has to go to TU Dortmund university's faculty of statistics and all the countless students and colleagues I met during the last ten years. You made this place a home and yourselves a family for me.

Contents

1 Introduction **1**

2 Optimization **3**
- 2.1 Example: Support Vector Machines 5
- 2.2 Sequential model-based optimization 7
- 2.3 Multi-objective optimization 10
- 2.4 Contributed publications 12

3 Hyperparameter Tuning **15**
- 3.1 The machine learning process 16
- 3.2 Example: Hyperparameter tuning of an SVM 18
- 3.3 Tuning in mixed and hierarchical parameter spaces 21
- 3.4 Tuning as a Stochastic Optimization Problem 24
- 3.5 Contributed publications 25

4 Multi-objective selection of algorithm portfolios **27**
- 4.1 Example: Comparing different SVM solvers for large data sets 28
- 4.2 Selection of portfolios for single data sets 31
- 4.3 Validating MOSAP rules 32
- 4.4 Analyzing multiple data sets 34
- 4.5 Contributed publications 38

5 Conclusion and Future Work **39**

Literature **41**

List of abbreviations

BFGS	Optimization algorithm named after its authors initials
BVM	Ball Vector Machine, an approximative SVM solver
CMA-ES	Covariance Matrix Adaption Evolutionary Strategy
CVM	Core Vector Machine, an approximative SVM solver
DoE	Design of Experiments
EGO	Efficient Global Optimization algorithm, the first SMBO algorithm
EI	Expected Improvement, an infill criterion for SMBO
ES	Evolutionary Strategy, an optimization strategy
HV	Hypervolume, a multi-objective performance indicator
LASVM	An approximative SVM solver
LCB	Lower Confidence Bound, an infill criterion for SMBO
LHS	Latin Hypercube Sampling, a special DoE technique
LLSVM	Low-rank Linearized SVM, an approximative SVM solver
MBMO	Model-Based Multi-objective Optimization
MMCE	Mean Missclassification Error, a performance measure for classification
MOES	Multi-Objective Evolutionary Strategy
MOP	Multi-objective Optimization Problem
MSE	Mean Squared Error, a performance measure for regression
NFL	No Free Lunch theorem
OP	Optimization Problem
ParEGO	Pareto EGO, an MBMO strategy
RF	Random Forest, a machine learning method
RS	Random Search, the simplest optimization algorithm
SMBO	Sequential Model-Based Optimization, an optimization strategy
SMO	Sequential Minimal Optimization, the exact SVM solver
SMS-EGO	S-Metrix Selection EGO, an MBMO strategy
SVM	Support Vector Machine, a machine learning method
SVMperf	An approximative SVM solver

Chapter 1

Introduction

Data is the sword of the 21st century, those who wield it the samurai.

– Jonathan Rosenberg, former Senior Vice President of Products at Google

This and many other quotes can be heard these days when talking about data and data science. Data is called the *oil* or even the *soil* of the 21st century, data science the *sexiest* job. Many companies have distinct data science departments and some companies like Google or Facebook make billions of dollars every year, mainly based on their ability to process large datasets. Nowadays, to collect, analyze and learn from large amounts of data is both a challenge and an opportunity.

At the same time, data science has a relatively young history. The term itself goes back around forty years and has started to be more frequently used in the last decade. Study courses on data science aren't sprouting more than two or three years ago. However, data has been collected for much longer. The Romans and even earlier cultures performed population censuses and measured other related information, the probably most popular data collection is even mentioned in the Christmas story. Although it has not been called data science, data has been analyzed ever since. At the beginning of the 20th century, the term statistic arose for this scientific field and its mathematical foundations were laid. Data, however, was scarce, and statistic is often focused on making the best out of only a few observations. Today, in the age of the world wide web with companies like Google and Facebook, data is produced at a rapid pace. Ten years ago, in 2008, Google alone processed estimated 20 peta bytes of data per day.[1] Hence, data science often suffers from an abundance of available data, rendering most classic statistical methods inappropriate or even inapplicable. Nevertheless, both fields rely on the same foundation: on data.

Most data sets consist of two types of variables. First, there is a set called target variables, here denoted with Y. These are the variables one wants to gain new knowledge about. Examples are the quality of a product, the risk of an accident happening or the (monetary) profit. Often, only a single target variable is considered, however, in some situations it is necessary to investigate multiple ones. Second, there is a set called the independent variables, here denoted with X. These

[1] `https://www.niallkennedy.com/blog/2008/01/google-mapreduce-stats.html`, 09/18/2018

variables can be any other property of the unit of observation, as for example the height and the sex of a person, or some diagnostic measures of a workpiece. Moreover, X variables are assumed to have an influence on the Y variables: If the value of an X variable is changed, it is expected that also the value of the target variables changes, i.e. some kind of relationship $X \rightarrow Y$ is assumed.

Although data science and statistics are all about data, it is neither the X nor the Y variables one is mostly interested in. Instead, it is the arrow in-between those variables: To understand the relationship between the dependent and the target variables. Typical questions are:

- Interpretation: If a certain X variable is changed, how will it affect the Y variables?
- Prediction: Given values of the X variables, what values will the Y variables take?
- Optimization: Which values of the X variables will result in optimal values of the Y variables?

Finding good answers to these questions often requires a deep understanding of both, the X and the Y variables, as well as of the underlying data generating process.

Most times, these questions demand an important intermediate step: To find a good model describing the relationship between X and Y, also known as model selection. Here, not only an adequate model class has to be found, but also its hyperparameters have to be set. Hence, finding a good model is not only an intermediate step in most data analyses, it is also a data analysis itself, where the target variable is defined as the quality of the model and the dependent variables are the choice of the model class and its subsequent hyperparameters. Many data science problems imply solving this internal optimization problem, the optimization of the model quality. Therefore, the main focus of this work lays on optimization. It is organized as follows:

The second chapter gives an introduction into the topic of optimization. At first, an overview over different disciplines within the topic is given and the training of a support vector machine is presented as an exemplary task. Afterwards, the sequential model-based optimization approach for expensive problems and its extensions for multi-objective optimization problems are presented.

The third chapter focuses on hyperparameter tuning as a special optimization problem. At first, an introduction to machine learning and the general machine learning scheme is given. Again, the SVM is used as an example, here for tuning its hyperparameters, both in a single- and a multi-objective setting. Tuning as an optimization problem can have difficult properties. It can be both expensive and stochastic, and the parameter space can include mixed and hierarchical parameters. The chapter continues with adapting sequential model-based optimization for these properties.

The fourth chapter discusses algorithm selection. This special data science task deals with the choice of the best algorithm for solving a given problem. Here, Y is the available algorithm, while X variables include performances of the algorithms on various problems. This work addresses a multi-objective context. In contrast to the single-objective case, here not only a single but a whole set of optimal algorithms has to be selected with respect to multiple contradicting performance measures. Once again, the SVM is used as an application: Approximate SVM solvers are compared with respect to the objectives training time and missclassification error.

Chapter 2

Optimization

Optimization problems (OPs) occur in many – if not all – practical data analyses. In technical processes it may be the quality of the produced objects, in biometrics the efficacy of a drug, or model qualities in general statistic applications. No matter how diverse these situations are in their details, they do not differ in their main components. All of them feature some kind of process, which is controlled by several parameters. Given a parameter setting, i.e. a specific value for each parameter, the process can be executed and some sort of performance value is returned. The goal is to find parameter settings that result in best performances. This procedure is called optimization, the best setting, corresponding to an optimal performance, is called the optimum.

Mathematically, the underlying process can be described by a function $f : \mathscr{X} \to \mathbb{R}$, where $\mathscr{X} = \mathscr{X}^{(1)} \times \mathscr{X}^{(2)} \times ... \times \mathscr{X}^{(d)}$ denotes the set of all feasible parameter settings. For a specific parameter setting $\boldsymbol{x} = [x^{(1)}, x^{(2)}, \ldots, x^{(d)}]^T \in \mathscr{X}$, where $x^{(i)} \in \mathscr{X}^{(i)}$ is the value of parameter $X^{(i)}$, the associated fitness value is denoted by $f(\boldsymbol{x})$. Since maximizing f is equivalent to minimizing $-f$, only minimization problems are considered here. The (global) minimum of f is defined as the set of all solutions fulfilling

$$\min_{\boldsymbol{x} \in \mathscr{X}} f(\boldsymbol{x}) := \{\boldsymbol{x} \in \mathscr{X} \,|\, \nexists\, \tilde{\boldsymbol{x}} \in \mathscr{X} : f(\tilde{\boldsymbol{x}}) < f(\boldsymbol{x})\}.$$

The class of all OPs contains an incredible amount of rather distinct problems. Therefore, the existence of a global optimization strategy, working quite well for every OP, is pretty unlikely. An algorithm that performs reasonably well for many OPs is the most simple one: random search. Random search (RS) iteratively draws uniformly distributed settings from $\mathscr{X}$ and evaluates them with f until some kind of termination criterion is reached, usually a predefined budget. Finally the setting with minimal target value is considered as the optimum.

It has actually been proven by Wolpert and Macready (1997) that all optimization algorithms have the same expected performance as random search if the OP is chosen uniformly from all possible OPs. This behavior is known as the *no free lunch theorem* (NFL). However, it is a merely theoretic result and, in practice, algorithms do perform better than RS. Random search's most important advantage is also a huge disadvantage: It incorporates (almost) no application specific knowledge. Hence, if meta-information about a specific subclass of OPs is available, a reasonable

algorithm using this information should give better results than RS. This is not in contradiction with the NFL, since now the OP is not chosen from a uniform distribution of all possible OPs, but comes from a smaller subclass.

One example is the subclass of purely continuous OPs. In this subclass, the parameter space $\mathscr{X}^{(i)}$ of each parameter $X^{(i)}$ is given by an interval $\left[x_{left}^{(i)}; x_{right}^{(i)}\right]$. If it can also be assumed that f is convex and if the analytical form of its derivative is known, Quasi-Newton algorithms and especially the BFGS method (simultaneously published by Broyden, Fletcher, Goldfarb and Shanno in 1970) have shown to be very effective.[1] Another example is the pure discrete setting. Here, each parameter X_i can only take a finite number of different values. If the cardinality of $\mathscr{X}$ is small enough, every parameter setting can be evaluated with f and, consequently, the global optimum of f will be found.

In both examples, $\mathscr{X}$ is described by so-called box-constraints and no further restrictions are given. Such OPs are called unconstrained. Though, in many practical applications further restrictions on $\mathscr{X}$ do exist. For example, $\mathscr{X}$ can be confined by constraint functions g and h. A setting $\boldsymbol{x}$ is only feasible if both $g(\boldsymbol{x}) \geq 0$ and $h(\boldsymbol{x}) = 0$ hold. In section 2.1 an exemplary constraint OP is presented. As a second example, hierarchical parameter structures can exist. A hierarchical parameter $X^{(i)}$ is only active (i.e., has an influence on f) if other parameters fulfill certain conditions. Instances of such OPs are given in chapter 3. Further restrictions on $\mathscr{X}$ may be possible, but are not investigated here.

In the first example subclass, some major information on the structure of f is given: f is convex. However, in many practical OPs the structure of f is mostly unknown, aside from the parameter space $\mathscr{X}$. This class of OPs is referred to as blackbox OPs. It can be further distinguished between pure continuous settings, where all parameters are continuous, and mixed settings, where both continuous and discrete parameter do exist. Both, continuous and mixed blackbox OPs are addressed later in this work.

State-of-the-art approaches for solving continuous blackbox OPs include evolutionary strategies[2] (ESs). Especially the Covariance Matrix Adaption ES (CMA-ES) by Hansen et al. (2003) shows on-top performances in many ongoing benchmarks, for example in the blackbox optimization benchmarks by Hansen et al. (2016). However, ESs require (hundreds of) thousands of function evaluations until they converge towards the global optimum. This is infeasible for many applications due to possible high costs of single evaluations. Here, the term cost mostly refers to the time a single evaluation takes, but can also be interpreted in other application dependent ways as its monetary costs. Such OPs are referred to as expensive OPs.

In recent years, a class of algorithms for solving expensive OPs has been developed. Based on the efficient global optimization (EGO) procedure by Jones et al. (1998), their main idea is

[1] See, e.g., Weihs et al. (2013) for an introduction to Quasi-Newton methods.

[2] See, e.g., Yu and Gen (2015) for an introduction to evolutionary strategies.

to replace f by an inexpensive surrogate function $\hat{f}$. In order to estimate $\hat{f}$, some evaluations have to be performed with f itself. These so called sequential model-based optimization (SMBO) approaches iterate between evaluating f and optimizing $\hat{f}$. Parts of this work are focused on extending the general SMBO approach for some special applications.

This chapter continues by introducing the training of support vector machines (SVMs) as a specific class of OPs as well as a dedicated specialized optimization algorithm. Subsequently, the SMBO approach is explained in its details. For some OPs, not only a single but multiple objectives have to be optimized mutually. These multi-objective OPs are introduced in the last section, along with related extensions for the SMBO procedure.

2.1 Example: Support Vector Machines

The Support Vector Machine (SVM) (Cortes and Vapnik, 1995) is a binary classification method. For n observations $(\boldsymbol{x}_1^T,...,\boldsymbol{x}_n^T)^T$ of d continuous parameters and a binary label $y_i \in \mathscr{Y} =: \{-1,1\}$ for each observation, the SVM estimates the true, unknown relationship $f : \mathscr{X} \to \mathscr{Y}$. In its most simple form, the so-called linear hard-margin SVM assumes linearly separable classes. Thus, f is modeled by

$$\hat{f}(\boldsymbol{x}) = \text{sign}(\boldsymbol{\beta}^T\boldsymbol{x} - \beta_0),$$

with $\boldsymbol{\beta} \in \mathbb{R}^d$ and $\beta_0 \in \mathbb{R}$ such that $y_i \cdot (\boldsymbol{\beta}^T\boldsymbol{x}_i - \beta_0) \geq 1$ holds for all observations and $\boldsymbol{\beta}^T\boldsymbol{x}_i - \beta_0$ equals -1 and $+1$ for at least one observation. The line $\boldsymbol{\beta}^T\boldsymbol{x}_i - \beta_0 = 0$ is called decision boundary. Since there is an infinite amount of feasible vectors $(\beta_0, \boldsymbol{\beta})$, the SVM uses an additional optimality criterion: the size of the so-called *margin*. It is defined as the distance between the two hyperplanes $\boldsymbol{\beta}^T\boldsymbol{x} - \beta_0 = \pm 1$. As known from linear algebra, this distance is given by $\frac{2}{||\boldsymbol{\beta}||_2}$, whose maximization is equivalent to the easier minimization of $\frac{1}{2}||\boldsymbol{\beta}||_2^2$. Hence, training a linear hard-margin SVM means solving the optimization problem

$$\begin{aligned} \min_{\beta \in \mathbb{R}^d} \quad & \frac{1}{2}||\boldsymbol{\beta}||_2^2 \\ \text{subject to} \quad & y_i \cdot (\boldsymbol{\beta}^T\boldsymbol{x}_i - \beta_0) \geq 1 \quad \forall i = 1,...,n. \end{aligned}$$

The assumption of linear separable classes is rather unrealistic. Therefore, the soft-margin SVM allows some observations $\boldsymbol{x}_i$ to lie inside the margin, i.e. $y_i \cdot (\boldsymbol{\beta}^T\boldsymbol{x} - \beta_0) \in [0;1)$, or even on the wrong side of the decision boundary, i.e. $y_i \cdot (\boldsymbol{\beta}^T\boldsymbol{x} - \beta_0) < 0$. Figure 2.1 shows an exemplary data situation, including five observations violating the margin. The former hard constraints are softened by introducing so-called *slack variables* ξ_i and the OP is be extended to optimizing both the size of the margin and the sum of the slack variables. A hyperparameter C is introduced to

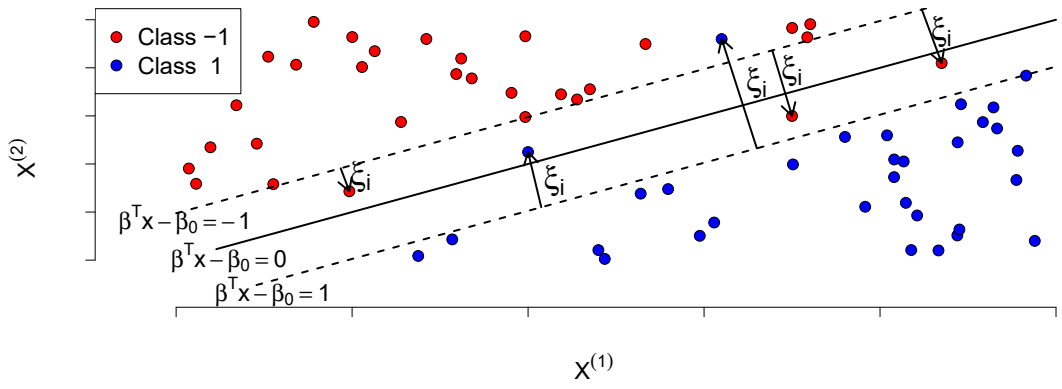


Figure 2.1 *Example classification problem for classes -1 and 1. The solid line corresponds to the decision boundary, the dashed lines visualize the margin.*

control the trade-off between those two objectives, resulting in the optimization problem

$$\begin{aligned} \min_{\beta \in \mathbb{R}^d} \quad & \frac{1}{2}||\boldsymbol{\beta}||_2^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{\beta}^T \boldsymbol{x}_i - \beta_0) \geq 1 - \xi_i \quad \forall i = 1,...,n. \end{aligned}$$

The assumption of a linear decision boundary between the two classes is also quite naive. To handle classification problems with non-linear decision boundaries, the SVM utilizes the so-called *kernel-trick*: By applying a function Ψ the parameters are transformed into a higher-dimensional feature space, in which the classes are more likely to be linearly separable. Finally, the OP of the non-linear SVM can be formulated as

$$\begin{aligned} \min_{\beta \in \mathbb{R}^d} \quad & \frac{1}{2}||\boldsymbol{\beta}||_2^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{\beta}^T \Psi(\boldsymbol{x}_i) - \beta_0) \geq 1 - \xi_i \quad \forall i = 1,...,n. \end{aligned}$$

According to Bottou and Lin (2007), this so-called primal OP is a quadratic, convex OP and can be solved with appropriate standard algorithms. However, some information on the structure of the OP is available and should be utilized. At first, the Karush-Kuhn-Tucker conditions (Kuhn

and Tucker, 1951) can be used to formulate the so-called dual OP

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \Psi(\boldsymbol{x}_i)^T \Psi(\boldsymbol{x}_j) \\ \text{subject to} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0 \text{ and } 0 \leq \alpha_i \leq C \; \forall i \in \{1, \ldots, n\}. \end{aligned} \tag{2.1}$$

This OP does no longer depend on the true coordinates $\Psi(\boldsymbol{x})$ of the observations in the higher-dimensional feature space, but only on the results of the scalar products $\Psi(\boldsymbol{x}_i)^T \Psi(\boldsymbol{x}_j) =: k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Hence, instead of defining and computing Ψ explicitly, it is implicitly given by the kernel function k that maps two observations $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ to a real number, the scalar product. In order to define a proper kernel, k must be a proper inner product.[3] In particular, the kernel matrix $\boldsymbol{K}$ with $\boldsymbol{K}_{i,j} := k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ must be positive semi-definite. A kernel enjoying most popularity is the radial basis function (RBF) kernel, defined by

$$k^{(RBF)}(\boldsymbol{x}_1, \boldsymbol{x}_2) := \exp\left(-\frac{||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2}{2\gamma^2} \right),$$

where γ is an additional kernel hyperparameter.

The dual OP is typically solved using the *sequential minimal optimization* (SMO) algorithm by Platt (1998), among others it is implemented in the C++ library LIBSVM (Chang and Lin, 2011). SMO refers to the strategy of analytically solving minimally sized sub-problems that allow for feasible update steps. Utilizing some additional performance enhancing techniques, the SMO algorithm is able to solve equation (2.1) with high precision. However, its runtime scales up to cubic with the data size (Bottou and Lin, 2007) and is therefore not suitable for large data sets. This problem is further addressed in chapter 4. The choice of the hyperparamter C, the kernel and its parameters is postponed to chapter 3.

2.2 Sequential model-based optimization

Sequential model-based optimization describes a class of algorithms developed for solving expensive optimization problems. It is a modular framework and can thus be customized for a variety of different tasks. Based on Bischl et al. (2017c), it consists of the following six main steps.

Step (1): Sample an initial design, containing n_{init} parameter settings $\boldsymbol{x}_j \in \mathscr{X}$ and evaluate it with f to yield outcomes $y_j = f(\boldsymbol{x}_j)$, $j = 1, \ldots, n_{\text{init}}$. Set $n = n_{\text{init}}$.

Step (2): Fit a surrogate model $\hat{f}$ to the current design respecting tuples $(y_j, \boldsymbol{x}_j)$, $j = 1, ..., n$.

Step (3): Optimize an infill criterion to propose l new parameter settings $\boldsymbol{x}_{n+j}$, $j = 1, \ldots, l$.

Step (4): Evaluate the proposed settings and add tuples $(y_{n+j}, \boldsymbol{x}_{n+j})$, $j = 1, \ldots, l$ to the design.

[3] See Young (1990) for an introduction to inner product spaces.

Step (5): If the termination criterion is not fulfilled, set $n = n + l$ and proceed with step 2.

Step (6): If the termination criterion is met, return the best evaluated parameter setting.

This general description is also illustrated in Figure 2.2. It omits most details, since each step can be instantiated in many different ways. Therefore, each step is explained more in-depth in the next paragraphs. As it is the first representative of his kind, the original implementations of the Efficient Global Optimization procedure by Jones et al. (1998) for continuous unconstrained OPs are highlighted.

(1) Initial Design The initial design is the starting point of the SMBO procedure. It is sampled from $\mathscr{X}$, evaluated with f and used to fit the initial surrogate model. Any design of experiment (DoE) technique can be used here, reaching from pure random sampling towards D-optimal designs.[4] However, most SMBO implementations rely on Latin hypercube sampling (LHS) as EGO does (Stein, 1987).

Recent studies have shown that the choice of the DoE techniques does not have a significant influence on the quality of the optimization result (Steponavičė et al., 2016). Even optimization results using pure random sampling are indistinguishable from results with more advanced DoEs. It just seems to be important for the initial design to completely cover $\mathscr{X}$. If the design is too small, the fit of the initial model may be poor, it may even be impossible to fit the model. If the size of the initial design is too large, the remaining budget may be too small to sufficiently optimize f. Common recommendations for n_{init} reach from $5d$ to $10d + 1$ parameter settings.

(2) The Surrogate model The surrogate model $\hat{f}$ is refitted in each iteration and represents the current state of knowledge about f. Since $\hat{f}$ is cheap to evaluate, it can be extensively scanned in order to find promising points for real evaluations with f. Every regression model class can be used as the surrogate, including simple linear models, random forests or neuronal nets[5]. However, using linear models as surrogates results in poor optimization performance (Weihs et al., 2017).

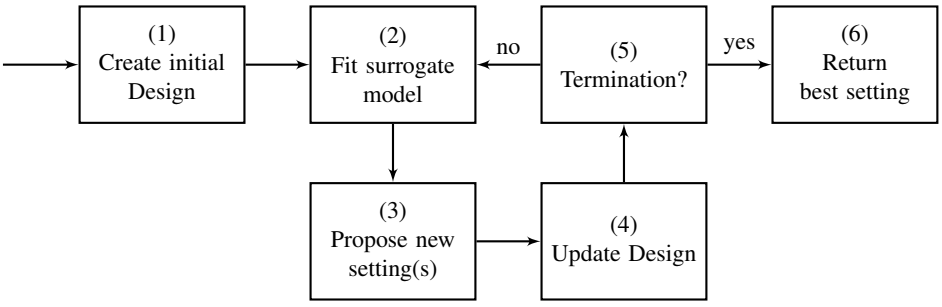


Figure 2.2: *Sketch of the SMBO approach.*

[4] See e.g. Montgomery (2006) for an introduction to design of experiments.

[5] See, e.g. Hastie et al. (2009) for an overview on regression model classes.

Common model class choices for $\hat{f}$ are Gaussian processes (and, especially, Kriging models) as in EGO or radial basis functions (Powell, 1992).

The choice of the model class (and its hyperparameters) is not easy, since typically no a-priori knowledge on the structure of f is available. Therefore, modern approaches include a model selection step. Here, multiple models from different classes (possibly including hyperparameter tunings) are fitted in each iteration and the best model is chosen. Bagheri et al. (2016) include a model selection step into an SMBO approach for solving constraint OPs. However, model selection can be time consuming, and, in extreme cases, more expensive than evaluating f itself.

(3a) Infill criterion The infill criterion is defined on the output of the surrogate model. It measures the potential of parameter settings for evaluation with f itself, balancing exploitation and exploration. This is usually achieved by combining posterior mean $\mu(\boldsymbol{x})$ and posterior standard deviation $s(\boldsymbol{x})$ in a well-balanced single formula. Both $\mu(\boldsymbol{x})$ and $s(\boldsymbol{x})$ are estimated by $\hat{f}$. Assuming that $\hat{f}$ is somewhat spatial in the sense that higher values of $s(\boldsymbol{x})$ indicate regions of the search space where no design points have been evaluated yet and / or the structure of f has not been well learned, parameter settings with low $\mu(\boldsymbol{x})$ and high $s(\boldsymbol{x})$ are most promising.

Arguably the most popular choice is the *expected improvement* $EI(\boldsymbol{x}) = \mathrm{E}(I(\boldsymbol{x}))$, where the random variable $I(\boldsymbol{x}) = \max(y_{\min} - \mu(\boldsymbol{x}), 0)$ defines the potential improvement for a parameter setting $\boldsymbol{x}$ over the currently best observed function value y_{min}. It was originally published by Mockus et al. (1978), its first uses in the context of SMBO go back to the EGO procedure (Jones et al., 1998). If $\mu(\boldsymbol{x})$ follows a normal distribution, as it does in the case of Kriging as the surrogate model, $\mathrm{EI}(\boldsymbol{x})$ can be expressed analytically in closed form as

$$\mathrm{EI}(\boldsymbol{x}) = (y_{\min} - \mu(\boldsymbol{x}))\,\Phi\left(\frac{y_{\min} - \mu(\boldsymbol{x})}{s(\boldsymbol{x})}\right) + s(\boldsymbol{x})\,\phi\left(\frac{y_{\min} - \mu(\boldsymbol{x})}{s(\boldsymbol{x})}\right),$$

where Φ and ϕ are the probability and density function of the standard normal distribution, respectively. A simpler approach to balance $\mu(\boldsymbol{x})$ and $s(\boldsymbol{x})$ is given by the *lower confidence bound*

$$\mathrm{LCB}(\boldsymbol{x}, \lambda) = \mu(\boldsymbol{x}) - \lambda \cdot s(\boldsymbol{x}),$$

where $\lambda > 0$ is a constant that controls the *exploration versus exploitation* trade-off. Weihs et al. (2017) show, that EI and LCB reach comparable optimization results .

(3b) Optimization of the infill criterion New parameter settings are proposed by optimizing the chosen infill criterion over $\mathscr{X}$. A single evaluation of the infill criterion is cheap, since it is based on cheap predictions with the surrogate model. Hence, a large number of different parameter settings can be investigated in order to find the most promising one. Since the infill optimization is just an intermediate step in the SMBO procedure, finding a *quite good* instead of the most promising parameter setting is sufficient. Therefore, any blackbox optimizer can be used here and

its choice should not have a significant impact on the global optimization result. Typical choices are ESs or branch-and-bound algorithms (Land and Doig, 1960) as in the original EGO procedure.

Contrary to proposing the single best parameter setting found through infill optimization, modern approaches try to find $l > 1$ settings per iteration. This is usually achieved by adapting the infill optimization step, Bischl et al. (2014) for example use an multi-objective infill optimization instead. Another option is to optimize l infill criteria in parallel and to propose each optimal point.

(4) Update In this intermediate step, the proposed points are evaluated with f. If more than one point is proposed ($l > 1$), these evaluations should be made simultaneously in order to decrease evaluation time. Especially in computer simulation, the usage of multiple CPUs for parallel evaluations can speed up optimization by an idealized factor up to the number of used CPUs. Afterwards the evaluated tuples are added to the current design.

(5) Termination SMBO implementations typically terminate when a given budget on time or number of function evaluations is exceeded. This behavior is poor, since no convergence guarantee can be given. Both early-stopping (the global optimum is not yet reached) and late-stopping (the global optimum was reached some iterations before) can occur. However, SMBO is motivated by practical applications that have prescribed budgets. If these budgets are exceeded, it is often not possible to continue the optimization, even in the case of early-stopping. Still, early-stopping should at least be detected and reported, and late-stopping is an issue that should be addressed. Finding useful termination criteria that prevent both early and late stopping is still a research question, some ideas have been published by Huang et al. (2006) and Weihs et al. (2017).

(6) Return Finally, an optimization result has to be returned. Typically, this is the parameter setting with the lowest observed function value. There are some other options available, such as fitting a final model and returning the best predicted setting, but they are not used frequently.

2.3 Multi-objective optimization

In some situations it is not sufficient to optimize a single objective. Examples may be technical processes, where not only quality, but also monetary costs have to be examined. In case of the soft-margin SVM, the size of the margin and the sum of the slack variables have to be considered. Instead of optimizing them simultaneously, the SVM reduces this biobjective OP into a single-objective one by summation of the objectives and introduces a hyperparameter C to balance them.

OPs like these with multiple objective functions $(f_1, ..., f_m) =: \boldsymbol{f}$, $\boldsymbol{f} : \mathscr{X} \to \mathbb{R}^m$, are called multi-objective OPs (MOPs). In general, the objectives are contradicting, and best achievable trade-offs are sought. Hence, in contrast to single-objective OPs, the optimum of a MOP is a set containing all parameter settings with optimal trade-offs.

A parameter setting $\boldsymbol{x} \in \mathscr{X}$ is said to be better as (also called to *dominate*) a setting $\boldsymbol{x}' \in \mathscr{X}$ if

$\boldsymbol{x}$ is at least as good as $\boldsymbol{x}'$ in all objectives and strictly better in at least one objective:

$$\begin{aligned}\boldsymbol{x} \prec \boldsymbol{x}' \Leftrightarrow & \forall i \in \{1,...,m\} : f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{x}') \\ & \wedge \exists \in \{1,...,m\} j : f_j(\boldsymbol{x}) < f_j(\boldsymbol{x}').\end{aligned}$$

This relation defines a partial order on $\mathscr{X}$, allowing incomparable parameters. It is sufficiently strong for a definition of optimality: a solution $\boldsymbol{x}$ is called Pareto optimal if and only if it is not dominated by any other solution $\boldsymbol{x}'$. The set $\left\{ f(\boldsymbol{x}) \,\middle|\, \boldsymbol{x} \in \mathscr{X} \text{ is Pareto optimal} \right\}$ of all non-dominated solutions is called Pareto front and is approximated in multi-objective optimization.[6]

MOPs are typically solved using evolutionary strategies. Since ESs are set-based optimizers, they can easily be extended to approximate Pareto fronts. Many multi-objective ESs (MOES) have been published in recent years, but they can not overcome the main disadvantage of their single-objective counterparts: A high number of function evaluations is required to reach the global optimum. Thus, MOESs are inappropriate for solving expensive MOPs. As in the single-objective case, the objective functions can be replaced by surrogate models. Several different approaches on extending the SMBO procedure towards MOPs, called model-based multi-objective optimization (MBMO) algorithms, have been proposed. Two of them are presented here, full taxonomies can be found in Horn et al. (2015) and Deb et al. (2017).

The first algorithm, ParEGO (Knowles, 2006), belongs to the class of scalarization-based MBMO algorithms. In each iteration, the m objectives are scalarized to a single one before the model is fitted. For scalarization, ParEGO uses the augmented Tschebyscheff norm

$$\kappa(\boldsymbol{x}) = \max_{j=1,...,d} \left\{ w_j \cdot (f_j(\boldsymbol{x}) - \boldsymbol{i}_j) \right\} + \rho \sum_{j=1}^{d} w_j \cdot (f_j(\boldsymbol{x}) - \boldsymbol{i}_j). \tag{2.2}$$

Here, $\boldsymbol{i} := \left(\min_{\boldsymbol{x} \in \mathbb{R}} f_1(\boldsymbol{x}), ..., \min_{\boldsymbol{x} \in \mathbb{R}} f_d(\boldsymbol{x}) \right)$ is the ideal point, $\boldsymbol{w}$ with $\sum_{j=1}^{m} w_j = 1$ is a weight vector controlling the trade-off between the objectives and ρ is a small, positive constant, typically set to 0.05. Afterwards, the general SMBO procedure is used in its original EGO instantiation to optimize the scalarized objectives. In order to generate optimal points along the entire Pareto front, ParEGO samples the weight vector anew in each iteration. In Horn et al. (2015), two extensions for the standard ParEGO algorithm are presented.

First, it is proposed to exchange the EI with the LCB infill criterion, since experiments based on artificial test functions show a clear advantage of LCB over EI. However, a newly discovered

[6] See e.g. Ehrgott (2013) for an extensive introduction to multi-objective optimization.

implementation bug leaves doubt on the bad performance of the EI.[7] Nevertheless, the LCB shows comparable results to other MBMO approaches and therefore offers a reasonable alternative.

Second, a multi-point extension for ParEGO is suggested. The original algorithm proposes only a single new point in each SMBO iteration. Horn et al. (2015) suggested to sample $c \cdot l$ different weight vectors per iteration. Afterwards, the most similar vectors are eliminated until only l remain. Hence, the selected weights cover the weight space in an almost uniform way. In their experiments, Horn et al. (2015) set $c = 5$. The approach showed top performance compared to other multi-point approaches, even exceeding the performance of original ParEGO with $l = 1$.

The second MBMO algorithm, SMS-EGO (Ponweiser et al., 2008), belongs to the class of direct indicator based algorithms. It fits individual models for each objective in step (2) of the general SMBO approach. Afterwards, in contrast to single-objective SMBO, the infill criterion is based on those $m > 1$ models. It measures the contribution of new parameter settings to the current approximation of the Pareto front using the hypervolume indicator (Zitzler et al., 2003). Consequently, the infill criterion still returns a single continuous value, and the standard SMBO procedure for proposing new points can be applied. SMS-EGO shows top performances among other MBMO algorithms in the experiments by Horn et al. (2015).

Just like for ParEGO, Horn et al. (2015) present a multi-point variant of SMS-EGO. Here, the concept of simulated evaluations is used: The first parameter setting $\boldsymbol{x}_{n+1}$ is proposed in the standard way by optimizing the SMS-EGO infill criterion. Afterwards, $\boldsymbol{x}_{n+1}$ is not immediately evaluated with the objective function, but its LCB value is added to the current design. Without refitting the surrogate model, a second setting $\boldsymbol{x}_{n+2}$ is proposed by optimizing the infill criterion again. $\boldsymbol{x}_{n+2}$ differs from $\boldsymbol{x}_{n+1}$, since the infill criterion of SMS-EGO is based on the Pareto front of the current design including $\text{LCB}(\boldsymbol{x}_{n+1}, \lambda)$. By adding the LCB value it is ensured that the proposed points are not too close to each other. This procedure iterates until l points have been proposed. After evaluating all l settings, the LCB values are replaced with the corresponding objective function values. The experiments in Horn et al. (2015) show that this variant performs slightly worse than the original SMS-EGO algorithm. However, being able to evaluate l points in parallel can speed up the optimization enough to justify this small deterioration.

2.4 Contributed publications

D. Horn, T. Wagner, D. Biermann, C. Weihs, and B. Bischl. Model-Based Multi-objective Optimization: Taxonomy, Multi-Point Proposal, Toolbox and Benchmark. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 64–78. Springer, 2015.

[7] A corrected version of the experiments is available in Bischl et al. (2017c), however, the comparison of the different infill criteria is omitted there.

C. Weihs, S. Herbrandt, N. Bauer, K. Friedrichs, and D. Horn. Efficient global optimization: Motivation, variations and applications. *Archives of Data Science, Series A (Online First)*, 2 (1):3–28, 2017.

B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang. mlrMBO: A modular framework for model-based optimization of expensive black-box functions. *arXiv preprint arXiv:1703.03373*, 2017c.

B. Bischl, M. Lang, J. Bossek, D. Horn, J. Richter, and D. Surmann. *BBmisc: Miscellaneous Helper Functions for B. Bischl*, 2017b. URL `https://CRAN.R-project.org/package=BBmisc`. R package version 1.11.

B. Bischl, M. Lang, J. Bossek, D. Horn, J. Richter, and P. Kerschke. *ParamHelpers: Helpers for Parameters in Black-Box Optimization, Tuning and Machine Learning*, 2017a. URL `https://CRAN.R-project.org/package=ParamHelpers`. R package version 1.10.

Since this chapter presents the fundamentals of my dissertation, the majority of its content is not my own work, but taken from original literature. Only the taxonomy of the MBMO algorithms and their batch proposals have been developed by Bernd Bischl, Tobias Wagner and myself. The results were presented at the 8th International Conference on Evolutionary Multi-Criterion Optimization 2015 in Guimarães, Portugal and the corresponding paper has been published in the associated conference volume (Horn et al., 2015). Moreover, in Weihs et al. (2017) some results regarding the usage of linear models as surrogates as well as a comparison between the EI and the LCB criterion have been contributed by myself.

In my first years as a Ph.D. student, I participated in the development of the `mlrMBO` software. `mlrMBO` is an `R`-package (R Core Team, 2018) that implements the general SMBO approach. It is currently available on CRAN in version 1.1.1, maintained by Jakob Richter. `mlrMBO` allows the instantiation of most SMBO steps in many different ways. In particular, `mlrMBO` is based on the `mlr` package (Bischl et al., 2016) and, therefore, any regression model available in `mlr` can be used as surrogate. `mlrMBO` also offers implementations of the MBMO algorithms ParEGO, SMS-EGO, ε-EGO (Wagner, 2013) and MSPOT (Zaefferer et al., 2013), integrated into the package mainly by myself. Furthermore, the package offers many user-friendly features like a decent error handling and parallel function evaluations in whose development and implementation I was involved. The development of `mlrMBO` resulted in a paper available on arXiv (Bischl et al., 2017c).

`mlrMBO` uses some helper packages. I am co-author of two of them: `ParamHelpers` (Bischl et al., 2017b) and `BBmisc` (Bischl et al., 2017a). `BBmisc` provides many small functions that make `R` programming a little bit more comfortable. `ParamHelpers` solves two issues: First, it implements a data structure that describes the parameter set of an objective function. Second, it implements a data structure that can trace an optimization process. It can contain all made function evaluations including a lot of additional information. Together with my student worker Karin Schork, I worked on the visualization of optimization processes: `ParamHelpers` is capable of plotting every stage of arbitrary OPs. An example optimization is visualized in Figure 2.3.

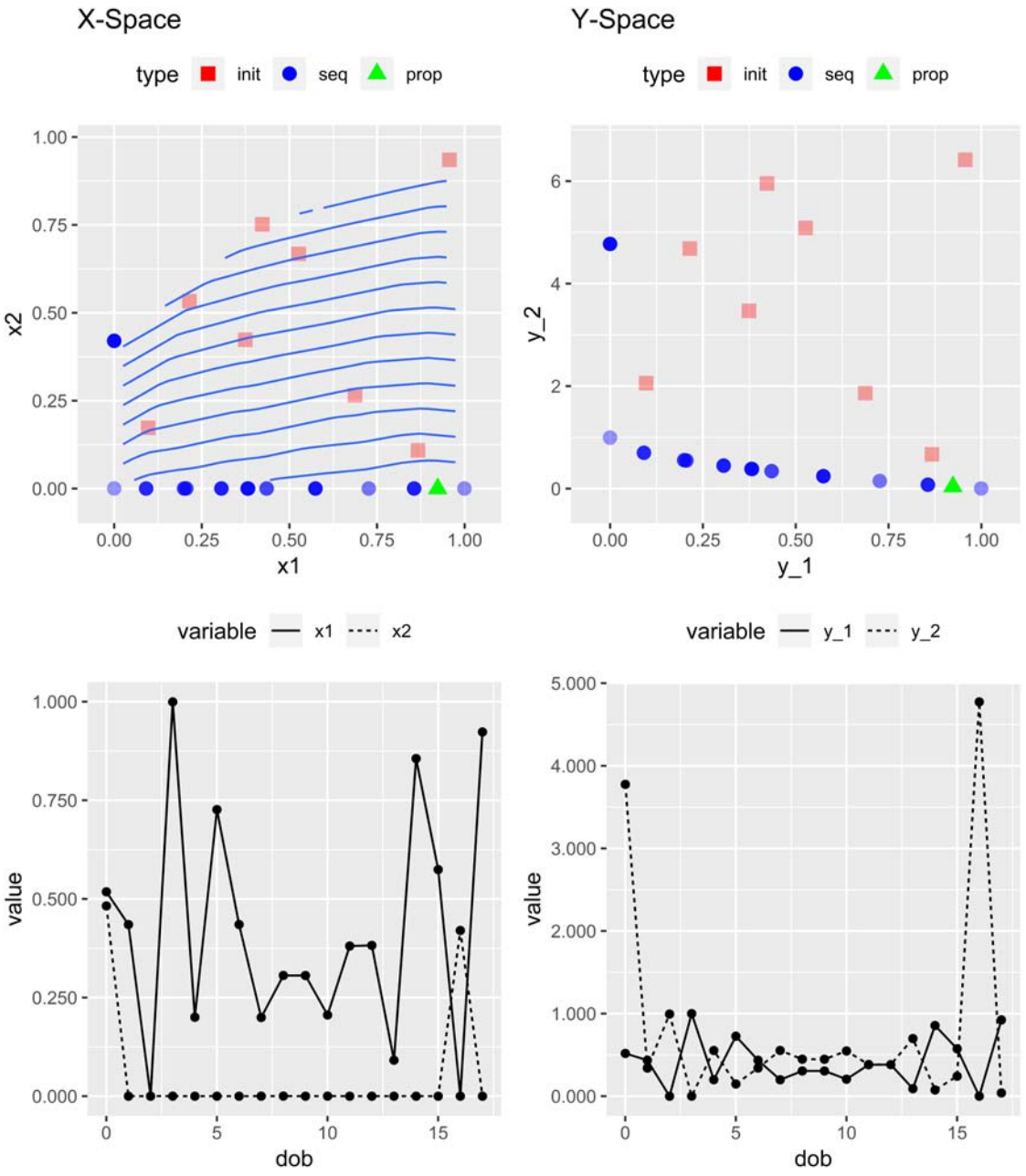


Figure 2.3 *Optimization path of the ZDT1 function (Zitzler et al., 2000) optimized with SMS-EGO. The top left plot shows all evaluated parameter settings in the parameter space. Red points correspond to settings evaluated during the initial design, blue points to settings evaluated during the sequential optimization, the green point to the recently evaluated setting. The blue lines are contour lines of the used Kriging model for the first objective. The top right plot shows the evaluated settings in the objective space, using the same colors as the top left plot. The intensity of the points in the upper plots corresponds to the iteration they were proposed in. The bottom plots show the time course. Here, dob is short for date of birth and is equivalent to the iteration of the optimization. In the bottom left plot, the value of each parameter in each iteration is shown, in the bottom right plot the value of each objective in each iteration.*

Chapter 3

Hyperparameter Tuning

As in optimization, in machine learning (ML) a function f is analyzed that describes the relationship between a set of parameters $\{X^{(1)}, ..., X^{(d)}\}$ and a target variable Y. In optimization, Y is continuous, $f : \mathscr{X} \to \mathbb{R}$ is known (or, at least, can be computed) and the goal is to find parameter settings corresponding to the smallest values of Y. Contrary, in ML, Y can be both continuous or discrete, f is unknown and the goal is to find a good model $\hat{f}$ approximating f. For this purpose, observations of $\{X^{(1)}, ..., X^{(d)}\}$ and corresponding values of Y are required. An example can be found in the SMBO procedure: In each iteration the true target function f is approximated by a surrogate model $\hat{f}$, which is estimated using already evaluated parameter settings.

Another example is introduced in section 2.1. The SVM is able to model the relationship between continuous parameters $X^{(i)}$ and a binary target variable. Prior to fitting an SVM model, some hyperparameters have to be set. This applies to the parameter C and the kernel parameter γ if the RBF kernel is used. Depending on the choice of those hyperparameters, the *goodness* (or *performance*) of the resulting model can vary greatly. Moreover, the performance of hyperparameter settings highly depends on the particular data and cannot be estimated a-priori. The process of finding good hyperparameter settings is called *hyperparameter tuning*.

Hyperparameter tuning is not an SVM-specific problem. In fact, most model classes have multiple hyperparameters influencing their model's performances. For many of them, default settings that will often attain high performing models are available. However, hyperparameter tuning is required if model performance has to be maximized. For some classes like the SVM, a good hyperparameter setting is obligatory in order to obtain a reasonable model.

This chapter continues with an introduction to machine learning and hyperparameter tuning. It will eventually turn out that tuning is just a special optimization problem and thus, algorithms from chapter 2 can be used. If the number of observations is large, tuning is an expensive OP and the SMBO procedure should be applied. The following section returns to the SVM and presents both exemplary single- and multi-objective hyperparameter tunings solved with SMBO algorithms. Hyperparameter tuning can be easily extended in a way that the standard SMBO procedure is no longer applicable. The two remaining sections consider two extensions and show how to incorporate them in the SMBO procedure: mixed parameter spaces and stochastic optimization.

3.1 The machine learning process

Machine learning is an extensive topic covering a lot of different applications. Each of them comes with a different background, history, theory, model classes and much more. However, the general course of action is the same for all of them. It is visualized in Figure 3.1, motivated by the workflow of the `mlr` R-package (Bischl et al., 2016). More details on machine learning and machine learning algorithms are given by Hastie et al. (2009).

The central aspect of each ML problem is the underlying data set. It consists of n observations $(\boldsymbol{x}_1^T,...,\boldsymbol{x}_n^T)^T =: \boldsymbol{X}$ of d variables $X^{(1)},\ldots,X^{(d)}$, also called parameters or features. As in chapter 2, $\mathscr{X}^{(i)}$ denotes the set of all possible values of $X^{(i)}$ and $\mathscr{X} := \mathscr{X}^{(1)} \times \mathscr{X}^{(2)} \times ... \times \mathscr{X}^{(d)}$. The parameters are assumed to have an influence on (typically) a single target variable Y with possible values $\mathscr{Y}$. In contrast to chapter 2, Y is not limited to be continuous ($\mathscr{Y} \subset \mathbb{R}$), but can also be discrete, as in case of the SVM ($\mathscr{Y} = \{-1,+1\}$). Often, observations $(y_1,...,y_n) =: \boldsymbol{y}$ of Y are available. This case is referred to as supervised learning. If no values of Y are given, the problem is called unsupervised. Depending on supervision and the type of Y, ML problems can be categorized into the subclasses regression, classification and cluster analysis (see table 3.1).

No matter which subclass a specific ML problem belongs to, the goal of an ML algorithm (also called *learner*) is to approximate $f : \mathscr{X} \to \mathscr{Y}$ by a model function $\hat{f}$. The calculation of a specific model is called training. Naturally, the goal is not to find any model, but rather to find the best one (or at least one that is good enough). Therefore the model performance, describing the goodness of the model, has to be quantified. Typically, the prediction performance of a learner is of interest, i.e., the model's ability to predict the outcome y_j for an observation $\boldsymbol{x}_j$ not present in the data set. Therefore a set of new observations, which are not used for training the model, and

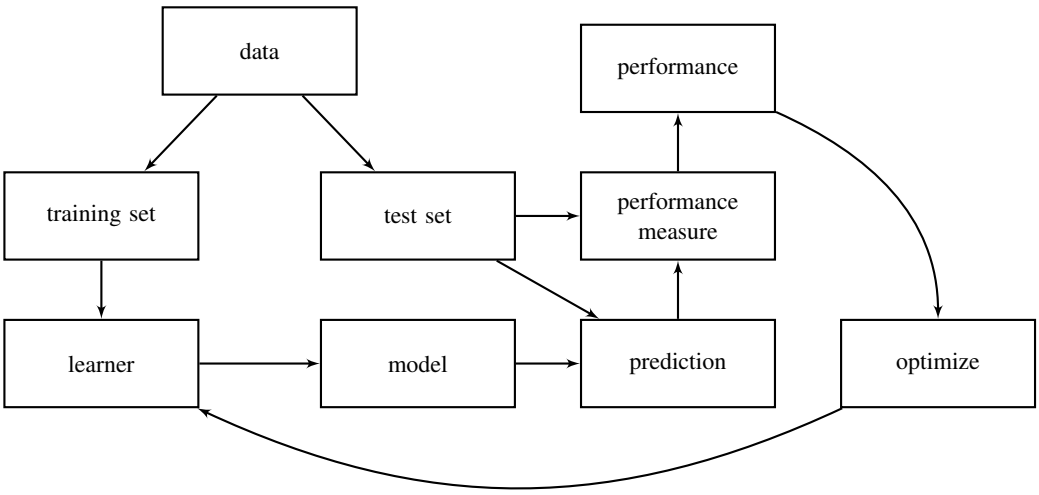


Figure 3.1: *Sketch of the machine learning workflow.*

	Y is continuous	Y is discrete
Y is available	regression	classification
Y is not available	-	cluster analysis

Table 3.1: *Different subclasses of ML problems, depending on the form of the target variable Y.*

corresponding realizations of Y are required. They are typically generated through a resampling procedure. The original data $\{\boldsymbol{X},\boldsymbol{y}\}$ is repeatedly split into training set $\{\boldsymbol{X}^{(train)},\boldsymbol{y}^{(train)}\}$ and test sets$\{\boldsymbol{X}^{(test)},\boldsymbol{y}^{(test)}\}$, containing $n^{(train)}$ and $n^{(test)}$ observations respectively.

Hence, a typical ML procedure runs through the following steps (c.f. Figure 3.1):

1. Calculate training and test sets.
2. For each training set: Train the chosen learner on $\{\boldsymbol{X}^{(train)},\boldsymbol{y}^{(train)}\}$ and get model $\hat{f}$.
3. Predict with each $\hat{f}$ on the corresponding $\boldsymbol{X}^{(test)}$ to get estimates of the outcomes $\hat{\boldsymbol{y}}^{(test)}$.
4. Evaluate the performance measure using the true and the predicted outcomes $\boldsymbol{y}^{(test)}$ and $\hat{\boldsymbol{y}}^{(test)}$.
5. If the performance is not satisfying, go back to step 2.

Training and test sets First, the training and test sets have to be selected via resampling. Typical resampling procedures include holdout, crossvalidation, bootstrapping and subsampling. All of them have in common that a random subset of the data is chosen as training set and the remaining observations are used as test set. This is repeated until the required number of resampling iterations is reached. The methods only differ in how the random subset is chosen: Bootstrapping performs random sampling with replacement, subsampling without. Holdout is equivalent to subsampling with only a single training set. In a k-fold crossvalidation, each observation is guaranteed to be in exactly $k-1$ training sets and in one test set. Moreover, in some applications it is be necessary to include additional domain specific knowledge. For instance, if data of k different months is available, it might be necessary to use the data of $k-1$ months for training and the last month for testing or to use stratification so that each month is equally often present in each training set.

Training and prediction Second, the chosen learner is applied to all training sets and the learner returns a model $\hat{f}$ for each training set. The SVM, for example, trains its model parameters by optimizing the margin size. Afterwards, $\hat{f}$ is used to predict the outcome of all observations on the corresponding test set: $\hat{y}_i^{(test)} = \hat{f}\left(\boldsymbol{x}_i^{(test)}\right)$, $i = 1,...,n^{(test)}$.

Performance Finally, the vector of predicted outcomes $\hat{\boldsymbol{y}}^{(test)} := \left(\hat{y}_1^{(test)},...,\hat{y}_{n^{(test)}}^{(test)}\right)$ has to be compared with the vector of true outcomes $\boldsymbol{y}^{(test)}$. Therefore a performance function mapping those two vectors to a single, real value has to be defined. Depending on the subclass of the ML problem, typical measures are the mean squared error (MSE) for regression

$$MSE(\hat{\boldsymbol{y}}^{(test)},\boldsymbol{y}^{(test)}) = \frac{1}{n^{(test)}} \sum_{i=1}^{n^{(test)}} \left(\hat{y}_i^{(test)} - y_i^{(test)}\right)^2$$

and the mean misclassification error (MMCE) for classification

$$MMCE(\hat{\boldsymbol{y}}^{(test)}, \boldsymbol{y}^{(test)}) = \frac{1}{n^{(test)}} \sum_{i=1}^{n^{(test)}} \mathbb{1}_{\left\{\hat{y}_i^{(test)} \neq y_i^{(test)}\right\}},$$

where $\mathbb{1}$ denotes the indicator function. In this manner, a performance measure is calculated for each training set. The global performance of the model then is given by the average of the performance measures over all resampling iterations.

Optimization Combining all prior steps, a process $f^{(tune)}$ is defined that maps a learner with an associated hyperarameter setting to a single real number representing the learner's performance. Hence, an order can be defined on the models with respect to their performances. By repeating the previous three steps with different learners or hyperparameter settings, it can be searched for the best one. This process is called hyperparameter tuning and can actually be interpreted as an OP with target function $f^{(tune)}$ and parameters $X^{(i)}$ corresponding to the hyperparameters.

Optimal hyperparameters can now be found using standard optimization algorithms. For instance, in the case of an SVM, the performance can be optimized with respect to the hyperparameters C and γ. Since the OP has just two continuous parameters, it can be solved using all algorithms for continuous OPs from chapter 2 like random search or the BFGS method. However, since estimating the performance of a model can take quite some time for larger data sets, specialized algorithms for expensive OPs like the SMBO approach should be used.

In a more complex setting, $X^{(1)}$ can be a discrete parameter describing the choice of the learner. Let, for example, $\mathscr{X}^{(1)} = \{\texttt{SVM}, \texttt{randomForest}\}$. If $X^{(1)}$ takes the value `SVM`, an SVM model is fitted, for `randomForest` a random forest (RF, Breiman (2001)) is chosen. The optimizer can vary the model class, in order to automatically determine the best one. This idea might sound promising, but is difficult to put into practice, because the underlying OP is hard to solve. Since $X^{(1)}$ is discrete and most model classes contain continuous hyperparameter, the OP is very likely to have a mixed parameter space. Moreover, the hyperparameters of a specific learner only influence the performance if that learner is chosen and therefore $\mathscr{X}$ includes hierarchical parameters. Hence, an algorithm for solving an expensive OP with mixed and hierarchical parameters is required.

3.2 Example: Hyperparameter tuning of an SVM

As mentioned in the prior sections, the hyperparameters C and γ of an SVM can be set via parameter tuning. This section presents an exemplary hyperparameter tuning on the *eeg-eye-state* data set. For SVM training, the implementation of the SMO algorithm from the `libSVM` library (Chang and Lin, 2011) is used, available in `R` via the package `e1071` (Meyer et al., 2017).

The data set is taken from the OpenML web service (Vanschoren et al., 2013) and was originally published in the UCI machine learning repository. (Dheeru and Karra Taniskidou, 2017)

It contains 14 features measured with the EEG method, describing brain activities of the test persons[1]. The investigation target is to identify whether the eyes of the test person were opened (6 723 persons) or closed (8 357 persons) during the measurement. Although it is not a really large data set (14 980 observations in total), it is already large enough to point out the limits of SVMs.

Now, a performance measure and a resampling procedure have to be selected. Since no additional information is available, the default performance measure seems adequate and hence the MMCE is used. To reduce the experiment's runtime, a 3-fold crossvalidation is applied for resampling. Moreover, parameter spaces for C and γ have to be defined. Both parameters can take any positive real number. Since the order of magnitudes is known to be more important than the value itself, both parameters are varied on the exponential scale $2^{[-15,15]}$. Hence, while C and γ are used for training, the optimizer gets $\log_2(C)$ and $\log_2(\gamma)$ as parameters.

At last, an optimization algorithm has to be chosen. Here, the results of two optimizers are compared: Grid search as the probably most prominent tuning algorithm for SVMs, and EGO as the standard SMBO approach. Each algorithm is given a budget of 49 performance measurements ($\equiv$ function evaluations). Hence, grid search can evaluate a 7×7 grid. The initial design of the SMBO approach consists of 8 points, leaving 41 evaluations for the sequential optimization. The tuning is performed using the `mlr`–package (internally using `mlrMBO`), all remaining tuning parameters are set to `mlr` defaults, which can be looked up in the `mlr` manual.

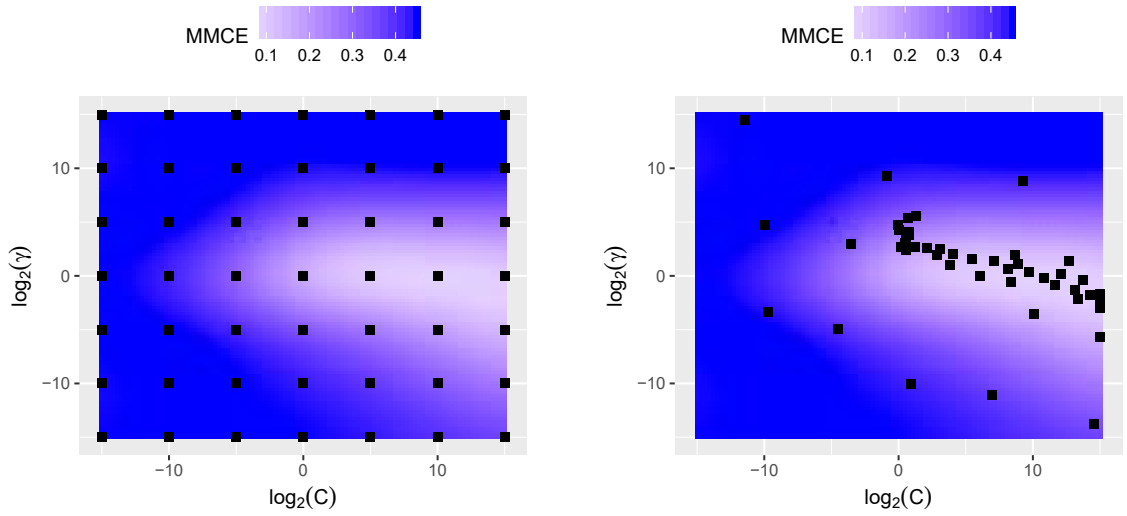
The results of the tunings are displayed in Figure 3.2. Obviously, the grid search wastes a lot of its budget for hyperparameter settings with high MMCEs, while the SMBO procedure focuses most of its function evaluations on regions with top performances. Both optimizers return hyperparameter settings with comparable MMCEs, slightly favoring the SMBO approach (grid search: 0.0923, SMBO: 0.0909). However, after only 17 of its 41 iterations SMBO finds a setting with an MMCE of 0.0917, which is already better than the final grid search MMCE.

In this example, a single fit of an SVM model took between 10 and 300 seconds (median: 30 seconds).[2] Taking the three crossvalidation folds and the 49 iterations into account, each tuning spend more than two hours on SVM training. This is still an acceptable time for parameter tuning. However, the *eeg-eye-state* data set consists of less than 15 000 observations. Both, the training time of a single SVM and the sequential computation time of the parameter tuning will quickly increase for larger data sets. Tuning algorithms can be sped up using parallel computing. In particular, the grid search can evaluate all 49 hyperparameter settings simultaneously. Thus, the real elapsed time can be much less than the sequential computation time.

Since many data sets include at least some redundant information, random subsets are likely to capture the data sets main essence. A simple trick to reduce training time is to train the learner on

[1] See, e.g., Teplan (2002) for explanations on EEG measurements.

[2] The experiments were run on an Intel(R) Core(TM) i5-4460 @ 3.20GHz with 8 GB RAM using R 3.5.0 on Windows 7 Enterprise.



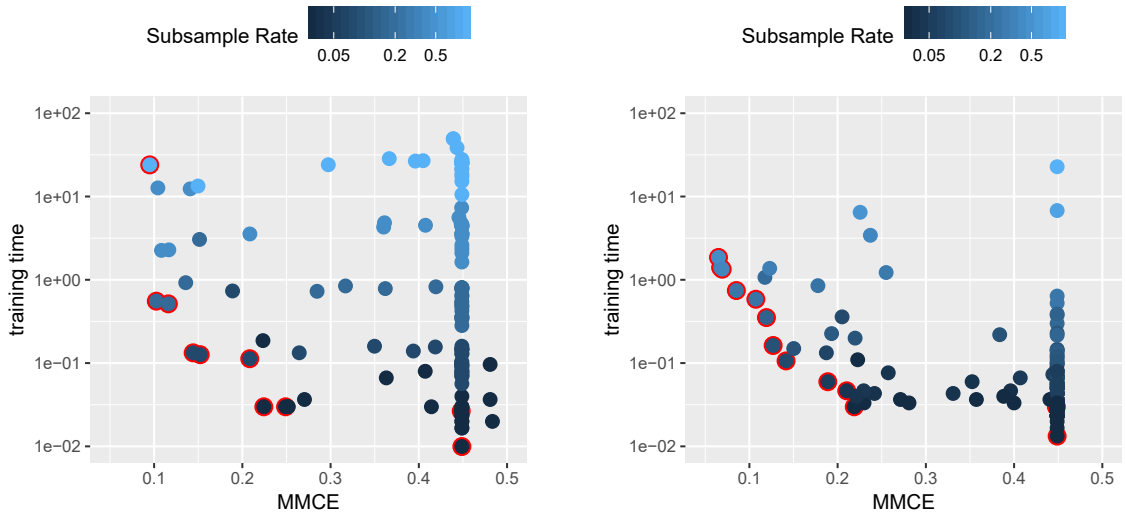
(a) Grid Search (b) SMBO

Figure 3.2 *Results of the single-objective tuning. The background shows the target function's landscape. Since the true landscape is unknown, it is interpolated with a Kriging model based on the 49 evaluations of grid search.*

such a subset (*downsampling*). Hereby, each training data set $\boldsymbol{X}^{(train)}$ is shuffled at first. For each performance assessment, the model is trained on the first $\lambda \cdot n_{train}$ observations of $\boldsymbol{X}^{(train)}$, where $\lambda \in (0,1]$ is the downsampling rate. The performance itself is still measured on the entire test data sets $\boldsymbol{X}^{(test)}$. The smaller λ, the faster the training will be. However, a smaller subset is also more likely to result in a worse MMCE. The effect of downsampling can be studied by multi-objective parameter tuning: In addition to the MMCE, the duration of a single SVM training is minimized as a second objective. λ is considered as an additional hyperparameter and is tuned together with SVM parameters C and γ. The parameter spaces of C and γ are again set to $2^{[-15,15]}$, while for λ the space $2^{[-7,0]}$ is used. Again, two tuning approaches are compared: Grid search and ParEGO as a standard SMBO variant. In order to compensate the additional hyperparameter λ, the budget is increased to 125 evaluations. All remaining settings are left at their `mlr` defaults.

Figure 3.3 shows the resulting Pareto fronts of both tunings. ParEGO reaches a clearly better approximation of the Pareto front, it finds both more and better points than grid search. The most outstanding part of the fronts may be the cluster of points with an MMCE of 0.45. This MMCE equates to the ratio of the majority class in the *eeg-eye-state* data set. Since the SVM falls back to predicting the majority class if parametrized poorly, many settings end up with that error.

The best found hyperparameter settings by ParEGO have MMCEs reaching down to 0.065, even better than the best settings from the single-objective tunings. Apparently, the increased budget does not just compensate for the additional parameter but also improves the optimization result. Moreover, since only 42% of the data are used to achieve this top performance, the training time



(a) Grid Search (b) SMBO

Figure 3.3 *Results from the multi-objective tunings for MMCE vs. training time, Pareto optimal points are marked in red.*

is decreased to only 1.5 seconds. Further reductions of training data lead to further improvements in training time, but come with the cost of higher MMCEs.

3.3 Tuning in mixed and hierarchical parameter spaces

In the previous section the standard SMBO procedure was used, since all considered hyperparameters were continuous. However, many ML algorithms have discrete hyperparameters. Further, if the learner itself is considered as an additional hyperparameter, also hierarchical parameters are present. Both discrete and hierarchical parameters can not be handled by the standard SMBO procedure. Horn and Bischl (2016) explain how it can be extended towards such mixed and hierarchical parameter spaces. This is achieved by adapting each individual SMBO step.

(1) Initial Design In most SMBO algorithms, LHS is used as initial design, which is defined for continuous parameter spaces only. Horn and Bischl (2016) propose an alternative based on a thinning approach: First, a pure random design with size $c \cdot n_{init}$, $c \gg 1$, is sampled. Afterwards, pairwise distances between all design points are calculated and one of the two points with minimal distance is removed from the design repeatedly until the design is reduced to size n_{init}. Thus it is ensured that no two points are too close to each other and the complete parameter space is covered. The approach leaves open the choice of an appropriate distance measure. For mixed and hierarchical parameter spaces, the Gower distance (Gower, 1971) may be used.

(2) The surrogate model Most SMBO approaches use Kriging or RBF models. Both model classes presume the parameter space to be continuous and, thus, they are not applicable here.

However, since any regression model can be picked as surrogate, they can be replaced by a more suitable one. Hutter et al. (2011) propose to use random forests, Horn and Bischl (2016) adopt this idea. RFs are a model class with many practical properties: They allow both continuous and discrete parameters and, given enough data, can model any functional relationship. Moreover, they often result in high-performance models even without hyperparameter tuning.

The handling of the hierarchical parameter space is a different matter. Even if the *i*th parameter of a setting $\boldsymbol{x}$ is inactive, a value $\boldsymbol{x}_i$ is still available, although it does not have an influence on the objective function. Hence, the missing information has to be encoded in some way. Horn and Bischl (2016) propose to treat these values as missings. Although an RF can fit a model on this missing values, a simple imputation technique should be applied to increase performance. Missing values of a continuous parameter $X^{(i)}$ are imputed with $x^{(i)}_{right} + 2 \cdot \left(x^{(i)}_{left} - x^{(i)}_{right}\right)$, i.e. a value outside its parameter space $\left[x^{(i)}_{left}; x^{(i)}_{right}\right]$, for discrete features a new level is introduced. Since the trees of the RF perform splits based on single variables, it can split the data into active and inactive settings in a single node of each tree. Thereby it is ensured that the RF can still recognize and utilize the missing-information in the data.

(3b) Infill optimization In each SMBO iteration, the infill criterion has to be optimized. Since it is defined on the same parameter space as the target function, the optimizer has to operate on both mixed and hierarchical parameters. Since the evaluation of the infill criterion is cheap, it can be evaluated with a large number of different parameter settings. Therefore, even RS is a suitable alternative here. Horn and Bischl (2016) describe an enhanced version called *focus search*. It performs multiple RS iterations and, in between the iterations, shrinks the parameter space around currently found optima.

(3a, 4 – 6) Infill criteria, Update, Termination, Return None of these three steps is affected by the parameter space and, thus, no adaption is needed.

Horn and Bischl (2016) survey whether these adaptions result in a reasonable SMBO variant. In their experiments, the performance measures false-positive rate and false-negative rate are optimized simultaneously on nine different binary classification data sets, i.e. a ROC like curve is estimated.[3] The multi-objective tuning is performed over three different model classes (RF, SVM and regularized logistic regression), where the choice of the model is a tuning hyperparameter itself. Moreover, weighting the positive class is allowed via weight-hyperparameter ω. Hereby each observation of the positive class is given the weight ω. For $\omega \to \infty$, all observations will be classified in the positive class and therefore the false-positive rate will eventually be 1, while the false-negative rate will be 0, and vice versa for $\omega \to 0$. Hence, a trade-off between the two objectives can be achieved using ω. The resulting parameter space is shown in Figure 3.4. The adapted SMBO variant outperformed a baseline on nearly all considered data sets.

[3]More details on performance measures are given by Hastie et al. (2009).

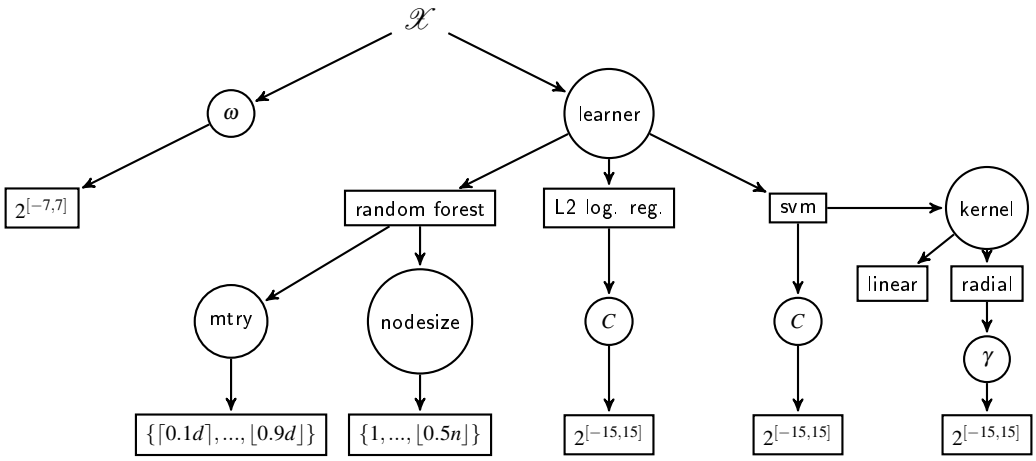


Figure 3.4 *Joint parameter space of the multi-objective tuning experiment with the three learners random forest, SVM and L2 regularized logistic regression . Circles denote variables, rectangles denote values of continuous and discrete parameters, arrows denote the hierarchic structure. Here n denotes the number of observations, d the number of features in the respective data set.*

It is questionable how safe the results by Horn and Bischl (2016) are, since the proposed method is only compared with a baseline. However, up to today no other multi-objective optimization methods for expensive OPs with mixed and hierarchical parameter spaces have been published. Certainly, this topic is not fully exploited yet and improvements are still necessary.

The choice of the surrogate model class appears to be most crucial. RFs do not seem to be the best choice. Especially for continuous parameter spaces, RFs are missing some important properties compared to Kriging models. First, Kriging's uncertainty estimator has a spatial interpretation: It is zero for all training observations and it increases with an increasing distance to the nearest training observation. Hence, the uncertainty can guide the optimization towards parameter regions that have not been exploited yet. RFs do not provide such an uncertainty estimator. Some kind of uncertainty can be simulated via bagging (Breiman, 1996). However, such estimators lack the spatial interpretation. Second, since RF models consist of trees, their models are always step functions. Obviously, step functions are not as suitable in guiding the SMBO procedure as smooth functions. This could be avoided by using extremely randomized trees (Geurts et al., 2006), however, no experiments on belonging optimization performances have been published yet.

Because of RF's disadvantages, it might prove promising to reconsider Kriging models. Some approaches exist that extend Kriging toward more complex parameter spaces. Since Kriging is a kernelized method, only the kernel and its underlying distance function have to be changed. Hutter and Osborne (2013) present a kernel for mixed and hierarchical parameter spaces, called the Arc-kernel. Zaefferer and Horn (2018) propose additional kernels and compare them with the Arc-kernel on a simple artificial test function. Although the results are promising, results for hyperparameter tuning experiments as well as a comparison with RFs are still missing.

3.4 Tuning as a Stochastic Optimization Problem

Up to this section, only deterministic OPs are considered: Evaluating the same parameter setting $\boldsymbol{x}$ multiple times with objective function f results in identical function values. This is an unrealistic assumption for many practical OPs, where repeated evaluations of the same setting will lead to differing values due to noise. In such so-called stochastic OPs, f itself cannot be observed. Instead, only observations of $\tilde{f}(\boldsymbol{x}) := f(\boldsymbol{x}) + \varepsilon(\boldsymbol{x})$ are available, where $\varepsilon(\boldsymbol{x})$ is the observational noise, a random variable with unknown distribution and expected value 0.

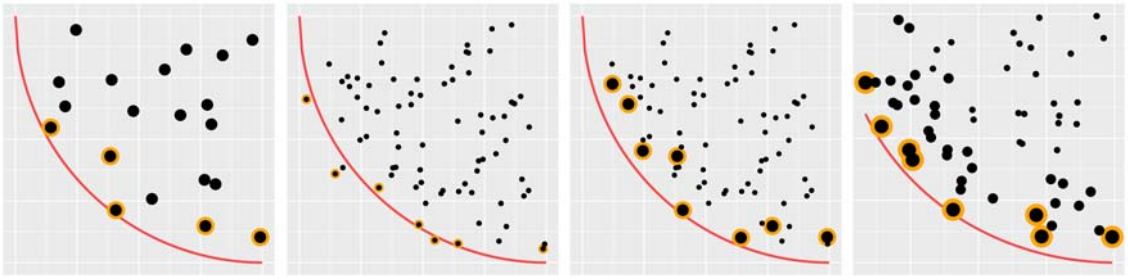
In fact, hyperparameter tuning is a stochastic OP, since the performance of each model is estimated using a resampling procedure. Evaluating the same setting $\boldsymbol{x}$ on different training and test sets results in varying performance values and $\varepsilon(\boldsymbol{x})$ only depends on the choice of these sets.[4] In previous sections, hyperparameter tuning is treated as a deterministic OP by using the same, fixed train and test sets for each performance assessment. Doing so, the same realization of $\varepsilon(\boldsymbol{x})$ is used each time $\tilde{f}(\boldsymbol{x})$ is evaluated and the corresponding fixed version of $\tilde{f}$ is optimized afterwards. However, the optimum of this new target function is likely to differ from the optimum of f.

f can be estimated from $\tilde{f}$ by using repeated evaluations (*re-evaluations*) of the same parameter settings. Since $\varepsilon(\boldsymbol{x})$ is assumed to have expected value 0, $\lim_{k\to\infty} \frac{1}{k} \sum_{i=1}^{k} \tilde{f}(\boldsymbol{x}) = f(\boldsymbol{x})$ holds and can be used as and estimator for $f(\boldsymbol{x})$ whose quality increases with k. Naturally, k should be chosen as large as necessary and as small as possible. This idea is already utilized in resampling procedures: Performing a k-fold cross validation corresponds to doing k re-evaluations.

Simple approaches use a constant k for each parameter setting. However, especially in the context of expensive OPs, it could be beneficial to adapt k for different parameter settings to save unnecessary evaluations. On the one hand, for the worst settings a single evaluation may be enough to reject them and the remaining evaluations can be omitted. On the other hand, for the best settings k evaluations may not be enough to detect relevant differences between them and additional re-evaluations should be made. A simple approach to determine k is to perform re-evaluations until significant differences with respect to some statistical test can be observed.

Horn et al. (2017b) propose and compare different methods to choose the (individual) numbers of re-evaluations in a multi-objective model-based setting. Each strategy is allowed to perform a total number of $k \cdot n$ evaluations. Figure 3.5 visualizes the behavior of all four approaches for some random data points. The *repeated* variant uses a constant amount of k re-evaluations for each parameter setting, just like the standard cross validation does. Doing so, only n different parameter settings can be investigated. The *enlarged* variant evaluates every setting just once. The saved budget is used to examine more distinct parameter settings, so that a total amount of $k \cdot n$ settings is evaluated. The *reinforced* variant starts like the enlarged one by evaluating many settings just once. In order to increase the reliability of the final Pareto front, the last part of the

[4] Here it is assumed, that a deterministic model class like SVM is used.



(a) Repeated (b) Enlarged (c) Reinforced (d) Rolling Tide

Figure 3.5 *Visualization of the four re-evaluation strategies. Point sizes correspond to the number of re-evaluations. The true Pareto front is marked by a red line, Pareto optimal points by orange circles.*

budget is used to evaluate all settings in the final Pareto set k times. Depending on the size of the final Pareto front, this strategy evaluates a large amount of rather bad settings just once and a small amount of promising settings k times. The *Rolling Tide* variant is based on the Rolling Tide ES by Fieldsend and Everson (2015). In addition to the newly proposed point, $k-1$ promising settings are re-evaluated in each iteration. In this way, n different parameter settings are investigated during the optimization and their count of re-evaluations increases with their quality.

Horn et al. (2017b) conduct two series of experiments, one using artificial test functions with varying noise and one using a real multi-objective tuning setting. The results show no clear winner, and especially the variance of the noisy target function has a major impact. For small amounts of noise, the reinforced approach is the best one. For larger amounts, it is beaten by the Rolling Tide variant. In the tuning experiment, Rolling Tide showed best performances again. Overall, including the stochasticity into the tuning process seems to be a promising approach. However, very little work on stochastic expensive multi-objective optimization has been published.

3.5 Contributed publications

C. Weihs, S. Herbrandt, N. Bauer, K. Friedrichs, and D. Horn. Efficient global optimization: Motivation, variations and applications. *Archives of Data Science, Series A (Online First)*, 2 (1):3–28, 2017.

D. Horn and B. Bischl. Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–8. IEEE, 2016.

M. Zaefferer and D. Horn. A first analysis of kernels for kriging-based optimization in hierarchical search spaces. In *Parallel Problem Solving from Nature – PPSN XV*, pages 399–410, Cham, 2018. Springer International Publishing.

D. Horn, M. Dagge, X. Sun, and B. Bischl. First investigations on noisy model-based multi-objective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 298–313. Springer, 2017b.

In the second half of my time as a Ph.D. student, I mainly worked on developing and refining hyperparameter tuning methods. Thereby I especially focused on the large data application, where a single model fit can take several minutes, hours or even days and, hence, as few model fits as possible should be conducted. Although I am mostly interested in the case of multi-objective tuning, most developed methods can also be applied to the single-objective case.

My work on this topic showed that simple tuning algorithms like RS and grid search are viable options for large data sets. First, they can work on any parameter space, continuous or mixed, constrained or unconstrained, having hierarchical parameters or not. Second, both algorithms can evaluate all their search points in parallel. If a sufficient amount of parallel computation power is available, this advantage more than makes up for the missing guidance in the optimization process itself. In Weihs et al. (2017) some of my experiments showing these advantages are published.

If, however, only limited parallel computation power is available, SMBO is superior to RS. The default SMBO procedure is defined for continuous parameter spaces only. We described in Horn and Bischl (2016), how the individual SMBO steps can be adapted for mixed and hierarchical parameter spaces. The paper itself focuses on the multi-objective case, but most of its ideas can be directly adopted for the single-objective case. For most SMBO steps, the extension to more advanced parameter spaces is straight forward and, especially, the infill criterion does not have to be adapted at all. Unfortunately, the most crucial step is also the one that performs worst. The commonly used Kriging model is defined for continuous parameter spaces only in its original form and has to be adapted by exchanging its kernel.

External literature proposes to use a regression random forest instead, but using RF as surrogate often results in an optimization performance that is only slightly better than random search. This happens, because RFs are missing some of Kriging's convenient properties. An alternative is to extend the Kriging model towards more complicated parameter spaces. Martin Zaefferer and I recently proposed some variants (Zaefferer and Horn, 2018) and compared them to an approach by Hutter and Osborne (2013). We limited ourselves to tests on a simple artificial test function. Real experiments using realistic tuning problems are planed for future work.

In 2016, I took a look at the stochasticity of tuning problems. In Horn et al. (2017b) we incorporated and compared noise handling mechanisms in the MBMO procedure. At EMO2017, I used the proposed algorithm to participate in the Video Games Track of the Black Box Optimization Competition (Loshchilov and Glasmachers, 2018) that featured expensive, stochastic, multi-objective optimization problems with continuous parameter spaces. I scored the first place, however, the competition had only two serious participants. In my opinion this shows how little work on this important topic has been done yet.

In sum, I have developed an optimization algorithm for expensive, stochastic, multi-objective OPs with mixed and hierarchical parameter spaces, well suited for solving multi-objective parameter tuning problems. Unfortunately, it is hard to say how good this algorithm really is, since I am not aware of any competitor algorithms in this setup.

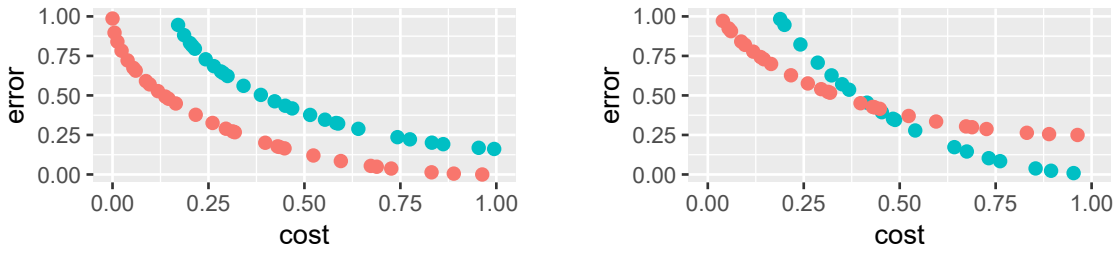
Chapter 4

Multi-objective selection of algorithm portfolios

For many tasks, a large quantity of different algorithms is available, forcing the user to choose a suitable one. For example, although the training of an SVM is typically done using SMO, dozens of alternatives are at hand. Choosing an algorithm with respect to a single performance measure (e.g. the MMCE for SVMs) is straightforward: After tuning each algorithm's individual hyperparameters, the algorithm with optimal performance on an independent test set can be selected.

In numerous situations, additional performance measures are required. For instance, training time of SMO is considered as a second performance criterion in section 3.2. Although typically contradicting, both cost and solution quality of an algorithm often have to be optimized simultaneously. Contrary to the case with a single criterion, it is ambiguous how to select the best algorithm with respect to multiple ones. Nevertheless, a (now multi-objective) hyperparameter tuning has to be performed for each algorithm at first. Afterwards the resulting Pareto fronts can be compared. Most times there will be no single algorithm covering the complete combined Pareto front, but a set of algorithms alternating in dominating each other. Consequently, not a single best, but a portfolio of Pareto optimal algorithms has to be selected. Moreover, a rule is required to specify which algorithm should be used for a given trade-off between the objectives.

Figure 4.1 shows simulated Pareto fronts of the objectives cost and error. It imitates results from individual hyperparameter tunings of two algorithms Red and Blue. In the left plot, Red clearly outperforms Blue along the entire front and should therefore always be selected. In the



(a) Algorithm Red dominates algorithm Blue.

(b) Algorithms take turns in dominating each other.

Figure 4.1: *Example situations for multi-objective algorithm selection with artificial data.*

right plot, both Red and Blue take a share of the combined Pareto front: Blue should be used for solutions with low error, Red if solutions with low cost are desired.

This chapter continues with further investigations on SVMs. In the first section, a multi-objective comparison of some algorithms for approximate SVM training is presented. The second section presents a method to select an optimal algorithm portfolio based on single data sets. Moreover, optimal portfolios for single data sets of the SVM comparative study are shown. The third section presents a validation of the method, the fourth section an extension towards multiple data sets and final portfolios for the SVM study.

4.1 Example: Comparing different SVM solvers for large data sets

Bottou and Lin (2007) state that exact training of non-linear SVMs via SMO has a runtime complexity between $O(n^2)$ and $O(n^3)$. Consequently, the SMO algorithm can not be used for large data sets. Alternative training algorithms apply approximations to the SVM OP in order to speed up the optimization. In most algorithms, the degree of approximation can be controlled through hyperparameters, so the user can specify whether he wants a fast or a precise solution.

In practical applications it is unclear which approximating algorithm should be picked. Horn et al. (2016a) try to answer this with an extensive simulation study. Naturally, only implementations and not algorithms are compared. The implementation of SMO in the LIBSVM library (Chang and Lin, 2011) is chosen as baseline. Several well-known implementations of alternative algorithms are considered, covering a representative set of different approximation techniques:

BVM/CVM: Reformulations of the SVM as enclosing ball problems, solved by specialized approximate algorithms (Tsang et al., 2005) / (Tsang et al., 2007).

LASVM: An online learning variant of SMO (Bordes et al., 2005).

LLSVM: A low-rank linearization of the SVM solved with LIBLINEAR by Fan et al. (2008) (Zhang et al., 2012) .

SVMperf: An adaptation of the cutting-plane method to SVM training (Joachims and Yu, 2009).

For each implementation, an independent multi-objective hyperparameter tuning is performed. Beside the common SVM parameters C and γ with the typical parameter space $2^{[-15,15]}$, hyperparameters controlling the approximation degree are tuned for each algorithm (cf. table 4.1). Each

SVM solver	Parameters	Optimization Spaces
BVM/CVM	ε (Accuracy)	$2^{[-13,-1]}$
LASVM	ε (Accuracy), #Epochs	$2^{[-13,-1]}$, $2^{[0,7]}$
LIBSVM	ε (Accuracy)	$2^{[-13,-1]}$
LLSVM	Matrix rank	$2^{[4,11]}$
SVMperf	ε (Accuracy), #Cutting planes	$2^{[-13,-1]}$, $2^{[4,11]}$

Table 4.1: *Hyperparameters of the SVM solver including parameter spaces for the tuning.*

model is trained on the first 50% of the available observations, its training time is measured and the MMCE is estimated on the next 25%. The estimated MMCEs of the Pareto optimal parameter settings are too optimistic because the settings are optimized to work well on these observations. In order to get realistic MMCEs, an additional validation step is executed: The models are re-trained on the first 50% of the data and the last 25% are use used to estimate the performance.[1] In the following, only these validation results are shown.

A simple downsampling technique is used as an additional baseline. The downsampling rate $\lambda \in 2^{[-7,0]}$ is treated as an additional hyperparameter subject to tuning. Hence, two hyperparameter tunings are performed for each algorithm: One with disabled and one with enabled downsampling. The algorithms are compared on several data sets collected from the LIBSVM web page (Chang and Lin, 2011), in addition some private data sets are used. Each tuning is conducted using the batch variant of ParEGO with a batch size of 20 and a total budget of 220 performance assessments. The experiment is implemented using the `mlrMBO` package and executed on the LiDO Cluster of TU Dortmund university.

The resulting Pareto fronts on one data set are shown in Figure 4.2, extensive results are available online (cf. Horn et al. (2016b)). In the results without downsampling (left plot), LIBSVM is the fastest algorithm reaching top MMCEs. The approximate solvers can accelerate training up to a factor of 200, but this gain is always bought with higher MMCEs. Especially the Pareto front

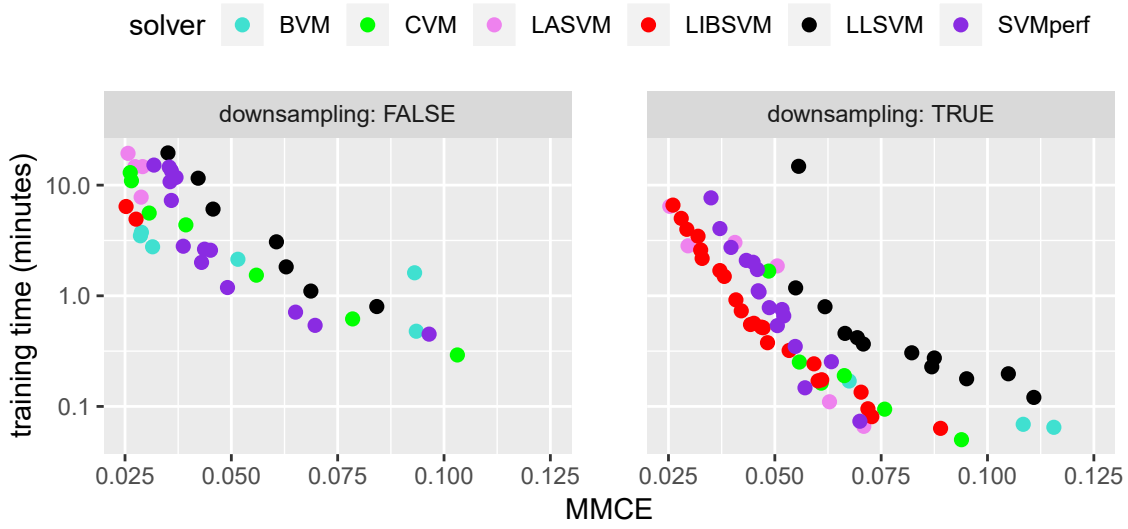


Figure 4.2: *Pareto fronts for the multi-objective hyperparameter tuning of the approximative SVM solver.*

[1] Normally, the model should have been retrained on the union of 50% and 25% of the data used in the tuning and validated on the remaining 25%. However, since the training times on 75% of the data would be incomparable to those on just 50%, a different approach has been chosen here.

of SVMperf shows a good trade-off between the two objectives. An adequate portfolio consists of LIBSVM for high quality solutions and SVMperf for faster solutions.

Although this result is not consistent for all considered data sets, it holds for most of them. This can be quantified with the Hypervolume indicator (HV, c.f. Zitzler et al. (2003)). For two objectives, the HV can be interpreted as the area between the Pareto front and a predefined reference point. Hence, large HV values indicate a high performing Pareto front. Here, the HV is used to compare three different portfolios: One containing all six algorithms (*all*), one containing only LIBSVM and SVMperf (*set 1*) and one containing LIBSVM, SVMperf and BVM (*set 2*). Table 4.2 shows the quotients between the HV of all algorithms and the HVs of the smaller portfolios. On seven out of twelve data sets, the quotient of set 1 is higher than 0.9, hence, LIBSVM and SVMperf form a sound portfolio. However, the performance is rather poor on some of the remaining data sets. The addition of BVM as a third algorithm yields considerably better results for all these data sets.

The right plot of Figure 4.2 shows the results with enabled downsampling. Most algorithms reach clearly better trade-offs between the objectives, but the best trade-offs are now achieved by LIBSVM itself. This is consistent for most data sets. In Table 4.3 the portfolio consisting of only LIBSVM is compared to the portfolio containing all six algorithms. On eight out of the twelve data sets, LIBSVM scores Pareto fronts only slightly worse than the combined fronts of all algorithms and it fails to build an adequate front only on the spektren data set.

In sum, these results are devastating for all approximate SVM solvers. Each algorithm itself is able to accelerate SVM training with the cost of a worse model. However, most times the achieved trade-offs between training time and solution quality are worse than the trade-offs for LIBSVM combined with a simple downsampling strategy. Hence, an optimal algorithm portfolio could consist of only LIBSVM. If the data set is too large to be solved in a reasonable amount of time, downsampling should be applied to decrease the training time.

data set	arthrosis	aXa	cod-rna	covtype	ijcnn1	mnist
HV(Set 1) / HV(all)	0.998	0.975	0.720	1.000	0.496	0.966
HV(Set 2) / HV(all)	0.998	0.975	0.882	1.000	0.922	0.966

data set	poker	protein	shuttle	spektren	vehicle	wXa
HV(Set 1) / HV(all)	1.000	1.000	0.121	0.044	0.942	0.351
HV(Set 2) / HV(all)	1.000	1.000	0.939	0.940	0.942	0.854

Table 4.2 *Hypervolume ratios without downsampling. all: a portfolio containing all six algorithms Set 1: a portfolio containing libSVM and SVMperf, Set 2: a portfolio containing libSVM, SVMperf amd libBVM.*

data set	arthrosis	aXa	cod-rna	covtype	ijcnn1	mnist
HV(libSVM) / HV(all)	0.564	0.912	0.977	1.000	0.942	0.957

data set	poker	protein	shuttle	spektren	vehicle	wXa
HV(libSVM) / HV(all)	0.703	0.681	0.961	0.096	0.950	0.992

Table 4.3 *Hypervolume ratios with downsampling: all: a portfolio containing all six algorithms libSVM: a portfolio containing only libSVM.*

4.2 Selection of portfolios for single data sets

In the previous sections, only an exploratory analysis of the simulation results has been done. However, manually looking through tuning results is time consuming and the chosen portfolio is always biased towards personal views. An automatic method is required to select optimal algorithm portfolios. According to Horn et al. (2017a), the main question for multi-objective selection of algorithm portfolios (MOSAP) problems can be formulated as:

Choose a portfolio of algorithms as small as possible, with a median Pareto front as close to the median [combined] Pareto front of all algorithms as possible. Which algorithms should be selected for the portfolio, and which of them should be used for which trade-off?

In MOSAP problems, k algorithms $\{A_1,\ldots,A_k\} =: \mathscr{A}$ are compared over multiple data sets with respect to multiple performance measures. For each algorithm on each data set an individual multi-objective hyperparameter tuning has to be performed at first. Due to stochasticity in both tuning and performance assessment, the tuning should be repeated $r > 1$ times. A MOSAP method calculates a decision rule for selecting a suitable algorithm $A_j \in \mathscr{A}$ for each trade-off between the performance measures. The method proposed by Horn et al. (2017a) is only defined for single data sets and two arbitrary performance measures. It consists of three steps:

Step 0: Normalization In order to get interpretable results, the objectives are normalized to $[0;1]$ at first.

Step 1: Remove dominated algorithms Some algorithms do not contribute to the combined Pareto front at all. These algorithms should be detected and removed at first. Horn et al. (2017a) propose to remove an algorithm if it does not score at least one non-dominated point in at least $\lceil \eta \cdot n \rceil$ out of n repetitions, where $\eta \in [0,1]$ is a control parameter.

Step 2: Select the best algorithm portfolio Although all remaining algorithms contribute to the combined Pareto front, some of these contributions may be insignificant. The corresponding algorithms should be removed, since the portfolio aims to be as small as possible. This is a biobjective problem itself: Select a set of algorithms with minimal size and maximal performance. While the size of a portfolio can simply be counted, its quality can be measured in different ways. Horn et al. (2017a) propose to use the HV difference between the examined portfolio and the portfolio

of all algorithms. In order to respect the *r* replications, the HV is calculated based on the 50% empirical attainment function (EAF, c.f. López-Ibáñez et al. (2010)). Since the set and, thus, the power set of algorithms is finite, this OP can be solved by enumerating all combinations: For each subset of algorithms both size and quality are reported. Afterwards, the best portfolio is selected by minimizing a scalarization of the objectives, the augmented Tschebyscheff norm. Its weight vector *w* controls whether a small or a high performing portfolio is desired.

Step 3: Obtain a concise decision rule. Lastly, the decision rule for the portfolio has to be computed. It is sufficient to predict the optimal algorithm given only one of the objectives, because the one objective will always decrease while the other one increases for biobjective Pareto fronts. This is a classification problem, where the remaining objective is the only feature and the optimal algorithm the target variable. A standard classification method can be used to produce a decision rule. Horn et al. (2017a) propose to use a pruned decision tree, where pruning controls the number of switches between optimal algorithms.

The method is implemented in an `R`-package available on github[2] and is used to analyze the results from the SVM study. In order to get more reliable results, the tuning is repeated ten times for four data sets. The MOSAP method is parametrized as follows: η is set to 0.5 in compliance with the goal of a good median Pareto front. The reference point for calculating the HV is set to $(1, 1.5)$ instead of $(1, 1)$ in order to give higher weight to solutions with low MMCEs. The weight vector *w* is set to $(0.05, 0.95)$, setting a higher weight to the quality of the portfolio without ignoring its size. The complexity parameter of the pruning is manually tuned to 0.1 in order to achieve an acceptable amount of switches between optimal algorithms. The selected portfolios and decision rules are displayed in Figure 4.3.

Although the portfolios with disabled downsampling (c.f. top row of Figure 4.3) differ quite a lot in their details, some common patterns can be observed: a) If present, LASVM or LIBSVM is used for solutions with the lowest MMCEs. b) SVMperf tends to take the lion's share of the Pareto front. c) If present, LLSVM is used for solutions with low training times. In the bottom row of Figure 4.3 the respective results with enabled downsampling are shown. Here LIBSVM dominates three of the four decision rules, while SVMperf gives better results on the protein data set. However, for this data set the combined Pareto fronts of the rules with and without downsampling are almost identical. Presumably there is not much redundancy in it and, hence, downsampling is not profitable.

4.3 Validating MOSAP rules

Although MOSAP should mainly be treated as a descriptive method, some insights on the performance of the presented method exist. On this account Horn et al. (2016c) present a simulator for

[2] `https://github.com/danielhorn/multicrit_result_test`

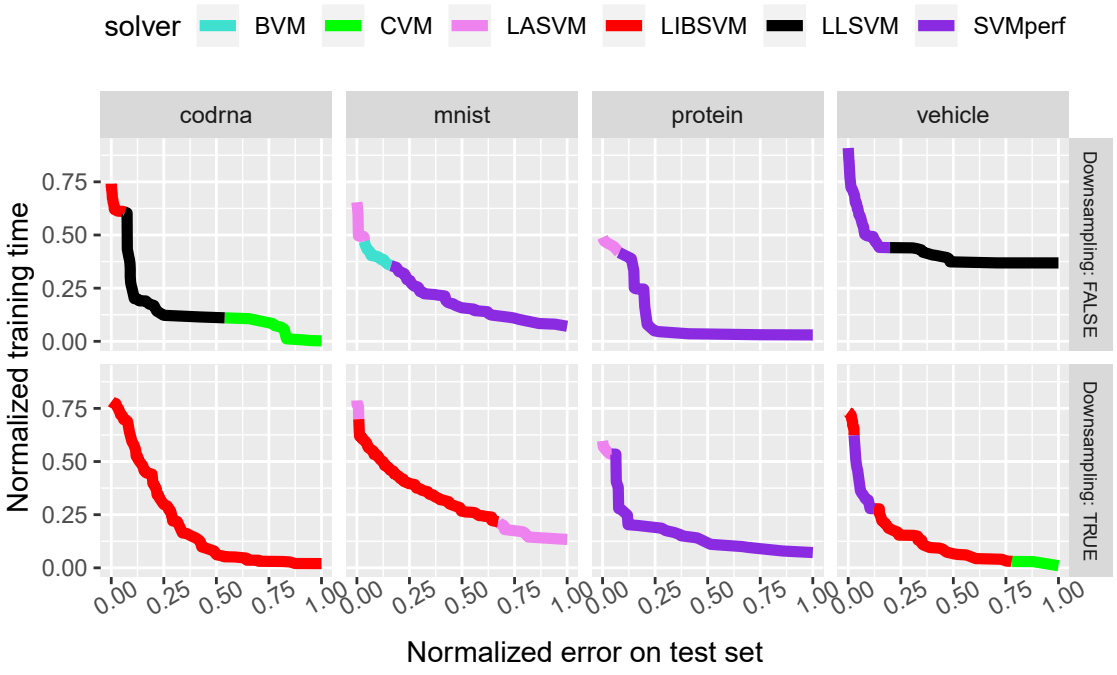


Figure 4.3: *Selected portfolios and decision rules for the SVM solvers on the four data sets.*

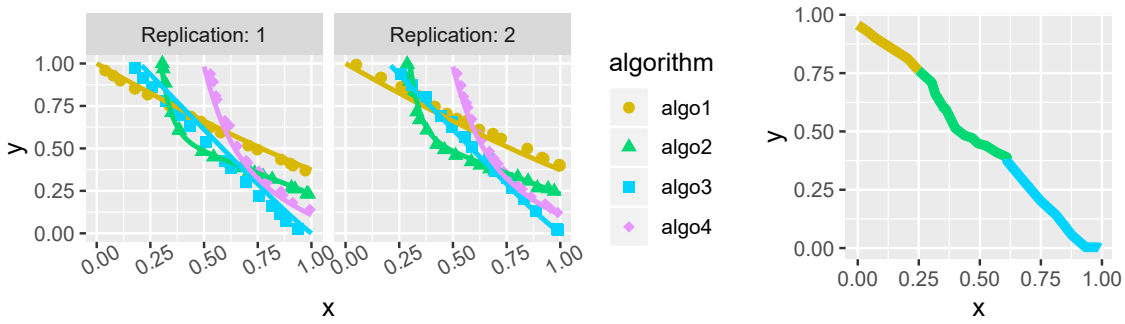
artificial MOSAP data sets with two objectives x and y. In these data sets the quality of the final MOSAP rule can be quantified since it is known which algorithm is optimal for which trade-off.

The simulator starts with sampling a true continuous Pareto front for each algorithm. Afterwards, discrete approximations of the true Pareto fronts are generated, whereat it is possible to control the number of discrete approximations (correlating to the number of tuning replications), the number of observations for each discrete approximation and the sampling procedure. In Figure 4.4 exemplary data of two discrete approximations with three optimal and one disturbing algorithm is displayed.

Furthermore, the simulator controls the values of x at which the optimal algorithm switches. In Figure 4.4, these switches occur at $x = 0.33$ and $x = 0.67$. Hence, an oracle function $p : [0,1] \to \mathscr{A}$ is available, giving the optimal algorithm for each value of x. A MOSAP rule can now be interpreted as an estimator $\hat{p}$ for this oracle. The z-value

$$z(p,\hat{p}) = \int_0^1 \mathbb{1}(x)_{\{p(x)=\hat{p}(x)\}} dx$$

gives the ratio of correct predicted algorithms and can be used as a performance indicator of $\hat{p}$. In Figure 4.4 (b), the z-value of the selected portfolio is 0.86. This rather low performance can be explained by the small number of just two replications.



(a) Simulated data

(b) Selected portfolio

Figure 4.4 *Exemplary simulated MOSAP data with three optimal and one disturbing algorithm. In plot (a), the simulated data is shown. The solid lines are identical in both figures and mark the true Pareto fronts, with split points between the three optimal algorithms at 0.33 and 0.67. The points differ between the figures and display the discrete approximations of the Pareto front. Plot (b) shows the corresponding selected portfolio with estimated switches at* $x = 0.25$ *and* $x = 0.61$.

Horn et al. (2016c) review the performance of their MOSAP method in a simulation study. For different parameter settings artificial data sets are generated and the z-value is calculated. Concentrating the results, the MOSAP method achieves adequate z-values, indicating that reasonable decision rules are computed. However, without competitor approaches available, these are just arguable indications and its true quality remains dubious.

4.4 Analyzing multiple data sets

Understanding the behavior of algorithms on single data sets is of great importance. However, a portfolio for these data sets is not required, since they have been extensively solved in the underlying hyperparameter tunings. Contrary, a portfolio is needed for data sets not used during its selection. As portfolios calculated on single data sets will not generalize well, the MOSAP method has to be extended towards multiple data sets.

Calculating a rule over multiple data sets can be dangerous, since the existence of a global rule is not guaranteed. In terms of the no free lunch theorem, the existence of a global rule is even impossible. However, if the considered data sets are similar enough, a joint decision rule may exist. Therefore, the extended MOSAP method should not only calculate the rule itself but also a measure indicating its validity within the considered data sets.

The MOSAP method can be adapted by extending each of the three steps from section 4.2 individually. A fourth step is introduced in order to measure whether the decision rule holds for all data sets.

Step 1: Remove dominated algorithms For each individual data set it is checked via the standard method, whether an algorithm would be removed. An algorithm is now removed globally, if the check is positive on more than $\nu \cdot r$ out of r data sets, where $\nu \in [0,1]$ is a new control parameter.

Step 2: Select the best algorithm portfolio This step remains mostly unchanged, only the quality of a subset of algorithms has to be redefined: The hypervolume difference is calculated for each individual data set and the mean difference is used as quality indicator.

Step 3: Obtain a concise decision rule Instead of estimating the decision tree on a single data set, it is calculated on the union of all r data sets.

Step 4: Measure the validity As validity measure, the accuracy of the decision tree estimated via r-fold crossvalidation is used. In each fold, the tree is trained on $r-1$ data sets and the error is estimated on the missing one. An accuracy of nearly 1 indicates a rule valid for all data sets, while lower values indicate heterogeneous data sets.

An updated version of the simulator can be used to validate these extensions. At first, an oracle function p, as described in section 4.3, has to be defined. Next, r data set specific oracles $p^{(i)}, i \in \{1,...,r\}$ are created, which can differ from the global oracle. Last, discrete approximations are generated for each data set specific oracle. In Table 4.4 nine types of dissimilarities between $p^{(i)}$ and p are presented. If type 1 is used, $p^{(i)} = p$ holds for all $i \in \{1,...,r\}$. Hence, the MOSAP method should find a decision rule that is valid for all data sets. For types 2–7 different sorts of noise are added to split points and algorithm order. However, the MOSAP method should still be able to recognize common patterns. For types 8 and 9 the MOSAP method should detect that no common decision rule exists, since the data specific rules are completely random.

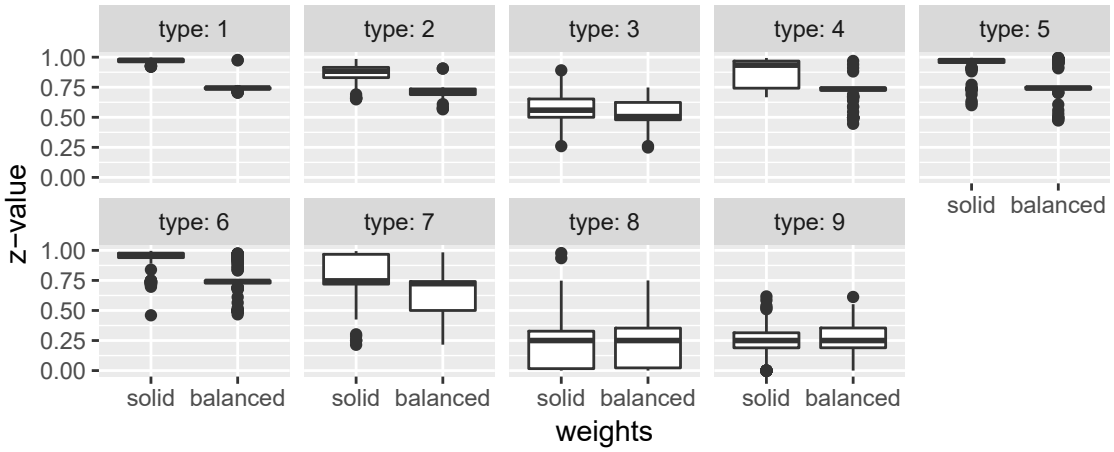
For all nine types 200 data situations with four optimal and two disturbing algorithms on four data sets are simulated. For each situation two portfolios are selected: one using weight vector $(0.001, 0.999)$ in order to select a large portfolio of high quality (the *solid* portfolio) and one using

Type	Explanation	Probability
1	No changes occur	–
2	Normal distributed noise added to split points	–
3	Split points are sampled from uniform distribution	–
4	One optimal algorithm is not present on the Pareto front	0.5
5	One disturbing algorithm is present on the Pareto front	0.5
6	One optimal and one disturbing algorithm are swapped	0.5
7	Two optimal algorithms are swapped	0.5
8	Algorithm order random	–
9	Split points and order random	–

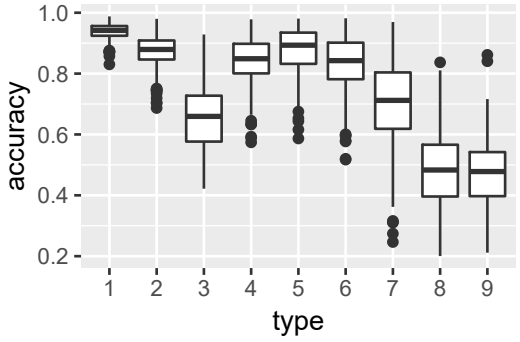
Table 4.4 *Nine different types how $f^{(i)}_{MOSAP}$ can differ from f_{MOSAP}. For types 4–7, the respective changes occur with the given probability for each data set.*

weight vector $(0.1, 0.9)$ aiming at a portfolio with a good trade-off between quantity and quality (the *balanced* portfolio).

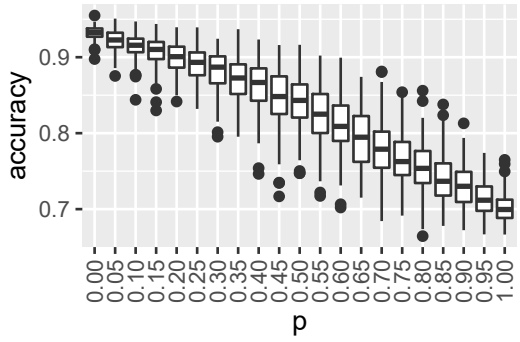
Figure 4.5 (a) shows the resulting z-values. For type 1, the solid portfolio nearly reaches best possible z-values of 1, while the balanced one only scores 0.75. This indicates that only three out of four optimal algorithms have been selected. However, since the balanced rule tries to trade off size and quality of the portfolio, it is allowed to omit some algorithms. This confirms that the MOSAP method can find reasonable decision rules. For types 2 and 4–6 the z-value deteriorates, but still reaches values close to 1 frequently. Hence, the MOSAP method is able to detect and deal with the respective noise. The z-values of types 3 and 7 are worse: For type 3 the split points are blurred so much that they can not be estimated from the data. For type 7, the possibility of swapping optimal algorithms increases the heterogeneity of the data sets a lot. As a consequence,



(a) z-values for all nine types for both the solid and the balanced portfolio.



(b) Accuracies for all nine types.



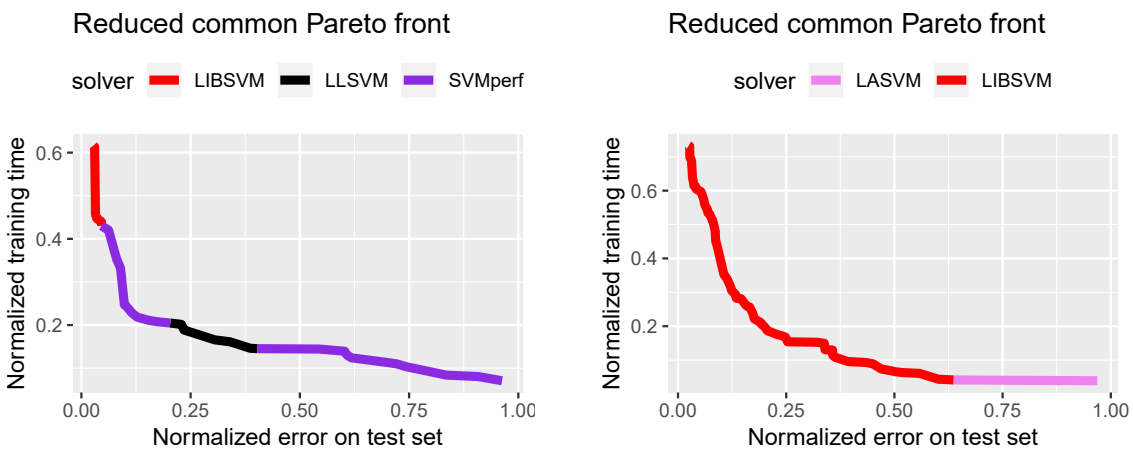
(c) Accuracies for type 6 with differing probabilities.

Figure 4.5 *Validation results for the extended MOSAP method. (a) Shows the z-values for all nine types (c.f. Table 4.4) for both the solid portfolio with a maximum number of algorithms and the balanced portfolio with an acceptable trade off between size and quality, (b) shows the associated accuracy values, (c) shows accuracy values for type 6 with differing probabilities.*

the solid portfolio often selects three algorithms, while the balanced one selects only two of them. Although not as good as for the other types, the selected rules still seem to be alright. As intended, no reasonable decision rules can be selected for types 8 and 9. Their z-values are not better than 0.25, corresponding to randomly guessing one out of four algorithms.

Overall, the MOSAP method is able to select a decision rule if the underlying data sets are similar enough. Step 4 of the method detects whether the selected rule holds for all considered data sets. In Figure 4.5 (b), the accuracy values of the simulations are shown. Its spread reaches from 0.95 for type 1 ($p^{(i)} = p$ is guaranteed) to 0.5 for types 8 and 9 ($p^{(i)}$ random). The other types score in between, while the more complicated types 3 and 7 are accounted with lower accuracies. For a more in-depth picture, Figure 4.5 (c) shows a more detailed study on type 6: The probability p of switching an active with an inactive algorithm, i.e. the amount of heterogeneity in the data, is varied from 0 to 1. The results of 200 replications show that the estimated accuracy drops while p increases. Hence, the accuracy measure fulfills its purpose.

Finally, the extended MOSAP method can be applied to analyze the SVM data from section 4.1. In Figure 4.6 portfolios and decision rules with disabled and enabled downsampling are shown. In both portfolios, LIBSVM is used to achieve solutions with best MMCE values. While with disabled downsampling, SVMperf and LLSVM are used to trade-off MMCE and runtime, with enabled downsampling the common Pareto front is be built by LIBSVM nearly alone. For both portfolios, the parameters of the MOSAP method were manually set in order to achieve high performing portfolios. Both portfolios seem to be consistent along the considered data sets, since the achieved accuracies (0.734 and 0.833 respectively) are sufficiently large. Altogether, the results from section 4.1 are confirmed.



(a) Downsampling disabled, accuracy: 0.734

(b) Downsampling enabled, accuracy: 0.833

Figure 4.6 *Portfolios and decision rules for the SVM study over all four data sets. Note that the values in the two graphics can not be compared to each other due to individual normalizations.*

4.5 Contributed publications

D. Horn, A. Demircioğlu, B. Bischl, T. Glasmachers, and C. Weihs. A comparative study on large scale kernelized support vector machines. *Advances in Data Analysis and Classification*, pages 1–17, 2016a.

C. Weihs, D. Horn, and B. Bischl. Big data classification: Aspects on many features and many observations. In *Analysis of Large and Complex Data*, pages 113–122. Springer International Publishing, 2016.

D. Horn, B. Bischl, A. Demircioglu, T. Glasmachers, T. Wagner, and C. Weihs. Multi-objective selection of algorithm portfolios. *Archives of Data Science, Series A (Online First)*, 2(1): 183–196, 2017a.

D. Horn, K. Schork, and T. Wagner. Multi-objective selection of algorithm portfolios: Experimental validation. In *International Conference on Parallel Problem Solving from Nature*, pages 421–430. Springer International Publishing, 2016c.

My work as a Ph.D. student started with the development and implementation of the SVM study. It required a lot of technical work by my colleague Aydın Demircioğlu and myself, including the installation and debugging of the SVM libraries on the LiDOng cluster. A descriptive analysis of the study's results was presented at the European Conference on Data Analysis (ECDA) 2014 in Bremen and published in Advances in Data Analysis and Classification (Horn et al., 2016a), some preliminary results have been published in Weihs et al. (2016).

However, the purely descriptive analysis of the results was quite dissatisfying. Since neither we nor any colleague we talked to were aware of any advanced method for further analyses, we had to develop one by ourselves. In the following months we worked on the development of the method for multi-objective selection of algorithm portfolios. It was implemented by myself in an `R` package and is available on the online platform github. In order to apply the MOSAP method in the SVM study, another set of experiments was performed, now with less data sets, but ten replications per data set. Each algorithm was tuned in two different setups: Once with disabled and once with enabled downsampling. The results of the MOSAP analysis were presented at the ECDA 2015 in Colchester and published in the Archives of Data Science (Horn et al., 2017a).

At first, MOSAP analyzed each data set on its own. Hence, the resulting algorithm portfolios were only valid on the respective data set. Since the results on the four data sets showed common patterns both in the case with and without downsampling, an aggregating MOSAP approach seemed promising. However, I feared that developing a new method to analyze some data based on the same data held the danger of overfitting and would not give any new insight. Moreover, I wanted to have at least some kind of validation of the new method. Therefore, I firstly developed a simulator for similar data situations (Horn et al., 2016c). Based on such artificial data situations, I developed the extension of MOSAP towards multiple data sets. The extension itself as well as its application to the SVM data have been presented at the ECDA 2018 in Paderborn.

Chapter 5

Conclusion and Future Work

This dissertation deals with the research areas optimization and machine learning. However, both of them are too extensive to be covered by a single person in a single work, and that is not the goal of this work either. Therefore, this work focuses on interactions between these fields.

On the one hand, most machine learning algorithms rely on optimization techniques. First, the training of a learner often implies an optimization. This is demonstrated by the SVM, where the weighted sum of the margin size and the sum of margin violations has to be optimized. Many other learners internally optimize either a least-squares or a maximum likelihood problem. Second, the performance of most machine learning algorithms depends on a set of hyper-parameters and an optimization has to be conducted in order to find the best performing model. Unfortunately, there is no globally accepted optimization algorithm for hyper-parameter tuning problems, and in practice naive algorithms like random or grid search are frequently used.

On the other hand, some optimization algorithms rely on machine learning models. They are called model-based optimization algorithms and are mostly used to solve expensive optimization problems. During the optimization, the model is iteratively refined and exploited. One of the most challenging tasks here is the choice of the model class. It has to be applicable to the particular parameter space of the OP and to be well suited for modeling the function's landscape.

In this work, special attention is given to the multi-objective case. In contrast to the single-objective case, where a single best solution is likely to exist, all possible trade-offs between the objectives have to be considered. Hence, not only a single best, but a set of best solutions exists, one for each trade-off. Although approaches for solving multi-objective problems differ from the corresponding approaches for single-objective problems in some parts, other parts can remain unchanged. This is shown for model-based multi-objective optimization algorithms.

The last third of this work addresses the field of algorithm selection. Here, an offline technique is used, i.e. the best algorithm is guessed a-priori to applying any algorithm. Again, the work focuses on the multi-objective case: An algorithm has to be selected with respect to multiple, conflicting objectives. As with all offline techniques, this selection rule hast to be trained on a set of available training data sets and can only be applied to new data sets that are similar enough to those in the training set.

This dissertation covers a wide range of rather different publications. It starts with a paper on the general model-based multi-objective optimization approach. Additional papers cover its extension towards multi-objective tuning problems. Furthermore, a new approach for multi-objective algorithm selection is presented in a series of papers. As an application, the multi-objective comparison of approximate SVM solvers is presented in another publication. Although the dissertation concludes with this chapter, the corresponding scientific work is certainly not finished and leaves space for future research projects.

- Model-based optimization is an open research area, and especially the combination of model-based and multi-objective optimization is relatively young. ParEGO, as its most prominent representative has been developed only 15 years ago. New algorithms and improvements to old ones are frequently published and *the best* algorithm has not been found yet.
- The adaption of the MBMO framework to the special needs of hyperparameter tuning has been accomplished in Horn and Bischl (2016). However, at the moment it is just a basic framework that allows future improvements. Moreover, this task is highly related to finding a *best* MBMO algorithm. In this context, special focus should be given to multi-point proposals, since tuning is a task that can easily be parallelized.
- In Horn et al. (2017b) hyperparameter tuning is interpreted as a stochastic optimization problem and this aspect is included in the MBMO framework. However, the results are quite unsatisfying and future work on this topic is required. Since very little work has been published even on the single-objective case, it should be approached at first. Re-evaluation strategies have to be collected, implemented in `mlrMBO` and compared to each other. Afterwards, the adaption to the multi-objective can be reconsidered.
- The development of the MOSAP method is nearly finished and the method itself fulfills its purposes. It can be extended by one additional step: Instead of just proposing an optimal algorithm for a given trade-off, it could also give a recommendation on the algorithm's hyperparameters. Apart from that, MOSAP can be used for other applications. One example is the analysis of benchmark results of optimization algorithms. In this context, the two objectives are the optimization time and the best found function value.

Some of these projects are running already. The work on improving Kriging kernels for hierarchical parameter spaces is ongoing and new publications are scheduled for the near future, including a contribution to the Genetic and Evolutionary Computation Conference (GECCO) 2019. Moreover, as the final aspects of the MOSAP method have been developed just recently, a final publication is planned, hopefully including some new applications. Further work on expanding `mlr`'s and `mlrMBO`'s capabilities in multi-objective hyperparameter tuning is targeted.

Bibliography

S. Bagheri, W. Konen, and T. Bäck. Online selection of surrogate models for constrained black-box optimization. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2016.

B. Bischl, S. Wessing, N. Bauer, K. Friedrichs, and C. Weihs. Moi-mbo: Multiobjective infill for parallel model-based optimization. In *Learning and Intelligent Optimization*, pages 173–186. Springer International Publishing, 2014.

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5, 2016.

B. Bischl, M. Lang, J. Bossek, D. Horn, J. Richter, and P. Kerschke. *ParamHelpers: Helpers for Parameters in Black-Box Optimization, Tuning and Machine Learning*, 2017a. URL `https://CRAN.R-project.org/package=ParamHelpers`. R package version 1.10.

B. Bischl, M. Lang, J. Bossek, D. Horn, J. Richter, and D. Surmann. *BBmisc: Miscellaneous Helper Functions for B. Bischl*, 2017b. URL `https://CRAN.R-project.org/package=BBmisc`. R package version 1.11.

B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang. mlrMBO: A modular framework for model-based optimization of expensive black-box functions. *arXiv preprint arXiv:1703.03373*, 2017c.

A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, 2005.

L. Bottou and C.-J. Lin. Support vector machine solvers. *Large scale kernel machines*, pages 301–320, 2007.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

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

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

C. Cortes and V. Vapnik. Support vector machine. *Machine learning*, 20(3):273–297, 1995.

K. Deb, R. Hussein, P. Roy, and G. Toscano. Classifying metamodeling methods for evolutionary multi-objective optimization: First results. In *Evolutionary Multi-Criterion Optimization*, pages 160–175. Springer International Publishing, 2017.

D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

M. Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2013.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large

linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

J. E. Fieldsend and R. M. Everson. The rolling tide evolutionary algorithm: A multiobjective optimizer for noisy optimization problems. *IEEE Transactions on Evolutionary Computation*, 19(1):103–117, 2015.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1): 3–42, 2006.

J. C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27(4): 857–871, 1971.

N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation*, 11(1):1–18, 2003.

N. Hansen, A. Auger, O. Mersmann, T. Tusar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *CoRR*, abs/1603.08785, 2016.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2009.

D. Horn and B. Bischl. Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–8. IEEE, 2016.

D. Horn, T. Wagner, D. Biermann, C. Weihs, and B. Bischl. Model-Based Multi-objective Optimization: Taxonomy, Multi-Point Proposal, Toolbox and Benchmark. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 64–78. Springer, 2015.

D. Horn, A. Demircioğlu, B. Bischl, T. Glasmachers, and C. Weihs. A comparative study on large scale kernelized support vector machines. *Advances in Data Analysis and Classification*, pages 1–17, 2016a.

D. Horn, A. Demircioğlu, B. Bischl, T. Glasmachers, and C. Weihs. Supplementary material to: A comparative study on large scale kernelized support vector machines. `http://largescalesvm.de/htmlplots/`, 2016b. Accessed: 2018-07-09.

D. Horn, K. Schork, and T. Wagner. Multi-objective selection of algorithm portfolios: Experimental validation. In *International Conference on Parallel Problem Solving from Nature*, pages 421–430. Springer International Publishing, 2016c.

D. Horn, B. Bischl, A. Demircioglu, T. Glasmachers, T. Wagner, and C. Weihs. Multi-objective selection of algorithm portfolios. *Archives of Data Science, Series A (Online First)*, 2(1):183–196, 2017a.

D. Horn, M. Dagge, X. Sun, and B. Bischl. First investigations on noisy model-based multi-objective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 298–313. Springer, 2017b.

D. Huang, T. T. Allen, W. I. Notz, and N. Zeng. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization*, 34(3):441–466, 2006.

F. Hutter and M. A. Osborne. A kernel for hierarchical parameter spaces. Technical Report arXiv:1310.5738, arXiv, 2013.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *Proceedings of LION-5*, pages 507–523, 2011.

T. Joachims and C.-N. J. Yu. Sparse kernel svms via cutting-plane training. *Machine Learning*, 76(2-3):179–193, 2009.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4), 1998.

J. Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10 (1):50–66, 2006.

H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press, 1951.

A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *ECONOMETRICA*, 28(3):497–520, 1960.

I. Loshchilov and T. Glasmachers. Black box optimization competition (bbcomp), 2018. URL `https://bbcomp.ini.rub.de`.

M. López-Ibáñez, L. Paquete, and T. Stützle. Exploratory analysis of stochastic local search algorithms in biobjective optimization. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 209–222. Springer, Berlin, Germany, 2010.

D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2017. URL `https://CRAN.R-project.org/package=e1071`. R package version 1.6-8.

J. Mockus, V. Tiesis, and A. Zilinskas. The application of bayesian methods for seeking the extremum. *Towards Global Optimisation*, (2):117–129, 1978.

D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, Inc., 2006.

J. Platt. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In *Advances in Kernel Methods - Support Vector Learning*, pages 185–208. MIT Press, 1998.

W. Ponweiser, T. Wagner, D. Biermann, and M. Vincze. Multiobjective optimization on a limited amount of evaluations using model-assisted $\mathscr{S}$-metric selection. In *Proc. 10th Int'l Conf. Parallel Problem Solving from Nature (PPSN)*, pages 784–794, 2008.

M. J. D. Powell. Radial basis functions in 1990. *Adv. Numer. Anal.*, 2, 1992.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018. URL `https://www.R-project.org/`.

M. Stein. Large sample properties of simulations using Latin hypercube sampling. *Technometrics*, 29(2):143–151, 1987.

I. Steponavičė, M. Shirazi-Manesh, R. J. Hyndman, K. Smith-Miles, and L. Villanova. *On Sampling Methods for Costly Multi-Objective Black-Box Optimization*, pages 273–296. Springer International Publishing, 2016.

M. Teplan. Fundamental of eeg measurement. *IEEE Measurement Science Review*, 2, 2002.

I. W. Tsang, J. T. Kwok, P.-M. Cheung, and N. Cristianini. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6(4), 2005.

I. W. Tsang, A. Kocsor, and J. T. Kwok. Simpler core vector machines with enclosing balls. In *Proceedings of the 24th international conference on Machine learning*, pages 911–918, 2007.

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15:49–60, 2013.

T. Wagner. *Planning and Multi-Objective Optimization of Manufacturing Processes by Means of Empirical Surrogate Models*. Vulkan Verlag, Essen, 2013.

C. Weihs, O. Mersmann, and U. Ligges. *Foundations of Statistical Algorithms*. Chapman & Hall/CRC, 2013.

C. Weihs, D. Horn, and B. Bischl. Big data classification: Aspects on many features and many observations. In *Analysis of Large and Complex Data*, pages 113–122. Springer International Publishing, 2016.

C. Weihs, S. Herbrandt, N. Bauer, K. Friedrichs, and D. Horn. Efficient global optimization: Motivation, variations and applications. *Archives of Data Science, Series A (Online First)*, 2(1): 3–28, 2017.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1), 1997.

N. Young. *An Introduction to Hilbert Space*. Cambridge University Press, 1990.

X. Yu and M. Gen. *Introducation to Evolutionary Algorithms*. Springer-Verlag London, 2015.

M. Zaefferer and D. Horn. A first analysis of kernels for kriging-based optimization in hierarchical search spaces. In *Parallel Problem Solving from Nature – PPSN XV*, pages 399–410, Cham, 2018. Springer International Publishing.

M. Zaefferer, T. Bartz-Beielstein, B. Naujoks, T. Wagner, and M. Emmerich. A case study on multi-criteria optimization of an event detection software under limited budgets. In *Proc. 7th Int'l. Conf. Evolutionary Multi-Criterion Optimization (EMO)*, pages 756–770, 2013.

K. Zhang, L. Lan, Z. Wang, and F. Moerchen. Scaling up kernel svm on limited resources: a low-rank linearization approach. In *International Conference on Artificial Intelligence and Statistics*, pages 1425–1434, 2012.

E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, 2000.

E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *Transactions on Evolutionary Computation*, 8(2):117–132, 2003.