Efficient Tuning in Supervised Machine Learning

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof.mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op dinsdag 29 oktober 2013
klokke 13:45 uur

door

Patrick Koch
geboren te Lippstadt
in 1982

Promotiecommissie

Promotor:	Prof. Dr. T.H.W. Bäck
	Prof. Dr. W. Konen (Cologne University of Applied Sciences)
Chairman:	Prof. Dr. J.N. Kok
Overige leden:	Prof. Dr. J. Branke (University of Warwick)
	Dr. W.A. Kosters
	Dr. M.T.M. Emmerich

This work was partially supported by the Bundesministerium für Bildung und Forschung (BMBF) under the grant "SOMA" (FKZ 17N1009), by the Kind-Steinmüller-Stiftung, and by the Cologne University of Applied Sciences under the research focus grant COSA.

Cover artwork: Keith Peters

ISBN: 978-94-6191-892-5

Contents

Chapter 1

Introduction

The tuning of learning algorithm parameters has become more and more important during the last years. With the fast growth of computational power and available memory databases have grown dramatically. This is very challenging for the tuning of parameters arising in machine learning, since the training can become very time-consuming for large datasets. For this reason efficient tuning methods are required, which are able to improve the predictions of the learning algorithms. In this thesis we incorporate model-assisted optimization techniques, for performing efficient optimization on noisy datasets with very limited budgets.

Under this umbrella we also combine learning algorithms with methods for feature construction and selection. We propose to integrate a variety of elements into the learning process. E.g., can tuning be helpful in learning tasks like time series regression using state-of-the-art machine learning algorithms? Are statistical methods capable to reduce noise effects? Can surrogate models like Kriging learn a reasonable mapping of the parameter landscape to the quality measures, or are they deteriorated by disturbing factors? Summarizing all these parts, we analyze if superior learning algorithms can be created, with a special focus on efficient runtimes.

Besides the advantages of systematic tuning approaches, we also highlight possible obstacles and issues of tuning. Different tuning methods are compared and the impact of their features are exposed. It is a goal of this work to give users insights into applying state-of-the-art learning algorithms profitably in practice.

1.1 Motivation

Learning is the art of acquiring knowledge. Because nature and its processes can be very complex, machines are often used to filter information and help humans in understanding the behaviour of such processes. Learning can be performed in many different ways: one possibility is to learn from experience, thus gaining knowledge by self-learning or trial-and-error. Another option is to gain knowledge by learning from a teacher. In that case a teacher must be available, knowing the correct target concept. This type of learning is

called *supervised learning*.

Many different learning algorithms exist, each being able to perform tasks like classification or regression. Although these algorithms very often perform well in practice, they also have certain parameters, which need to be set carefully to get reasonable results. A solution can be to use global optimization for minimizing the empiricial risk (or maximizing the performance of the learning algorithm). But as the evaluation of a learning algorithm includes a complete model training and prediction of unseen test patterns, the optimization can become expensive for larger datasets. Under such circumstances the question remains, whether the optimization can be performed within a feasible amount of time.

Supervised learning comprises the learning of a concept from labeled data [2, 42, 78, 77]. It has been studied intensively in the 20th century and today probably belongs to the most important research areas in Artificial Intelligence (AI) [208]. In other disciplines as computer science, supervised learning is also sometimes referred to as pattern recognition [21]. Supervised learning especially received a boost with the increase in computational power at the end of the 20th century. Buzzwords like "big data mining" have become popular in economy and industry. Learning from data usually includes a complicated process comprising the choice of a suitable learning algorithm and special pre- and post-processing operators. Here careful decisions must be made, which appears to be even harder, when very little domain knowledge is available for the task. Additionally, noise in the data causes a further blurring of the concept to be learned. Under such circumstances it is obvious, that finding good settings for learning algorithms becomes difficult. As a possible solution, state-of-the-art optimization heuristics can be incorporated, which can help users defining better learning processes. A decision support system which compares chains of operators and delivers the best combination in the end can help to solve this task.

For creating such a decision support system, we have to find out how learning is performed. The learning task in machine learning (ML) is very similar to learning of humans: a ML concept or model is usually generated by learning from examples — the training data. Afterwards, a learning algorithm can be applied to unseen data (unbiased evaluation or prediction) and if desired, it can then be refined by humans in a stepwise manner. Learning the target concept by using labeled information seems to be very attractive, because machines can process information very quickly and can repeat the learning as often as needed. However, when a good input-output mapping is difficult to obtain, because many data points have to be processed, the refinement process can be very time-consuming and humans tend to bias personal favourite choices. Then, instead of this human intervention other approaches promise to be more systematic. These processes are known as *model optimization* or *model tuning*. Here optimization algorithms are considered, which test different configurations of learning algorithms, and combinations of pre-processing methods.

Although this works well very often, the model optimization must be discussed in terms

of its usability and effectiveness. In general, learning algorithms are only helpful when they do not have any biases to specific data and the time required to create them is acceptable. Model optimization is often not incorporated in practice, because it is too

- time-consuming, or
- error-prone to biases and wrong model assumptions caused by high noise levels.

These two topics are important points for all learning algorithms and we will address both issues in this thesis. It has to be analyzed how tuning of learning algorithms can be justified against no tuning at all, or a simple hand-tuning of learning algorithms.

1.2 Survey

This thesis is structured as follows: methods for ML and preliminaries of stochastic optimization are given in Chapter 2.

In Chapter 3 we add techniques for feature processing to the learning process and show how beneficial they are for the prediction accuracy of learning algorithms.

Learning algorithms are systematically coupled with optimization techniques in Chapter 4. The demand for parameter tuning is motivated here, by comparing baseline tuning algorithms with state-of-the-art optimization techniques. As a new field of research we apply model-assisted optimization to improve the performance of learning algorithms on noisy data with very restrictive budgets.

In Chapter 5 we especially focus on the efficiency of the tuning, which is one important aspect to make ML tuning ready-to-use for real-world applications. We apply sub-sampling techniques, combined with special noise handling strategies for surrogate models, and use repeated evaluations to handle noisy evaluations by the optimal computing budget allocation (OCBA).

The quality of the fit of parameter spaces is analyzed in Chapter 6. This chapter is especially interesting to reveal the weaknesses of certain surrogate models. Here, we show, that high noise levels can deteriorate the fitness landscapes of these models, requiring additional noise handling approaches to avoid misleading settings during the tuning.

A new, different perspective of tuning in ML is described in Chapter 7, where ML tuning is transferred to a multi-criteria task. Although multi-criteria optimization has been analyzed for engineering applications earlier, it has never been considered for ML optimization.

To the best of our knowledge we conclude the thesis with a summary and discussion in Chapter 8.

Chapter 2

Methods

In this chapter we introduce methods and operators which are necessary elements for the experiments in this thesis. We start with a brief introduction to machine learning in Sec. 2.1 and proceed with more detailed descriptions of feature selection and feature construction in Sec. 2.2 and Sec. 2.3 respectively. Finally we give an overview about methods of Computational Intelligence in Sec. 2.4 and present the state-of-the-art of model-assisted optimization in Sec. 2.5.

2.1 Machine learning

Machine learning is the process of gaining knowledge using computers without programming the knowledge explicitly. In general two different types of learning can be distinguished:

- *Supervised learning*: direct training feedback is available in form of examples for which the output is known.
- *Unsupervised learning*: training is indirectly done, e.g., by performing self-learning methods such as trial-and-error.

In this thesis the focus lies on supervised learning, nevertheless parts of the research can also be applied to unsupervised learning.

2.1.1 Supervised learning

Supervised learning belongs to the most widely researched fields in artificial intelligence. The goal is to train a computer model which is capable to predict new data.

In supervised learning a mapping from input data to a target variable is trained by using examples. The target variable can be represented by discrete values (or classes), whereas we call this a classification task. The target can also be a continuous variable, and in this case we call the learning task *regression*. As a third special case we will also investigate learning tasks that have a time dependency, which we refer to as *time series* problems.

2.1.1.1 Learning from data

The data for machine learning is usually stored inside a file or a database. The entries of this file or database are called *instances* or *patterns*. A dataset for ML can be written as a sequence of pairs $D = (\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_n, y_n)$, consisting of input data or feature vectors $\vec{x}^{(i)}$ and their corresponding outputs $y^{(i)}$. Each input $\vec{x}^{(i)}$ consists of N attributes describing the characteristics of the instance. Later we will also denote the input attributes as *features*. In supervised learning, the algorithms require outputs to train a prediction model. E.g., in classification they require at least one instance of each class in the training data. For *classification* tasks the target variables y_i are defined by a finite set with a fixed number of elements, the corresponding classes. In *regression*, the values y_i are defined by a continuous space, e.g. $y_i \subseteq \mathbb{R}$.

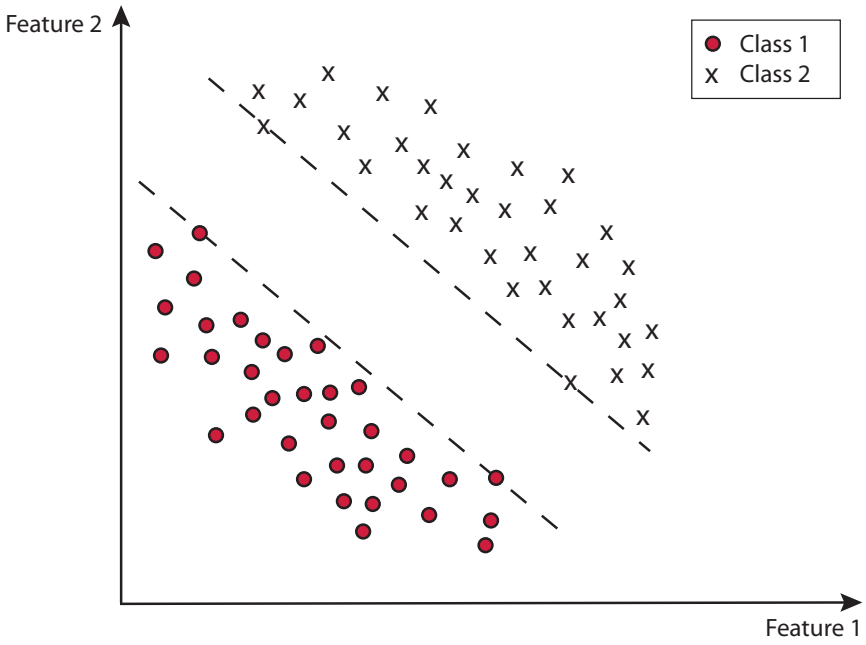


Figure 2.1: Linear separability of a binary classification problem. The figure shows data points (instances or pattern) in a two-dimensional feature space. The instances are marked by red points and black crosses for class 1 and 2 respectively. The two dashed lines illustrate the margin of possible separations of the classes. A sophisticated learning algorithm for classification would try to find a separating line with maximum margin to both class instances.

Figure 2.2: Poisonous or edible? The mushroom Amanita muscaria or "Fly agaric toadstool". Picture by Tony Wills, License: Creative Commons Attribution 3.0 Unported

2.1.1.2 Classification

In classification we are interested in finding a separating hyperplane in the N-dimensional feature space, which is able to classify each pattern to its belonging target label. E.g., in Fig. 2.1 a two-class example is shown, where the class patterns are displayed as crosses and circles. The two dashed lines represent possible margins for each class. Every hyperplane between these two margin lines can separate the class patterns by classifying all points to the left side as circles and the points on the right side as crosses. An early learning algorithm that makes use of a separating hyperplane is the perceptron invented by Rosenblatt [207], which later led to the concept of Artificial Neural Networks (ANN) [109].

An intuitive example for classification is to classify mushrooms. Mushrooms can be either *edible* or *poisonous*. If we write down examples for mushrooms, including attributes like colour, size, points, etc., and denote the class label for each example, we can train a learning algorithm which is capable to classify mushrooms. The learning algorithm would use the known examples for predicting new unknown examples into edible or poisonous. Of course, the probability of classifying new samples, increases, if a certain number of training patterns is available, and under the assumption that the training data is correct.

2.1.1.3 Regression

In classification the output was considered as a discrete and finite variable. However, in many applications it is desired to predict numerical values like real numbers, e.g., to predict sensor measurements in an engineering process. In this case we perform *regression*, instead of classification.

Although regression is a different approach, real-valued output functions can also be learned by most algorithms which are proposed for classification. An easy to understand

model for regression is the *linear model*:

$$y_i = w_0 + w_1 x_{i,1} + w_2 x_{i,2} + \ldots + w_N x_{i,N} + \varepsilon_i \tag{2.1}$$

The linear model predicts the target by learning the optimal weighted sum of the input features. The weights $\vec{w} = w_1, ..., w_N$ are determined by an optimization procedure minimizing the error of the training set. The weight w_0 is just an offset and the term ε_i acts as *slack variable*. The weights for the model are based on the training data and can be determined by minimizing a loss function like the mean squared error (MSE). For predictions $\vec{\hat{y}}$ and real observations $\vec{y}$ the MSE for a training set of size n is defined as follows:

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{2.2}$$

In practice often the root mean squared error is considered:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \tag{2.3}$$

It is possible to use interpretable models like the linear model in Eq. 2.1 to reveal feature importance. The disadvantage is that linear models are limited to easier data. More powerful models with the opportunity for interpretability are, e.g., multivariate regression splines (MARS) [87], or other techniques for symbolic regression like Genetic Programming [158].

2.1.1.4 Time series analysis

In many engineering applications data is dependent on a *time* label, which represents the information when the data was recorded. In history time series have been known at least since the 10th century (cf. Fig. 2.3). A well-known example of a time series application is the prediction of stock market prices. Here, the prices have an additional time information. Thus, the price of a stock is defined by its value at time t. Outgoing from time t the price of a stock can rise (hausse) or fall (baisse). It is important to note, that the price in the future $t + k$ is often dependent on time t and also on outer events from the environment which occurred in the period from $t + k$ up to some earlier time $t - \ell$. Of course the predictability in the future is usually limited to a certain k, because the stock market situation of today will only have a marginal influence to the situation in the next day or hour. But it is assumed that the time series can be predicted in the future up to a certain k. The learning algorithms can gain from adding influences from outside, e.g., actual news. The price of the stock can change according to the information given in such additional reports. The other assumption in time series analysis is *autoregression*, e.g., that the price will depend on the change of earlier values. This corresponds with time series like weather forecasts, where it can be assumed, that the weather in the next minute will be probably similar as before. Thus,

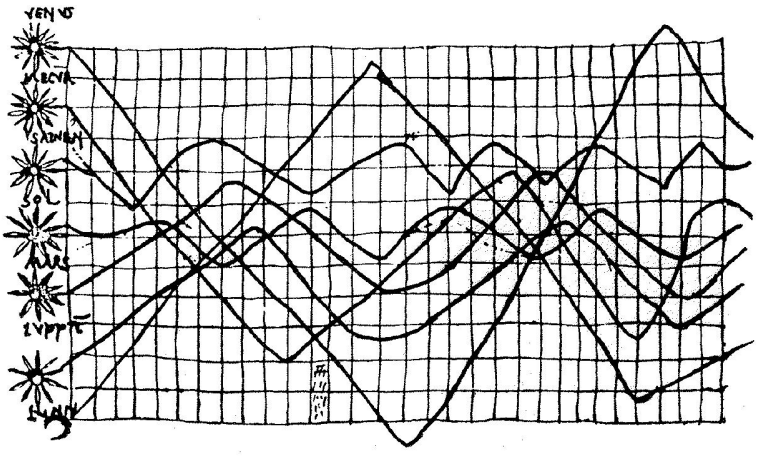


Figure 2.3: The figure shows a historic drawing of an early time series in the 10th century. Drawing found in [170, 241].

output values of time series data are correlated to a) the values of the time in the past and b) changes or actions in the underlying domain.

Usually a time series is recorded as a discrete stochastic process from time $T = 0$ to t:

$$(\vec{x}^{(0)}, y^{(0)}), (\vec{x}^{(1)}, y^{(1)}), \ldots, (\vec{x}^{(t)}, y^{(t)}) \tag{2.4}$$

In the simplest case the output is equal to the input $\vec{x}^{(i)} = y^{(i)}$, for all $i = 0, \ldots, t$. Then the time series can be simply written as

$$\vec{y} = y^{(0)}, y^{(1)}, \ldots, y^{(t)} \tag{2.5}$$

The time series in Eq. Eq. 2.5 is called a univariate time series. Here, the time series is solely defined by the probabilistic structure (the mean and variance) of the samples of y. In this case, classical time series analysis methods like ARIMA or GARCH processes can be used [31]. However, these methods are limited, because they are suited for linear processes and also assume a certain autocorrelation of the target. Instead, regression techniques from ML can be applied, to detect more difficult structures in the time series.

Today there is a large demand for analyzing multivariate time series. Here, at any time point t more than one variable is recorded. All variables have a certain influence on the target variable, which is in general unknown for arbitrary tasks.

2.1.2 Learning algorithms

The state-of-the-art in ML is difficult to grasp. Many algorithms exist, capable for many purposes, so that a certain favourite can hardly be named here. However, at least two algorithms showed good performances on a variety of tasks, and they can even be improved

by setting their parameters: random forest [26] as representative of decision tree-based methods and Support Vector Machines (SVMs) [245]. In this section we define the learning algorithms, a more detailed description on the underlying hyperparameters is then given in Sec. 4.2.

2.1.2.1 Random forest

Random forest (RF) [26] is a learning algorithm based on classification and regression trees (CART) [28]. RF uses an ensemble of decision or regression trees for the prediction. For ensemble methods we use a number of independent learning algorithms, which are later combined by an aggregation procedure. The goal is to get a better result using the ensemble, compared with possible weak results of single learning algorithms itself [102]. E.g., the CART approach by Breiman [28] has weaknesses, because the single trees are prone to overfitting. With the aggregation of many trees in an ensemble, such weaknesses of single learning algorithms are compensated. The most prominent two methods for ensemble learning are boosting [212] and bagging [24], which are described in the following sections.

RF uses the bagging approach (see Sec. 2.1.2.2) to create the tree ensemble. Additionally, a random feature selection for each node of the trees is taken from Ho [112, 113] and Amit and Geman [3]. These advantages of tree learners leads to a very robust algorithm, which is very stable to issues like overfitting or noise in the data.

Although RF is almost parameter-free, and already works well with default settings, it is sometimes advantageous, to optimize certain parameters [218]. As important parameters for RF, the number of trees (*ntree*) in the ensemble can be mentioned [232], as well as the number of splits performed in each tree node (*mtry*). Instead of performing a tuning of *mtry*, Liaw and Wiener [166] suggest the following defaults based on the number of features p:

$$mtry_{default} = \begin{cases} \max(\lfloor \frac{p}{3} \rfloor, 1), & \text{if task is regression} \\ \sqrt{p}, & \text{else} \end{cases} \tag{2.6}$$

It has to be noted, that this formula is only a rule of thumb. In our experiments we will also see advantages of tuning *mtry*. Other tuning options include the computation times of the algorithm, e.g., by setting the *sampsize* parameter (sample size used for the RF trees).

2.1.2.2 Ensemble approaches

Bagging

Bagging [24] is an acronym for *bootstrap aggregating*, where bootstrapping refers to a resampling method [67]. In bootstrapping k learning or training subsets of the training set size n are drawn repeatedly random with replacement from the complete training data. An ensemble learning algorithm would now train k prediction models (e.g., CART) for each

subset. For the prediction all models in the ensemble predict the test pattern. For the final output, the k predicted outputs of the models are *aggregated* by an aggregation function. As aggregation function, e.g., majority voting [190] can be used, or one can take the mean of all ensemble learners in regression.

Boosting

Boosting was presented by Schapire [211] and has been later improved by Freund and Schapire [86]. In boosting, the complete training data is used for the training, but a weight is assigned for each training pattern. At first the weights are uniformly distributed for all patterns, and then updated during the algorithm run. The weighting scheme is defined by variables $w_t(i)$, which denote the weight of pattern $\vec{x}^{(i)}$ in iteration t of the algorithm. In each iteration training patterns are sampled using $w_t(i)$ as probability distribution for the sampling with replacement. After having drawn the training data according to the weighting scheme all ensemble learners are trained using these sets. Then the weights are updated, assigning a smaller weight for correctly predicted patterns, and an increased weight for wrongly predicted patterns respectively. In the next round the sampling and training is applied again until a termination condition holds. Possible termination criteria are, e.g., a limited number of model trainings or a time limit.

When the target variable has a certain number of wrong values, the prediction accuracy degenerates for boosting, as shown by Dietterich [58]. For this reason bagging is used within RF.

2.1.2.3 Support Vector Machines

Support Vector Machines (SVMs) have been originally introduced as learning algorithms for binary classification and regression tasks [215], but they can also be used to solve multi-class problems, e.g., by incorporating multiple "one-against-all classifiers" (see [23] for a comprehensive overview).

Our main intention for using SVMs — besides that SVMs are known as a powerful ML tool — is that SVMs tend to be very sensitive to parameter settings, which makes them interesting for parameter tuning tasks. In binary classification, SVMs seek the maximal margin classifier, which best separates the two classes. In the simplest case this means to search for the optimal separating hyperplane by minimizing the empirical risk. However, if SVMs could only classify separable data, the applicability would be very limited, since most real-world problems are not linearly separable. Therefore, SVMs perform classification with the following extensions:

(1) Kernel-induced feature space: the input space is implicitly mapped to a higher-

dimensional space using a kernel function

$$K(\vec{X}, \vec{Z}) = \langle \phi(\vec{X}) \cdot \phi(\vec{Z}) \rangle \tag{2.7}$$

where ϕ defines a mapping from the input space. A simple example for such a transformation is to calculate the monomials of the input features and to consider them for the mapping. The kernel function denotes the similarity of two observations $\vec{X}$ and $\vec{Z}$. Because the kernel function can be interpreted as a dot product in a high-dimensional space, the computation is feasible also for very high dimensions. When the patterns cannot be separated by a linear classifier in the original feature space, this can be still possible in the kernel-induced feature space. The kernel function can be defined anew for each task, or pre-defined kernel functions can be chosen. It is important that kernel functions fulfill the Mercer theorem [180], as they have to be positive definite and symmetric (PSD property).

The most frequently used kernel functions for SVMs include the following functions:

- Linear

$$K(\vec{X}, \vec{Z}) = \langle \vec{X}, \vec{Z} \rangle \tag{2.8}$$

- Polynomial kernel:

$$K(\vec{X}, \vec{Z}) = \langle \vec{X}, \vec{Z} \rangle^d \tag{2.9}$$

- Radial basis function (RBF):

$$K(\vec{X}, \vec{Z}) = \exp\left(-\frac{||\vec{X} - \vec{Z}||^2}{2\gamma^2} \right) \tag{2.10}$$

where γ and d are parameters for the corresponding kernel functions.

(2) Regularized risk minimization: If some observations are still not classified correct in the kernel-induced feature space, SVMs can make use of the soft-margin concept by Cortes and Vapnik [51]. In soft-margin SVMs a regularization term is introduced, which penalizes wrongly classified patterns. Making this more rigorous, we can write the optimal SVMs classification model in the unconstrained dual form, denoting H as the reproducing kernel Hilbert space for the kernel function $K(\cdot, \cdot)$, in the following optimization problem:

$$\hat{F} = \arg \inf_{F \in H, b \in \mathbb{R}} ||F||_H^2 + C \sum_{i=1}^{n} L(Y_i, F(\vec{X}_i) + b) \tag{2.11}$$

Here F is the real-valued target function, and $\hat{F}$ the regularized target function. In case

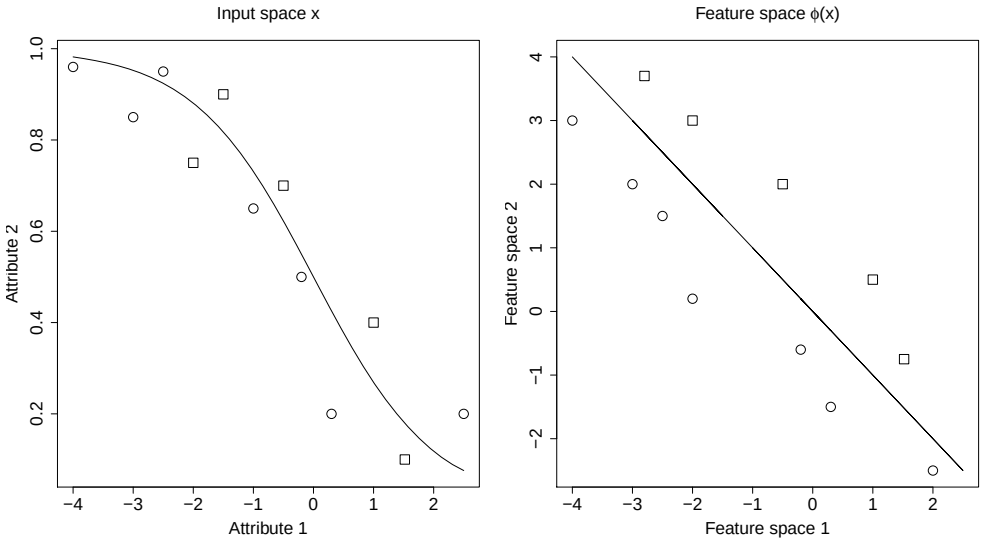


Figure 2.4: Mapping from input to feature space via kernel function ϕ. Before the mapping, the data is not separable, after the mapping a linear classifier can separate the class patterns.

of binary classification we would discretize the real-valued output of $\hat{F}$ by mapping it to $\{-1, 1\}$. The first term is called a smoothness penalty using a regulariser $||\cdot||_H^2$, where H is the Hilbert space of functions defined over the input domain [52, p. 40]. The 2-norm can be used to penalize non-smooth functions. The second term measures the closeness of our predictions to the true outputs by means of a loss function. In classification, we usually select the hinge loss $L(Y,t) = L_h(Y,t) = \max(0, 1 - Yt)$ for an intended output $t = \pm 1$ and a classifier score Y. For regression we often set $L(Y,t) = L_\epsilon(Y,t) = \max(0, |Y - t| - \epsilon)$ to the ϵ-insensitive loss. The hinge loss is a convex, upper surrogate loss for the 0/1-loss (which is of primary interest, but algorithmically intractable), while L_ϵ provides the estimation of the median of Y given $\vec{X}$. Both losses lead to quadratic programming problems for Eq. 2.11, which can be solved efficiently, and the non-differentiability of these two loss functions further provides for sparse solutions [43]. The two terms are balanced by the parameter C, sometimes also referred to as *Cost*. In recent SVM implementations, a value of $C = 1$ is taken as default, equally weighting loss function and smoothness penalty.[1]

The optimal kernel function may vary depending on the data. Without having any prior knowledge, the RBF kernel works well in most cases, presuming that the kernel parameters are set to good values. Of course the RBF and polynomial kernel functions are more complex

[1] In the *R* implementations contained in the *e1071* package [59] and in the *kernlab* package [136], the default value for C is set to 1.

and are able to classify non-separable data better than a linear kernel. But as a drawback they are more expensive to compute.

In regression, usually the Support Vector Regression (SVR) approach of Drucker *et al.* [63] is applied. For more information about SVR the interested reader is referred to the tutorial of Smola and Schlkopf [228]. For simplicity we do not distinguish between SVR and SVM in this thesis, but indicate which method is used where it is necessary.

A recent overview about the state-of-the-art in learning with kernels has been given by Signoretto and Suykens [221].

2.1.3 Generalization and benchmarking

In ML experiments a dataset

$$\mathcal{D} = \left((\vec{x}^{(1)}, y^{(1)}), (\vec{x}^{(2)}, y^{(2)}), \ldots, (\vec{x}^{(m)}, y^{(m)}) \right) \tag{2.12}$$

consists of m examples or instances and the corresponding output, e.g., a class label or real value.

If all training patterns in the data $\mathcal{D}$ are consistent, the optimal learning algorithm would always give the correct answer for every new example. Of course this is a very strong assumption for most datasets, and even when this holds, it remains unclear, if the learning algorithm also behaves optimal on future instances. Thus, in practice approximations of this optimal model are computed. In order to compare these models, we have to define a procedure, under which we can measure the model quality. This is known as benchmarking of ML algorithms.

Benchmarking of learning algorithms belongs to one of the most controversely discussed topics in ML and statistics: Hothorn *et al.* [116] describe a theoretical framework for benchmarking. Eugster and Leisch [73] and Eugster *et al.* [72] present techniques for visualizing benchmark results. Hornik and Meyer [114] propose to use consensus rankings for benchmarking ML algorithms. Bischl *et al.* [20] discuss resampling strategies for tuning of learning algorithms and describe common pitfalls and advantages of the methods.

In experiments it is important to take into account statistical analyses to provide sound results. Public benchmark sets like the UCI repository [84] are available and can be downloaded from the internet. There are at least two reasons for using public datasets: evaluation with public datasets makes the results of algorithms reproducible, because every researcher is able to test his method on these datasets. Second, researchers can help in improving methods when they are applied to common datasets.

However, we also think that public benchmark sets alone might be not sufficient for a sound benchmarking of algorithms and can result in overfitting to certain benchmark repositories itself. Besides that, the motivation behind developing new algorithms is to solve new problems. Here, public benchmark datasets might be too restrictive. Instead we think that with a combination of public benchmark sets and real-world data better progress can

be made. Learning algorithms are also thought to find hidden structures in unknown data, and if benchmarks are too restrictive, it can happen that in the sequel the development of learning algorithms stagnates.

In the following paragraphs, we introduce methods for evaluating ML algorithms. A desirable property of ML algorithms is, that they can generalize well and perform good on other unseen data. Therefore, special resampling strategies exist, which play a key role for all experiments made in this thesis.

2.1.3.1 Sampling strategies

ML models should be well generalizing, to learn a model for the dataset. The learning algorithm should be able to generate predictions for new patterns $\vec{x}^{(o)}$ with a maximal accuracy. In this section sampling methods are proposed, which aim at estimating the *generalization error*. These sampling methods include:

- Random sampling (holdout set)
- k-fold cross-validation
- Leave-one out cross-validation
- Sub-sampling
- Stratified sampling

Random sampling

For training and testing purposes the dataset $\mathcal{D}$ is usually split randomly into disjoint subsets

$$\mathcal{L} \subset \mathcal{D} \tag{2.13}$$

and

$$\mathcal{T} \subset \mathcal{D} \tag{2.14}$$

with $\mathcal{L} \cup \mathcal{T} = \mathcal{D}$. Now each learning algorithm can be trained on the training data $\mathcal{L}$, while it can be evaluated on test data $\mathcal{T}$. This accuracy estimation method is called *holdout method*, because the patterns of set $\mathcal{T}$ are held out during training.

Cross-validation

Another method for estimating the generalization performance is the k-fold cross validation (CV). Here the data is split into k disjoint subsets. For each subset $\mathcal{D}(i), i \in$

$\{1, \ldots, k\}$ a prediction model is trained, predicting the remaining patterns

$$\mathcal{D}(j) = \mathcal{D} \backslash \mathcal{D}(i) \tag{2.15}$$

Finally the errors of each learned model can be aggregated, e.g., by taking the mean of the errors.

Leave-one out cross-validation

Leave-one out cross-validation (LOOCV) is a special case of k-fold cross-validation, where the number k is equal to the number of patterns of the dataset ($k = |\mathcal{D}|$). LOOCV certainly gives the best estimate of the real generalization performance. Unfortunately LOOCV belongs to the most expensive methods to evaluate learning algorithms, and for this reason it is not well suited when many models must be built, as is the case in parameter tuning of learning algorithms.

Sub-sampling

Normally all available training patterns in the data set $\mathcal{D}$ are used for training. Sometimes $\mathcal{D}$ tends to be very large, and the training takes a lot of time. Additionally, outliers and redundant patterns can be misleading for learning the concept. In such cases, we can perform a training using only small subsets (subsamples) of the available training data. This sampling strategy is called *sub-sampling* and is very effective, when it is repeatedly performed, e.g., in a tuning process.

Learning with reduced training set sizes can give remarkable speed-ups for the model-building process, which is the main reason for using it in practice. Last [161] proposes partial learning by projective sampling to estimate the optimal training set size. For Artificial Neural Networks [38] and Support Vector Machines [220], intelligent sub-sampling near the decision boundary helps to speed-up the training process without loss of accuracy. In many studies it was observed that training with smaller sets does not necessarily lead to worse generalizing prediction models. Instead, the complexity of the trained models can be reduced as shown by Oates and Jensen [187] and Provost *et al.* [196]. Some people think that simpler hypotheses should be preferred, which was constituted in Occam's razor [22].

Stratified sampling

When performing sub-sampling, it can happen that the training set sizes tend to be very small, which is especially the case for smaller datasets. Another situation where this issue can occur is data, where the classes are imbalanced. This can lead to training sets, where whole classes are missing. Stratified sampling [186] is a sampling strategy for classification to solve this issue. Similar to other random sampling strategies, stratified sampling draws patterns

at random, but draws them from each *stratum* or class. By doing this the distribution of the class probabilities remains unchanged.

2.2 Feature selection

Feature processing probably belongs to one of the most important steps for obtaining better prediction models. In this section we introduce various methods for feature subset selection, which are capable to form better feature starting sets. It is shown in [255], that although the dataset itself does not contain more information with a reduced feature subset, the selection can be beneficial for the generalization performance. Although this does not seem to be very logical at first sight, the reason is that unimportant features in the dataset can deteriorate results. With a better feature set, noise is reduced in the data and a better classification or regression model can be obtained. Due to the high importance of feature selection, we include hyperparameters of feature processing algorithms in the tuning, in order to generate more powerful learning algorithms.

Feature subset selection can be of large importance, because the model benefits from a reduced feature subset, where non-informative features are excluded. Guyon and Elliseeff [101] present a comprehensive overview about this topic. In general we can distinguish between three different types of feature selection:

- *Filter approaches* give rankings of the features without training a learning algorithm
- *Wrapper approaches* return feature subsets by applying a specific learning algorithm in an iterated manner
- *Embedded methods* are integrated into some learning algorithms, which enables off-the-shelf feature rankings

All approaches have their advantages and disadvantages, which are briefly described in the following sections.

2.2.1 Filter approaches

Feature selection using filter methods is solely based on criteria without requiring to build and to evaluate a specific learning algorithm. For this reason these methods are very attractive, because building learning algorithms for large-scale data can become computationally expensive, and filter methods always provide a quick alternative for measuring the information content of a variable then.

2.2.1.1 Correlation-based filtering

A well-known measure to determine the similarity of two variables is the linear correlation. For two variables $\vec{x}$ and $\vec{y}$ the correlation is defined by

$$\rho = \frac{\sum_{i=1}^{m}(x_i - \mu(x_i)) \cdot (y_i - \mu(y_i))}{\sum_{i=1}^{m}\sqrt{(x_i - \mu(x_i))^2} \cdot \sqrt{(y_i - \mu(y_i))^2}} \tag{2.16}$$

where $\mu(x_i)$ and $\mu(y_i)$ are the mean values of variable $\vec{x}_i$ and $\vec{y}_i$ respectively.

The output ρ lies in the range between -1 and 1, indicating a negative or positive correlation between the two variables.

2.2.1.2 Entropy and information gain

Instead of the linear correlation measure, often the entropy is been used for measuring the information content of a variable $\vec{x}$. The entropy is defined by

$$H(\vec{x}) = \sum_{i=1}^{m} Prob(x_i)\log(Prob(1/x_i)) \tag{2.17}$$

Now we can define the following quantity for measuring the entropy of $\vec{x}$ after $\vec{y}$ was observed:

$$H(\vec{x}|\vec{y}) = -\sum_{j=1}^{m} Prob(y_j)\sum_{i=1}^{m} Prob(x_i|y_j)\log_2(Prob(x_i|y_j)) \tag{2.18}$$

where $Prob(x_i|y_j)$ is the posterior probability of $\vec{x}$ under $\vec{y}$.

As an alternative to the entropy, Quinlan [197] introduced the information gain, which is based on the entropy and became popular as splitting rule in the C4.5 learning algorithm [197]:

$$IG(\vec{x}|\vec{y}) = H(\vec{x}) - H(\vec{x}|\vec{y}) \tag{2.19}$$

2.2.1.3 Random forest importance

Random forest includes a method for ranking features, which is called *random forest importance* (RFI). RFI cannot be classified as a filter approach, because it exhibits several differences to the other approaches mentioned so far, but it is not a wrapper either. Instead it can be categorized as an *embedded method*, because although RF is used for determining a feature ranking, the ranking is rather *built-in* the learning algorithm [101]. For this reason a prediction model has to be learned only once to obtain a feature ranking and not multiple times like in wrapper approaches.

The advantage of the RFI method is that it can detect variable interactions, which the other filter approaches were not able to. The idea is as follows [27]: permute a variable or feature by looking how much the prediction error increases when all other variables remain the same. Two measures are considered for the RFI:

- the mean decrease in accuracy and

- the mean decrease in Gini index [92] describing the node impurity measure [25, 169] which is often used in tree-based classifiers.

These measures are obtained by performing the permutation test: When a certain feature V_j is permuted, the out-of-bag (OOB) performance or the Gini index changes. The difference between this performance and the prior performance is calculated for all trees. The average of the tree values then describes the RFI measure for feature V_j.

2.2.2 Wrapper approaches

In wrapper approaches the learning algorithm is used to perform a prediction of the task given a certain feature subset. In general words a search in the feature subset space is performed, using the prediction error as objective function value to be minimized. Well-known examples are exhaustive search (which is NP-hard, making the runtime intractable in high dimensions unless $P = NP$), Genetic Algorithms or feature forward selection and backward elimination. Kohavi and John [152] give a comprehensive survey of wrapper approaches. We describe the most often used approaches in the following sections.

2.2.2.1 Feature forward selection

Feature forward selection is an iterative procedure where one feature is selected in the beginning. Using this single feature a model training is performed. After evaluating the model, the accuracy on the evaluation set is taken as feature importance. In the first iteration the algorithm would use all features to build N prediction models. The feature with the best prediction accuracy is then selected as most important feature. Outgoing from this feature all other features are added and again N models are built. The procedure is iterated, until no more improvement with adding new features can be made.

The disadvantages of the forward selection method are obvious: first, if the number of features of the dataset is large, many model trainings must be performed to obtain a feature subset. This gets even more expensive, when a large fraction of the features is advantageous for the learning algorithm. Secondly, the method cannot detect complex variable interactions. Let us assume that features V_1 and V_2 together are perfectly suited to predict the target variable. However, feature V_1 and V_2 both do not give any improved prediction when considered exclusively. This means, that the method would converge and both features V_1 and V_2 would never be added to the feature subset.

2.2.2.2 Genetic feature selection

Another heuristic for feature selection is the classical Genetic Algorithm (GA) [98]. A set or population of binary strings of length N (corresponding to the number of features) is initialized first. A '1' at position i in the binary string indicates, that the feature is selected, while a '0' would mean to exclude the feature.

Note, that number of possible feature sets increases exponentially with the number of available features/attributes (2^N). For this reason a genetic search can become computationally expensive. Another point is that empty feature sets must be avoided, e.g., a string full of zeros would lead to crashes for most learning algorithms. Therefore in most implementations of genetic feature selection a parameter constant is added, denoting the number of features to be used.

2.3 Feature construction

In this section we describe the creation of new features by performing specific transformations or projections. We show how these transformations can be learned semi-automatically providing only a set of basis functions. Although datasets sometimes already contain attributes which are good descriptors of the target, it can be helpful to derive new features using the original attributes. These can be *projections* or other transformations performed on the data. Well-known data transformation methods are, e.g., the Principal Component Analysis (PCA), or logarithmic and Fourier (spectral) transformations. Another possibility is to calculate monomials of the input attributes of degree d.

In some cases features can be derived by recommendations of experts. However, this requires of course a detailed knowledge about the properties of the data. For this reason we propose to use methods, which require only little knowledge, and which can be applied to most numeric datasets without any prior information. In Sec. 3.1 we show how user-based feature construction can be combined with model-assisted tuning for improving prediction models. Another feature processing method which has received only little attention so far is the Slow Feature Analysis. Combined with a simple classifier it can be used as a state-of-the-art learning algorithm, producing almost as good results as a SVM and RF, but in much shorter time. Otherwise the main purpose is to use it as a pre-processing method for other learning algorithms. We applied SFA to a gesture recognition problem and achieved remarkable results.

2.3.1 Principal Component Analysis

Principal Component Analysis (PCA) was firstly proposed by Pearson [189] and later formalized and named by Hotelling [115]. The goal of PCA is to find the directions in the data which have the highest variance. It is a classical projection method for learning with reduced feature sets. PCA transforms the coordinate system by determining the maximum variance of the input dimensions. PCA is helpful when attributes are correlated. In such cases PCA transformations can help to reduce the total number of features, without losing too much information.

In the application of PCA, in a first step, the data is mean-centered. Afterwards, the covariance matrix is calculated for the points. The PCA then diagonalizes the covariance

matrix to obtain the eigenvectors. Then, the eigenvectors are sorted in order of decreasing eigenvalues. Note that the eigenvalues can represent the data, because they inherit the variances of the samples in the dataset. Eigenvalues are also often used in image calculations, since transformations can easily be made when the structure is available. Now we can use them for reducing the number of features of the data. With fewer features based on the largest eigenvalues of the covariance matrix we can often achieve better predictions for the task.

Formally PCA can be defined as follows: the covariance matrix of the data is denoted by $\vec{C}$. Now the eigenvectors of this matrix are determined, and sorted in order of decreasing eigenvalues. The matrix of the eigenvectors is written as $\vec{U}$. Now we simply calculate the product of $\vec{U}$ and $\vec{C}$:

$$\vec{Y} = \vec{U}^T \vec{C} \tag{2.20}$$

The output matrix $\vec{Y}$ comprises the transformed feature space, from which we can now select the first d rows, to reduce the original m-dimensional feature set to $d < m$ principal component features. The more components are selected, the less variance of the data can be observed in the transformed space. Because PCA is affected by scaling, all attributes are at first standardized to zero mean and unit variance. Instead of using PCA, sometimes the Singular Value Decomposition (SVD) proposed by Golub and Van Loan [100] is considered, since it is more robust in determining the eigenvectors, especially when the covariance matrix is singular.

The most relevant drawbacks of PCA and comparable methods are, that they are only suited to make transformations of real-valued attributes. If discrete attributes are present in the data, PCA is not applicable any more. A frequently used approach is then to remove these features from the dataset and to apply PCA only to the numerical part of the data.

From our perspective PCA is a part of the learning process and therefore should be also incorporated into the tuning. Although no direct parameter is required for PCA, the number of features d to be selected from the transformed feature space, can be seen as a hyperparameter.

2.3.2 Monomials

Monomials are products of attributes or features of degree d. Assume we have three features denoted by a, b, c in the dataset. The set of all possible monomials of degree 2 would then be

$$F := \{a^2, b^2, c^2, ab, ac, bc\} \tag{2.21}$$

It can be valuable to calculate such feature sets and use them instead of only the basis attributes. The reason is the higher-dimensional space the data is projected in. The main disadvantage is, that monomials of higher degrees, or monomials for a large-scale dataset are

expensive to calculate. For this reason monomials are usually only calculated for the most important features. The importance of the features can be determined by using a feature ranking, e.g., through a filter, or by PCA through the level of variance of the principal components.

2.3.3 Slow Feature Analysis

Slow Feature Analysis (SFA) is a learning algorithm from neuroscience which is capable of learning unsupervised new features or "concepts" from time series. SFA was originally developed in context of unsupervised learning of learning invariances in the visual system of vertebrates [252]. In [253] and [254] a detailed overview about the algorithm is given. Although SFA is inspired from neuroscience, it does not have the drawbacks of conventional Artificial Neural Networks (ANNs) such as long training times or strong dependencies on initial conditions. Instead, SFA is fast in training and it has the potential to find hidden features out of multidimensional signals, as shown by [16] for handwritten-digit recognition.

SFA is optimally suited to construct features for time series signals. The original SFA approach for time series analysis is defined as follows: For a (multivariate) time series signal $\vec{x}(t)$ where t indicates time, find the set of real-valued output functions $g_1(\vec{x}), g_2(\vec{x}), ..., g_M(\vec{x})$, such that each output function

$$y_j(t) = g_j(\vec{x}(t)) \tag{2.22}$$

minimally changes in time[2]:

$$\Delta y_j(t) = \langle \dot{y_j}^2 \rangle_t \text{ is minimal} \tag{2.23}$$

The Δ-value can be described by measuring the slowness of an output signal as the time average of its squared derivative [254]. To exclude trivial solutions we add some constraints:

$$\langle y_j \rangle_t = 0 \text{ (zero mean)} \tag{2.24}$$

$$\langle y_j^2 \rangle_t = 1 \text{ (unit variance)} \tag{2.25}$$

$$\langle y_k y_j \rangle_t = 0 \text{ (decorrelation for } k > j) \tag{2.26}$$

The third equation is only relevant from the second slow signal on to prevent higher signals from learning features already represented by slower signals.

For arbitrary functions this problem is difficult to solve, but SFA tries to find a solution by expanding the input signal into a nonlinear function space by applying certain basis functions, e.g., monomials of degree d. This expanded signal is sphered to fulfill the

[2] $\langle \cdot \rangle_t$ means average over time and $\dot{y}$ indicates the time derivative.

constraints of Eq. 2.24, Eq. 2.25 and Eq. 2.26. Then SFA calculates the time derivative of the sphered expanded signal and determines from its covariance matrix the normalized eigenvector with the smallest eigenvalue. Finally the sphered expanded signal is projected onto this eigenvector to obtain the slowest output signal $y_1(t)$.

Berkes [16] extended this approach to classify a set of handwritten digits. The main idea of this extension is to create many small time series out of the class patterns: let us assume that for a K-class problem each class $c_m \in \{c_1, ..., c_K\}$ has got N_m patterns. We then reformulate the Δ-objective function (2.23) for SFA with distinct indices k and l as the mean of the difference over all possible pairs:

$$\Delta(y_j) = \frac{1}{n_{pair}} \cdot \sum_{m=1}^{N_m} \sum_{\ell=k+1}^{N_m} \left(g_j(p_k^{(m)}) - g_j(p_\ell^{(m)}) \right)^2 \tag{2.27}$$

where n_{pair} denotes the total count of all pairs and $p_k^{(m)}$ and $p_l^{(m)}$ represent the k-th and l-th class pattern of class m. The constraints defined by Eq. 2.24, Eq. 2.25 and Eq. 2.26 can be reformulated then by substituting the average over time with the average over all patterns, such that the learned functions have a zero mean, unit variance and are decorrelated [16].

As shown by Berkes [16], the $(K-1)$ slowest SFA output signals are expected to have a low intra-class variation, but usually a high inter-class variation. Therefore Berkes [16] proposes to train a standard Gaussian classifier on the slowest $(K-1)$ SFA outputs produced from the training records. The Gaussian classifier will seek an optimal position and shape of a Gauss function for each class in this $(K-1)$-dimensional space. The class probabilities of patterns $\vec{x}$ are then defined by the posterior probabilities according to the Bayes decision rule.

2.3.4 Genetic Programming for feature processing

Genetic Programming (GP) [158] is a technique, which discloses a large variety of usage (see Sec. 2.4.4). GP can be applied to construct features, by learning non-linear combinations of the basis features. In contrast to monomials, GP has a much higher complexity of the search space. In earlier approaches, Krawiec [159] used GP for feature construction to build better features from the original feature set. Krawiec discovered that good features could be constructed using GP, but he also observed a remarkable overfitting to the training data. Likewise Smith and Bull [224] used GP for constructing new features, and combine the construction with a Genetic Algorithm (GA) for feature selection. They achieved improved results in 8 of 10 datasets compared with C4.5 [197], but also observed the problem of overfitting in some cases. Therefore they provided a reordering strategy, which enables better generalization performance again. We think that the large degree of freedom of GP can be an advantage, but can also be a great disadvantage at the same time. It is difficult to define good parameters for GP, and other issues like the large runtime caused by the wrapper approach and the large search space remain. Besides this, the overfitting problem must be handled. For this reason the search for better feature sets with GP is promising, but can become elusively slow, which can make it inefficient for practical use.

2.4 Stochastic optimization

Nonlinear optimization problems arise in many fields like engineering, mathematics or computer science. In contrast to linear optimization problems, nonlinear optimization problems are usually solved with heuristics.

2.4.1 Formulation of the problem

In a search space $S \subseteq \mathbb{R}^m$ we seek for the best solution $\vec{x}^* \in S$ of an evaluation function f:

$$f(\vec{x}) = f(x_1, x_2, ..., x_m) \tag{2.28}$$

The search space S can be of continuous type, that is $S \subseteq \mathbb{R}^m$, but can also contain discrete parts, or can be restricted by bounds.

In general, we can assume that the underlying problem is a minimization problem, that is we are seeking a point where the function value of f is minimal (if this exists):

$$f(\vec{x}) \to \min \tag{2.29}$$

Note, that maximization problems can be re-formulated to minimization problems, but without loss of generality minimization problems are considered in this thesis:

$$\max f(\vec{x}) = -\min(-f(\vec{x})) \tag{2.30}$$

Another point is that the parameters for learning algorithms are usually constrained, meaning that we need to respect at least box-contraints in the form of lower ($l\vec{B} \in S$) and upper bounds ($u\vec{B} \in S$), for the components of $\vec{x}$:

$$\begin{aligned} lB_1 \leq \quad & x_1 \quad && \leq uB_1 \\ lB_2 \leq \quad & x_2 \quad && \leq uB_2 \\ & \vdots \\ lB_m \leq \quad & x_m \quad && \leq uB_m \end{aligned} \tag{2.31}$$

In the following sections we describe approaches for solving such optimization problems. It has to be remarked, that our objective is to solve the optimization problems *globally*, that is we want to find a *best* solution, i.e., the vector producing the minimal value of f.

2.4.2 Local search

Local search comprises methods for stochastic optimization by refining a candidate solution. A well-known example is the classical steepest descent, which moves along the gradient of the function in a sequential process. Other methods for local search have been proposed, e.g., the Newton method, which uses the inverse Hessian matrix for a better estimate of the best search direction. However, as we are performing black-box optimization where no analytic solution of the objective function exists, it is not possible to calculate derivatives analytically.

Instead, direct search methods can be used which perform a random stochastic search, and thereby try to approximate the gradient or the Hessian matrix. An example of such a heuristic is the algorithm of Broyden, Fletcher, Goldfarb and Shanno (BFGS) [32, 81, 99, 219], which approximates Newton's method, but without requiring second-order derivatives to guide the search.

The BFGS algorithm is based on the method by Davidon [54] and Fletcher and Powell (DFP) [81], where new points are determined by deriving information from the previous search steps:

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{s}^{(k)}\vec{v}^{(k)} \tag{2.32}$$

where $\vec{s}^{(k)}$ is the step-size and $\vec{v}^{(k)}$ is an approximation of the inverse Hessian matrix.

Algorithms like BFGS are local search methods, i.e., they are performing a search using a starting point and guiding the search to a optimum. In the best case the global optimum is reached, but it can happen that the algorithm converges to a near local optimum. A simple strategy for finding the global optimum with local search methods is to perform *restarts*, that is initiating multiple local searches with different starting points. However, it might be the case that this restarting strategy is not successful either.

2.4.3 Evolution Strategies

Evolution Strategies (ES) are search heuristics inspired by biological evolution. They perform as well biologically-inspired variation as selection to guide the search to the optimal solution. One of the main advantages of ES is, that they don't require any additional information like gradient information. ES can be used for global optimization, at least when a set of solutions (a population) is initiated for approximating the global optimum. ES were firstly developed by Rechenberg [203] and Schwefel [217] in the 1960s.

The essential ingredient of today's state-of-the-art ES is a completely derandomized adaptation of the mutation step-sizes. This step-size adaptation was first-time established in the well-known Covariance-Matrix-Adaptation ES [107, 108]. Many extensions have been proposed for CMA-ES, including strategies for uncertainty-handling [105, 106], or mirrored sampling [30] which can improve the original algorithm. In our experiments with ES we use the CMA-ES by Hansen [104], but are aware of the fact that other variants can profitably support the CMA-ES. E.g., although the CMA-ES was originally proposed as a local optimization strategy in [107], we also use it as a global optimizer with larger population size and a uniformly but random initialization strategy based on Latin hypercube sampling (cf. Sec. 2.5.2). Instead of that, Auger and Hansen [4] have established the increasing population sizes (IPOP) strategy for global optimization on multimodal landscapes.

2.4.4 Genetic Programming

Genetic Programming (GP) can be seen as a substantial part of Evolutionary Algorithms (EAs). It was originally proposed for the automatic generation of computer programs [7, 158, 193]. Today, it has especially received interest in applications of *symbolic regression*. The goal of symbolic regression is to find a functional relationship between given input and measured output signals. Thereby it would be equivalent to regression, but in symbolic regression a symbolic representation of the functional relationship (e.g., by a mathematical expression) is returned. State-of-the-art is Pareto GP [227] which enables to determine functional relationships of varying complexities. Starting with a high-level problem definition, GP creates a population of random symbolic expressions, termed *individuals*, that are progressively refined through an evolutionary process of variation and selection until a satisfactory solution is found.

Although GP requires no prior knowledge about the solution structure it can be difficult to apply it to functions, where certain structures are required. The goal of GP is to minimize the error of a given task, which is usually defined by a *fitness function* like in ES. An inherent advantage of GP is the representation of solutions as symbolic expressions, i.e., as terms of a formal language, which makes them accessible to human reasoning and symbolic computation. The main drawback of GP is its high computational complexity due to the potentially infinitely large search space of symbolic expressions.

For applying GP, several problem specific and algorithm specific parameters have to be specified:

Fitness function

A fitness function associates a numerical fitness value to a candidate solution represented as a symbolic expression. This function encodes the task to be solved. GP is an optimization algorithm in the sense that it searches for solutions that (by convention) minimize this fitness function.

Symbolic expressions

Any GP function consists of function symbols, constant symbols, and variable symbols, used for constructing symbolic expressions. Together with the variation operators, these building blocks define the structure of the GP solution search space.

Initialization strategy

The *initialization strategy* defines how the initial GP population is generated. Often complete randomized strategies are considered for this, but we want to bias our search to

more simple individuals. Therefore a strategy that grows individuals to a random tree depth less then or equal to a maximum tree depth given as a parameter is employed [158].

Variation operators

Like in ES, variation operators are methods for mutating and recombining existing solutions. Because the implementation of these operators is highly dependent on the solution representation (tree, graph, etc.), a variety of different operators have been developed. Still, the classical mutation and crossover operators originally proposed by Koza often work well in practice and are used in a type-safe manner [158, 193].

General EA parameters

The remaining parameters are common to most Evolutionary Algorithms and include among others population size, selection strategy, and termination criteria. Most generic extensions to Evolutionary Algorithms, such as niching and automatic restarts, can be directly applied to GP.

2.4.5 Other search spaces

Besides the investigation of continuous parameter spaces (e.g., in Evolution Strategies or local search methods), other search space types are possible. Examples of such parameters are integer or discrete values. Schwefel [217] invented an ES for integer search spaces with a binomially distributed mutation operator. Compound representations with real-valued, integer and discrete attributes can be solved with a special formulation of ES as proposed by Bäck and Schwefel [6]. Various applications using this ES formulation can be found in [5, 70, 165]. It has to be noted, that discrete and integer parameters can not simply be considered as continuous values, but the variation operators can be adopted, and special mutation distributions can be used inside this mixed-integer ES formulation.

2.5 Model-assisted optimization

It is often desired to find solutions of nonlinear optimization problems under very limited budgets. While optimization heuristics like ES often require many function evaluations until they converge in an optimum, an alternative can be to perform the main part of the optimization on a surrogate model. This is what we call *model-assisted optimization*. The idea of model-assisted optimization is, that the evaluations of the surrogate model are very cheap, while the real function might be expensive. For this reason the main part of the optimization can be performed on the surrogate function, while the real objective function must be only considered for re-fitting of the surrogate model and for validation purposes.

2.5.1 Related work

In the Evolutionary Computation (EC) field global optimization problems are precisely solved by techniques as presented in Sec. 2.4 like the CMA-ES by Hansen and Ostermaier [107] or Differential Evolution (DE) by Storn and Price [231]. As both strategies often require a lot of function evaluations, we will describe strategies which are especially suited to solve optimization problems with very restrictive budgets. Nevertheless the strategies of EC are well understood and still remain important, because global optimization on the easier surrogate function can finally be performed using these strategies.

Jones *et al.* [133] presented the efficient global optimization (EGO) algorithm. This method is especially designed to solve very expensive functions which frequently occur in industry. EGO makes use of Kriging [160, 201] as a surrogate model, and the expected improvement (EI) criterion, which are both important concepts and are described in detail later in this thesis. Other frameworks for parameter optimization include the Relevance Estimation and Value Calibration (REVAC) by Nannen and Eiben [184]. REVAC was developed to determine robust parameter settings for evolutionary algorithms. While REVAC was mainly proposed to find robust parameters, Smit and Eiben [222, 223] extended the method with other heuristics to reduce the computation times. Birattari *et al.* [17] invented the F-Race algorithm based on the racing algorithm by Maron and Moore [173]. The algorithm ranks solutions by statistical tests for steering the search more directly. Birattari *et al.* [17] tested their algorithm especially for combinatorial optimization problems with good performances.

Bartz-Beielstein *et al.* [11] invented the sequential parameter optimization (SPO), which combines methods from classical Design of Experiments (DoE) [76] and Design and Analysis of Computer Experiments (DACE) [209]. EGO by Jones *et al.* [133] and SPO share that they are both model-assisted iterated optimizers, which means that they train and refine a model for saving evaluations on the real objective function f. EGO and SPO have several parallels, but also differ in the flexible number of repeated evaluations and the choice of the surrogate model in SPO. In Section 2.5.3 we give a brief overview of the SPO algorithm.

Based on the work of SPO and Racing [173], Hutter *et al.* [123] started their research with a variant called iterated local search (ILS), and later extended their work in [122] where they compare a variant of SPO which is evaluated on several deterministic test functions. The main change is to use log-transformations of the response values to give better estimates for the surrogate model. This idea has also been discussed by Wagner and Wessing [250] where they compare various response transformations for EGO. In a recent work Hutter *et al.* [121] propose the sequential model-assisted algorithm configuration (SMAC) which also incorporates Kriging and RF surrogate models.

The main difference between the existing model-assisted approaches and other algorithms for stochastic optimization is, that model-assisted optimization aims at keeping the number

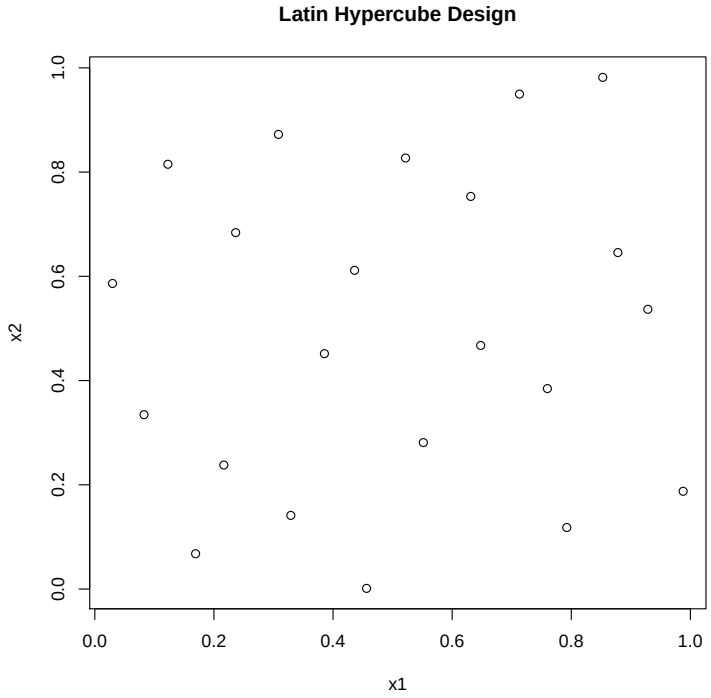


Figure 2.5: The Figure shows an optimal latin hypercube design of size 22 in a two-dimensional feature space with bounds $[0, 1]$ for both features $x1$ and $x2$.

of function evaluations relatively low. Besides that, sometimes also more about the sensitivity or importance of the parameters can be learned. This can support later analyses like the landscape analysis described in Chapter 6.

2.5.2 Design of Experiments

Design of Experiments (DoE) and here especially Latin hypercube sampling [175] can be seen as a starting point for model-assisted optimization. For exploration of the search space, a random uniform distribution of the design points is created. The number of points must be given, which dictates the coverage of the search space. However, in contrast to random sampling, the distribution of the design points most likely will lead to a better exploration of the search space, at least for small dimensions.

In classical Design of Experiments (DoE) [76] often systematic sampling is applied, to give good exploration of the search space. E.g., in Latin hypercube sampling (LHS) [175] a restricted search space $S \subseteq \mathbb{R}^m$ is assumed, having box-constraints as described in Eq. 2.31. When very few parameters have to be set, a simple grid search or LHS can be very effective, assuming that the optimum does not lie in narrow and steep valleys. But when the dimension

increases, these methods usually have difficulties due to the *curse of dimensionality* [172].

2.5.3 Sequential parameter optimization

In this section we describe the sequential parameter optimization (SPO) approach, where an arbitrary surrogate function is trained to enable cheaper and faster optimization runs. SPO was originally designed as a framework for improving the performance of optimization algorithms by experimentation. But SPO can also be used as an optimizer, which is also our main domain in this thesis. The SPO research started with the analysis of stochastic search algorithms [13], including *Evolution Strategies* (ES) and *Simulated Annealing* (SA) [143] to give a better understanding about the behaviour of these algorithms.

SPO combines methods from classical *Design of Experiments* (DoE) [76] and modern *Design and Analysis of Computer Experiments* (DACE) [209]. The optimization loop is visualized in figure 2.6. DoE dominates the first phase of the SPO development. This approach includes several established and well-understood procedures for the analysis of deterministic and stochastic data, especially regression and analysis of variance techniques.

In the first phase of SPO, an exploration of the parameter search space is performed. Therefore, a set of initial design points $\vec{x}$ is generated by a sampling strategy of the user's choice. Usually a space-filling design $DES := (\vec{x}^{(1)}, \vec{x}^{(2)}, ..., \vec{x}^{(l)})$ like a Latin hypercube sample of size $\ell = |DES| \in \mathbb{N}^+$ is created. All ℓ initial design points are then evaluated at least once on the real objective function f resulting in a vector of responses $(y_1, y_2, ..., y_\ell)$. SPO now uses the initial design points DES and their responses y for fitting a surrogate function. In general, any regression function or surrogate model can be selected therefore, but often Kriging is chosen, since it is almost parameter-free and can handle most landscapes. We will introduce this technique in more detail in Sec. 2.5.4.

The idea is now to sequentially refine this model by performing a pre-defined number of function evaluations. The prior information can be used for better exploration of the search space and to improve the surrogate models. The surrogate model serves as a cheap alternative for evaluations on the true objective function and a much larger number of evaluations can be performed to produce new design points, which in turn are evaluated by the algorithm and used to update the model. As surrogate model often Kriging is used, because it can nicely model non-linear functions with only few available design points. However, SPO does not require a specific model and provides an interface for setting other surrogate models like RF or linear models.

SPO distinguishes itself from other surrogate modeling approaches by the following components:

- refinement of the surrogate model by establishing additional design points,

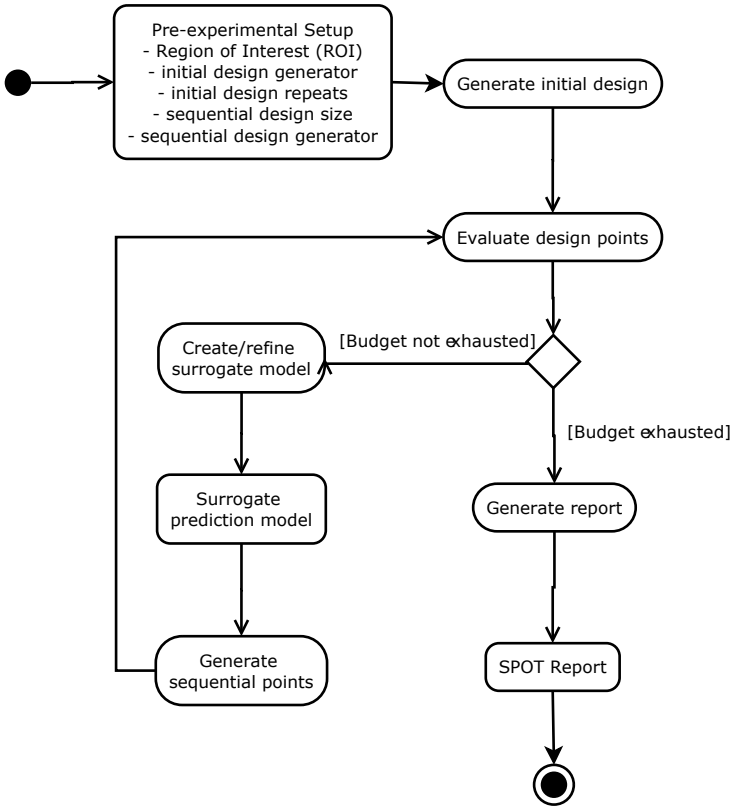


Figure 2.6: The sequential parameter optimization process.

Algorithm 1: Sequential parameter optimization (SPO)

```
// phase 1, building the model:
```
let A be the algorithm we want to tune;
generate an initial design $DES = \left(\vec{x}^{(1)}, \ldots, \vec{x}^{(n)}\right)$ of n parameter vectors;
let $k = k_0$ be the initial number of replications for determining estimated responses;
foreach $\vec{x} \in DES$ **do**
    run A with $\vec{x}$ k times to determine the estimated response y of $\vec{x}$;
end
```
// phase 2, using and improving the model:
```
while *budget not exhausted* **do**
    let $\bar{\vec{x}}$ denote the parameter vector from DES with best estimated response $\bar{y}$;
    let k be the number of repeats already computed for $\bar{\vec{x}}$;
    build surrogate model $Y(\vec{x})$ based on DES and $\left(y^{(1)}, \ldots, y^{(|DES|)}\right)$;
    optimize the model with respect to some utility function;
    thus produce a set DES' of d new parameter vectors;
    // improve confidence
    run A with $\bar{\vec{x}}$ once and recalculate its estimated mean response using all $k+1$ test results; let $k = k + 1$;
    run A k times with each $\vec{x} \in DES'$ to determine the estimated mean response;
    extend the design by $DES = DES \cup DES'$;
end

- repeated evaluations of design points using a dynamic number of performed evaluations (OCBA) and an aggregation function (e.g., mean) to receive a comparable quality measurement.

In Algorithm 1 the pseudo code of SPO is presented. Note, that in this section we will use the notation $\vec{x}^{(i)}, y^{(i)}$ for the data passed to the surrogate model instead of the data used for the learning algorithms.

In the sequential improvement loop SPO optimizes the current model $Y(\vec{x})$ over the considered space of input variables by means of a utility function. In the simplest case this utility function is the estimated output itself as this should reflect the performance of the algorithm A. But more sophisticated criteria are possible to select the next sampling points, and in case of Kriging surrogate models they are often termed infill criteria. When the new sampling points have been selected, the number of replications is increased, the required evaluations at the design points are performed and the surrogate model is updated.

The whole procedure serves two primary goals. One is to determine good parameter settings for A, thus SPO may be used as a tuner. Secondly, variable interactions can be revealed in order to understand how the tested algorithm works when confronted with a specific problem or how changes in the settings influence the algorithm's performance. The SPO approach tries to tackle both goals of (i) tuning and (ii) understanding complex procedures, e.g., optimization algorithms or machine learning models.

2.5.4 Kriging

In the 1950s Krige [160] presented an regression technique, which was mathematically formalized by Matheron [174] and later became popular as Kriging. Sacks *et al.* [209] used the Kriging approach in their Design and Analysis of Computer Experiments (DACE). Jones *et al.* [133] successfully integrated the DACE approach for performing a global search on various test functions yielding in the efficient global optimization (EGO) algorithm. Inspired by the findings in DACE and DoE, Bartz-Beielstein *et al.* [11] integrated Kriging as surrogate model into the sequential parameter optimization framework, which offers the user an easy to handle interface with a fully flexible design of inner components like the initial design choice, and infill criteria.

2.5.4.1 Kriging in SPO.

First of all we make the usual assumption for a general regression setting that the output data are subject to model errors, i.e., we have noisy measurements $y^{(i)}$ at the i-th data point $\vec{x}^{(i)}$, where $\epsilon^{(i)}$ is the measurement noise.

The SPO approach consists of two steps:

(1) model construction and

(2) optimizing the model

We will only describe the model construction for Kriging here, where both steps are based on the maximum likelihood estimation (MLE) approach presented in [132, 83].

The Kriging model is constructed as follows: Assume we have a number of evaluated points (e.g., an initial design by LHS) denoted by $\left((\vec{x}^{(i)}, y^{(i)})\right)$, $(i = 1, \ldots, n)$. The observed responses $\vec{y} = \left(y^{(1)}, y^{(1)}, \ldots, y^{(n)}\right)$ are considered as if they were from a Gaussian process, i.e., we will use a set of random vectors $\mathbf{Y} = \left(Y(\vec{x}^{(1)}), \ldots, Y(\vec{x}^{(n)})\right)^T$ with associated $n \times n$ correlation matrix

$$\mathbf{\Psi} = \left(\mathrm{cor}(Y(\vec{x}^{(i)}), Y(\vec{x}^{(j)}))\right)_{i=1,j=1}^{n} \tag{2.33}$$

and correlation function

$$\mathrm{cor}(Y(\vec{x}^{(i)}, Y(\vec{x}^{(l)}) = \exp\left(-\sum_{j=1}^{n} \theta_j (x_j^{(i)} - x_j^{(l)})^2\right), \tag{2.34}$$

where θ is the correlation parameter of the Kriging model. Under standard assumptions, cf. [83], the likelihood can be measured by

$$L(\mathbf{Y}^{(1)}, \ldots, \mathbf{Y}^{(n)}|\mu, \sigma) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{\sum(\mathbf{Y}^{(i)} - \mu)^2}{2\sigma^2}\right). \tag{2.35}$$

In order to filter noise, a regression constant λ can be added to the leading diagonal of $\mathbf{\Psi}$, which is known as nugget effect in the literature [248]. Expressing Eq. 2.35 in terms of the sample data, taking derivatives, and setting to zero, we obtain the estimates

$$\hat{\mu} = \frac{\vec{1}^T(\mathbf{\Psi} + \lambda\mathbf{I})^{-1}\vec{y}}{\vec{1}^T(\mathbf{\Psi} + \lambda\mathbf{I})^{-1}\vec{1}}, \tag{2.36}$$

$$\hat{\sigma}^2 = \frac{(\vec{y} - \vec{1}\hat{\mu})^T(\mathbf{\Psi} + \lambda\mathbf{I})^{-1}(\vec{y} - \vec{1}\hat{\mu})}{n}, \tag{2.37}$$

and the *concentrated ln-likelihood function*

$$\ln(L) \approx -\frac{n}{2}\ln(\hat{\sigma}^2) - \frac{1}{2}\ln\det(\mathbf{\Psi} + \lambda\mathbf{I}). \tag{2.38}$$

The unknown Kriging parameters can be determined, that is the vector $\vec{\theta}$ introduced in Eq. 2.34 and the regression constant λ, by maximizing the concentrated ln-likelihood function.

In a next step an optimization on this model can be performed using any optimization heuristic (e.g., CMA-ES or any iterated local search strategy).

A prediction at $\vec{x}$ can be given by

$$\hat{y}(\vec{x}) = \hat{\mu} + \vec{\psi}(\vec{x})^T(\mathbf{\Psi} + \lambda\mathbf{I})^{-1}(\vec{y} - \vec{1}\hat{\mu}), \tag{2.39}$$

with $\hat{\mu}$ as defined in Eq. 2.36 and $\vec{\psi}(\vec{x})$ the vector of correlations between the observed data and a new prediction, i.e.,

$$\vec{\psi}(\vec{x}) = \Big(\mathrm{cor}(Y(\vec{x}^{(1)}), Y(\vec{x})), \ldots, \mathrm{cor}(Y(\vec{x}^{(n)}), Y(\vec{x}))\Big)^T.$$

The variance (model uncertainty) at this prediction can be estimated by

$$\begin{aligned}\hat{s}^2(\vec{x}) =& \hat{\sigma}^2\Bigg(1 - \vec{\psi}(\vec{x})^T(\mathbf{\Psi} + \lambda\mathbf{I})^{-1}\vec{\psi}(\vec{x}) + \\ &+ \frac{1 - \vec{\psi}(\vec{x})^T(\mathbf{\Psi} + \lambda\mathbf{I})^{-1}\vec{\psi}(\vec{x})}{\vec{1}^T(\mathbf{\Psi} + \lambda\mathbf{I})^{-1}\vec{1}}\Bigg),\end{aligned} \tag{2.40}$$

with $\hat{\sigma}^2$ as defined in Eq. 2.37. Because of the inclusion of the nugget effect λ Eq. 2.40 does not completely reduce to zero when we calculate it at an already evaluated sample point, which can be advantageous for noisy optimization.

2.5.4.2 Infill criteria

After having built the Kriging model for the current design, we can perform an optimization on the surrogate function $\hat{f} = \hat{y}(\vec{x})$. This can be done by generating new design points for which a search on the surrogate function is performed. New promising points (infill points) are then evaluated on the real function, and thereafter the surrogate model is updated with the new information. It is wise not simply to choose the best response from

the surrogate model $\hat{f}$ as next candidate solution, but to select the next point using the *expected improvement* (EI) criterion proposed by Jones *et al.* [133] which is defined by the following function:

$$EI(\vec{x}) = E\left[max(f_{min} - Y(\vec{x}), 0)\right] \tag{2.41}$$

where f_{min} is the minimum of all previously obtained real evaluations, and $Y(\vec{x})$ is the random variable of the Kriging surrogate model, which is usually assumed to be normal distributed with mean $\hat{y}(\vec{x})$ and standard deviation $\hat{s}(\vec{x})$.

Jones *et al.* [133] have shown in their article that under this assumption the expected improvement can be calculated in closed form

$$EI(\vec{x}) = (f_{min} - \hat{y}(\vec{x}))\Phi\left(\frac{f_{min} - \hat{y}(\vec{x})}{\hat{s}(\vec{x})}\right) + \hat{s}(\vec{x})\rho\left(\frac{f_{min} - \hat{y}(\vec{x})}{\hat{s}(\vec{x})}\right) \tag{2.42}$$

where $\Phi(\cdot)$ and $\rho(\cdot)$ are the cumulative distribution function and probability density function of the normal distribution, respectively. Note, that the expected improvement in its presented form is only valid for deterministic problems, in other words with a regression constant of $\lambda = 0$. For the expected improvement the gradient can be calculated analytically, but as it is a multimodal function it is usually maximized with an Evolutionary Algorithm or by employing restart strategies.

The EI deals as infill criterion, that is by maximizing the EI new promising candidate points can be determined. While interpolating surrogate-models like Kriging assume uncertainties in undiscovered regions of the search space, an uncertainty of zero is inserted at evaluated points. As described in Sec. 2.5.4.1, in case of noisy observations a regression constant λ can be added to the leading diagonal of $\mathbf{\Psi}$. The EI can support the exploration, because high variances in unknown regions can lead to points with a better $\hat{y}$ promising as the next infill candidate. A more detailed study on infill criteria has been given by Picheny *et al.* [192].

2.5.5 Optimization in noisy environments

In noisy optimization, the target function is not deterministic. A computer experiment would produce an output $y_k^{(1)}$ for a point $\vec{x}^{(k)}$, but when the same point is evaluated again, a different output $y_k^{(2)}$ is produced, which can differ from $y_k^{(1)}$. In the following, we present two possible solutions to this.

2.5.5.1 Replications

Although the nugget effect tries to filter noise within the model predictions, it still can happen that very high noise levels occur. The usual procedure to cope with high noise levels in DoE is to use replicated measurements and to calculate an aggregated value, e.g., the mean of all replicates. This reduces the noise level and may avoid wrong tuning decisions.

But the price for replicated measurements under a limited budget is that fewer infill points can be generated: e.g., if each design point is evaluated three times, we have only one third in the number of infills.

Standard aggregation procedures like calculating the mean of a series of evaluations can be taken as value for the Kriging model. Since Kriging always assumes a certain confidence, it is valuable to pass the replicates to the Kriging model to refine the uncertainty estimation of the model. In this case SPO can make use of replications, that are a pre-defined number of evaluations, that are aggregated via a function $Aggregate$. Often the mean is chosen for this aggregation.

$$Aggregate(\vec{y}) := \frac{1}{m}\left(y^{(1)} + y^{(2)} + ... + y^{(m)}\right) \tag{2.43}$$

Also, different strategies for choosing the number of replications are possible, e.g., spending more repeats in the final exploitation phase, while spending only a small number of repeats in the beginning of the optimization. In order to keep things simple, we present a strategy where in each iteration the newly proposed points are evaluated $n_{repeats}$ times.

The aggregated value is finally fed to the Kriging model. This procedure is the usual way to cope with noise in DoE.

2.5.5.2 Re-interpolation

An elegant method for noisy Kriging optimization (NKO) is the re-interpolation (RI) method by Forrester *et al.* [82]. In SPO either the strategy using repeated evaluations described in Sec. 2.5.5.1 can be used or the *re-interpolation* is incorporated for this purpose: in re-interpolation the Kriging model is fitted twice. A first model is fitted with a non-interpolating Kriging variant due to the noisy observations, then the response values $y^{(i)}$ of the data are substituted by the predictions of the regression model. The data is interpolated in a second step by an interpolating Kriging model, for which the expected improvement can be calculated without modification. The corresponding pseudocode of the RI procedure is shown in algorithm 2.

Algorithm 2: Re-interpolation procedure for noisy Kriging optimization

Set initial design $DES := \left(\vec{x}^{(1)}, \vec{x}^{(2)}, ..., \vec{x}^{(k)}\right)$
Fit non-interpolating model mod_1 using nugget estimation for design DES
Evaluate DES using mod_1
Fit interpolating model mod_2 for responses from previous step
Maximize expected improvement for mod_2

Also, it is provable that for the second model the parameters of the covariance kernel from the first model are already optimal, leading to a much faster fitting algorithm.

2.5.5.3 Comparing Kriging and SVM

Note, that there is a close connection between Kriging (Gaussian processes) and SVM and this holds whether they are used to model regression or classification problems. This connection becomes clearer if one compares the resulting optimization problems for both models and especially if we consider an SVM with least squares loss function instead of the hinge loss. The most important difference is that no full stochastic model or interpretation for the SVM is currently known. Some work has been done by Sollich in that regard [230], but his interpretation does not seem to be universally accepted by the community (Rasmussen and Williams call his construction "rather contrived" in [201]). A full and proper discussion of this connection would exceed the scope of this work and the reader is referred to [74, 188, 201].

2.6 Conclusions

In this chapter we introduced different types of learning, including classification, regression, and time series problems. Learning algorithms that are capable to solve these learning tasks were described, as well as methods to evaluate them. We focus on two different learning algorithms, that are Support Vector Machines (SVMs) and random forests (RFs). Although there exist a large number of other learning algorithms, these two algorithms usually result in good performances on many instances or datasets. The generalization performance was introduced as desired performance measure, but its calculation is computationally intractable. For this reason we gave a number of other evaluation measures, including cross-validation, the holdout set error, and the root mean squared error, which all aim at approximating the real generalization performance. In the experiments of this thesis these evaluation measures are used for benchmarking purposes, and in most cases they give good informations about the real performances of the learning algorithms. As another important part of machine learning, we gave a thorough introduction to feature processing methods, including feature selection and feature construction.

In Section 2.4 we formally defined optimization problems, which are of great interest, since in the remainder of this thesis we apply optimization algorithms to the supervised learning algorithms. Finally, in Section 2.5 we gave a thorough introduction to model-assisted optimization. We described the sequential parameter optimization (SPO), which is a method for tuning algorithms using arbitrary surrogate models. Although the user is free in choosing any surrogate model within SPO, we mainly focus on Kriging surrogate models, since these models generally give good solutions also for complex tuning tasks.

Chapter 3

Feature processing

The term *feature* originates from pattern recognition and image processing and describes information derived from the original input attributes of a dataset. Features itself are attributes again, and serve to improve the prediction of learning algorithms. E.g., in classification the goal of feature processing is to get a better class separation before the prediction model is built. Since feature processing can improve the data quality significantly, it is an important part of ML. In Sec. 2.2 and Sec. 2.3 we have introduced different methods that are capable for feature selection and feature construction respectively.

We apply these methods for two time series applications. The first application is a time series regression problem, based on stormwater prediction. Here the goal is to predict fill levels of a stormwater overflow tank in Germany only based on past rainfall data. In the past, good working features have been given by experts. Nevertheless it is unknown, if the expert features and their parametrizations can even be improved using parameter tuning and feature selection.

The second application is about classifying gestures obtained from a Nintendo Wii console. We analyze a technique from the neuroscience, the Slow Feature Analysis, for creating an improved feature set, making it possible to perform classifications solely with a simple Gaussian classifier.

We highlight the advantages of automatic and semi-automatic methods for feature construction and selection. While feature selection and construction is often applied in a separate process relaxed from the main modeling, we recommend to attach it to the learning process. It can easily be seen that with very simple techniques we can achieve better results than experts, without requiring human interactions during the learning process. We demonstrate that the combination of feature processing together with an integrated tuning can generate better feature sets, which can be seen as a first step towards a comprehensive machine learning framework, see Konen *et al.* [154].

3.1 Generic feature processing for time series analysis

In this section we highlight the importance of feature processing, by combining a learning algorithm with pre-processing operators. We conduct a systematic tuning for the parameters of the pre-processing methods and the learning algorithm simultaneously and show the influence of the parameter tuning on the prediction quality. The simultaneous tuning of pre-processing parameters and the parameters of the learning algorithm are usually not done in practice. However, we think that this can be an important ingredient for building powerful learning algorithms. For this reason we show how the parametrization of the pre-processing operators affects the prediction models.

3.1.1 Research questions

We define the following research questions:

Q1 Is it possible to move away from domain-specific models without loss in accuracy by applying modern machine learning methods on data augmented through generic time-series pre-processing operators?

Q2 Can tuning heuristics give insights into the strengths and weaknesses of the features generated by generic pre-processing operators?

3.1.2 Predictive control

Intelligent predictive control systems provide solutions to environmental engineering processes. In water resource management, efficient controllers of stormwater tanks prevent flooding of sewerage systems, which reduces environmental pollution. With accurate predictions of stormwater tank fill levels based on past rainfall, such controlling systems are able to detect state changes as early as possible. Up to now, good results could only be achieved by applying special-purpose models especially designed for stormwater prediction. The question arises whether it is possible to replace such specialized models with state-of-the-art machine learning methods, such as Support Vector Regression (SVR) in combination with generic time series pre-processing operators, to achieve competitive performance. This study shows that even superior results can be obtained if both SVR and pre-processing parameters are tuned, to improve the performance of the learning algorithm.

Environmental engineering offers important concepts to preserve clean water and to protect the environment. Stormwater tanks are installed to stabilize the load on the sewage system by preventing rainwater from flooding the sewage systems and by supplying a base load in dry periods. Mostly, heavy rainfalls are the reason for overflows of stormwater tanks, causing environmental pollution from wastewater contaminating the environment. To avoid such situations, the effluent (the outflow) of the stormwater tanks must be controlled effectively and possible future state changes in the inflow should be recognized as early as

possible. The time series regression problem of predicting a stormwater tank fill-level at time t from a fixed window of past rainfall data from time t back to time $t - W$ will be referred to as the *stormwater problem* in the remainder of this thesis.

A model that predicts fill levels by means of rainfall data can be an important aid for the controlling system. Special sensors (Fig. 3.1) record time series data which can be used to train such a model.

Figure 3.1: Left: Rain gauge (pluviometer). Right: Stormwater tank.

Again, we are aware of the large number of methods for time series analysis [31], ranging from classical statistical regression to computational statistics, but as we indicated earlier, such methods usually are limited to predict the output by means of autoregression and are only suited for univariate time series [31]. As an alternative very often special-purpose models are incorporated for solving such tasks, which are specially adopted to the problems at hand. This situation is of course very unsatisfying for the practitioner in environmental engineering, because new models have to be created and parameters have to be set again for each problem.

Therefore, it would be an advantage to use methods which are general enough to be re-applied and can easily be adapted to new problems. In this experiment we use Support Vector Machines (SVMs) [63], more precisely Support Vector Regression, as a state-of-the-art method from ML and apply them to the stormwater problem. We commemorate, that learning algorithms usually suited for classification and regression, must be adapted for the time series data first. Therefore we investigate generic pre-processing operators to embed time series data and to generate new input features for the SVM model. In addition, we use sequential parameter optimization (SPO) [8] and a Genetic Algorithm (GA) to find good hyperparameter settings for both pre-processing and SVM parameters. We analyze the robustness of our method against *overfitting* and *oversearching* [198] of hyperparameters.

Preliminary work in stormwater prediction has been done by Konen *et al.*[156], Bartz-Beielstein *et al.* [14] and Flasch *et al.* [79]. A conclusion of these previous publications was that good models can be obtained by using specialized models which are adapted to the

Table 3.1: Real-world time series data from a stormwater tank in Germany. Four scenarios are defined, which are consecutive measurements from April to July 2005.

Set	Start Date	End Date
Set 1	2006-04-28 01:05:59	2006-05-15 09:40:59
Set 2	2006-05-15 09:40:59	2006-06-01 18:20:59
Set 3	2006-06-19 03:01:00	2006-07-06 11:41:00
Set 4	2006-07-23 20:21:00	2006-08-10 05:01:00

stormwater problem.

3.1.3 Stormwater data

Time series data was collected from a real stormwater tank in Germany and consists of $30,000$ data records, ranging from April to August 2006. Rainfall data are measured in three-minute intervals by a pluviometer as shown in Fig. 3.1. As training set we always used a $5,000$ record time window (Set 2, Fig. 3.2) in order to predict another $5,000$ record time window for testing (Set 4) which is not directly successive with regard to time to the training period (see Tab. 3.1). Later, we conducted a hyperparameter tuning with SPOT and feature selection where we used all 4 different datasets (Set1, Set2, Set3, Set4), each containing 5,000 records to analyze the robustness of our approach against oversearching.

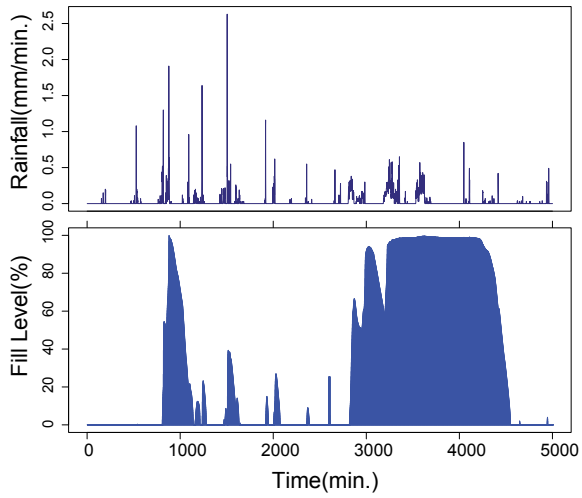


Figure 3.2: Time window used for training (Set 2) showing the rainfall and fill levels of the stormwater tank. It can be seen from the plot, that the rainfall depicts a very spiky curve, causing the difficult predictability of the fill levels.

Table 3.2: Real-world data from a stormwater tank in Germany.

Set	Start Date	End Date
Training Set	2006-04-28 01:05:59	2006-05-15 09:40:59
Test Set	2006-07-23 20:21:00	2006-08-10 05:01:00

3.1.4 Benchmarking time series

The prediction error on the datasets is taken as objective function for the optimization procedure and for the feature selection wrapper by means of GA. For comparing models, we calculate the root mean squared error (RMSE) as a quality measure:

$$RMSE = \sqrt{(mean(Y_{predicted} - Y_{true})^2)} \tag{3.1}$$

We also incorporate the Theil's U index of inequality [239], where the RMSE of the trained model is compared to the RMSE of a naive predictor. The advantage of Theil's U is, that it can be better interpreted by users, whereas the RMSE is just a number depending on the problem structure and range of the output variable. For Theil's U, we are using the mean of the training data target as a naive predictor:

$$U = \frac{RMSE(model)}{RMSE(naive)} \tag{3.2}$$

A value of U greater than 1 indicate models that perform worse than the naive predictor, while values smaller than 1 indicate models that perform better than the naive predictor.

3.1.5 The INT2 model

In previous works [14, 156], the stormwater tank problem was investigated with different modeling approaches, among them FIR, NARX, ESN, a dynamical system based on ordinary differential equations (ODE) and a dynamical system based on integral equations (INT2). All models in these former works were systematically optimized using SPO [8]. Among these models the INT2 approach turned out to be the best one [14]. The INT2 model is an analytical regression model based on integral equations. Disadvantages of the INT2 model are that it is a special-purpose model only designed for stormwater prediction and that it is practically expensive to obtain an optimal parameter configuration: The parameterization example presented in [14] contains 9 tunable parameters which must be set.

Table 3.3: RMSE values of the INT2 model using hand-tuned and SPO-optimized parameter configurations.

Testset	**Hand-tuned**	**SPO-best**
Training Set	24.27	23.79
Test Set	10.61	9.00

We selected two scenarios from the stormwater dataset, each containing 5000 records. The two scenarios served as training and test data respectively (Tab. 3.2). Results obtained by INT2 on the stormwater data is presented in Tab. 3.3 for the training and test sets. We compare hand-tuned INT2 parameters with the best parameter configuration determined by SPO in an earlier study [14].

3.1.6 SVR and time series data

Time series can usually not be handled directly by SVR. SVR does not know the time dependencies. Therefore, we transform the time series regression problem to a regression problem and apply SVR to it. Pre-processing operators are used to add new features based on the input data. There are two possibilities to integrate such operators into the SVR learning process:

- Integration into the SVR kernel function, i.e. replacing standard kernel functions by kernel functions that incorporate pre-processing
- Direct pre-processing of the data, i.e. by augmenting the input feature set with results of pre-processing

In this work we choose the second approach, because the effect of this integration into a model is easier to analyze. In a first step, we compare the effects of applying different types of time series pre-processing operators on SVM model accuracy. Details on pre-processing operators as the *embedding* operator and *leaky rain* as an integral operator suited for time-series analysis are given in Sec. 3.1.6.2 and Sec. 3.1.6.3 respectively. All additional input features generated through these pre-processing operators were used in different variants in the following experimental setups to make them comparable:

E1 Predicting fill levels based only on current rainfall

E2 Predicting fill levels with embedding of rainfall

E3 Predicting fill levels with embedding of leaky rain and rainfall

E4 Predicting fill levels with embedding of leaky rain only

E5 Predicting fill levels with embedding of multiple leaky rain kernel functions

In our experiments we used the radial basis SVM kernel from the *e1071* SVM-implementation in R, since we achieved best results with this kernel choice. Other SVR hyperparameters were obtained by SPO, namely parameters γ and ϵ, to make our results comparable to each other. All models in our study were trained on the training time window and evaluated on the test set (see Tab. 3.2). As an objective function for SPO tuning we

used the prediction error on the test set, which is topic of criticism and will be treated later in this thesis.

In the following subsections, we describe the setup of the pre-processing operators used for these different model variants in more detail.

3.1.6.1 E1: predicting fill levels without pre-processing

The most naive approach is to predict fill levels solely based on the current rainfall, taken as only input feature for the SVM. No matter what SVR hyperparameter configuration was used, the obtained RMSE for this model on the test set could not get about 45.8% better than a naive prediction.[1] Although the SVM prediction is more accurate than the naive one, it can be seen in Fig. 3.3 that the length of high fill level periods are frequently underestimated throughout the whole test period. For this reason, this very simple model is not competitive to models like the INT2 by Konen *et al.* [156].

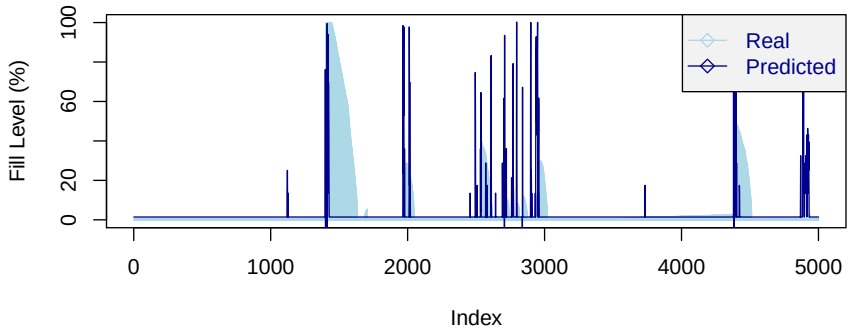


Figure 3.3: Plot of predicted fill levels using only rain data without any pre-processing.

3.1.6.2 E2: embedding of rainfall

One major difference of time series problems in comparison to standard regression problems is that they usually have certain dependencies of successive records. This has not been taken into consideration in our last model, which might be a main reason for its poor accuracy. Of course the model should incorporate that $l(t)$ (the fill level at time t) is somehow related to previous rainfall values $r(t-1), r(t-2), \ldots, r(t-W_{rain})$, for a positive W_{rain} (under assumption that the value $r(t-W_{rain})$ is available in the data). More formally an embedding Ω of a feature r from time t with dimension W_{rain} is defined as follows:

$$\Omega(r, t, W_{rain}) := (r(t), r(t-1), \ldots, r(t-W_{rain})) \tag{3.3}$$

[1] Naive prediction means predicting the mean value of the training set.

Table 3.4: Best parameter configuration for rainfall embedding. The region of interest (RoI) bounds have been refined after preliminary runs.

Parameter	Value	RoI
Embedding W_{rain}	43	$[5, 50]$
SVM γ	0.012	$[0.01, 0.03]$
SVM ϵ	0.012	$[0.01, 0.02]$

There should be a certain relationship between the value to predict and rainfall values (features) from the past. We analyze the extension of the input feature space $\mathcal{F}$ with its historical part $\Omega(r, t, W_{tain})$.

The influence of past data points on the current data point is generally unknown, but might be detected by the SVM model, if we augment each record $r(t)$ with its predecessors $r(t-1), \ldots, r(t-W_{rain})$. Since the embedding parameter might be important for the model quality, we add this parameter to the tuning. This setup led us to a tuning of either the embedding dimension W_{rain} or of the SVM parameters γ and ϵ. The tuning algorithm SPO returned near-optimal parameter values which are presented in Tab. 3.4.

With an RMSE of 14.98 (cf. Tab. 3.7), the SVM model has gained accuracy by using an embedding of past rainfall in comparison to just using the current rainfall data point.

3.1.6.3 E3: leaky rain and rainfall embedding

In the design of the previous model, the rainfall at time $t-40$ has been considered to be of the same importance as the rainfall at time $t-2$. This is of course not true, because loosely speaking, the rainfall from two days in the past should not have the same impact on the fill level as the rainfall from the last 20 minutes. Or more precisely, the rain is a measurable quantity which drains into the soil and then — depending on the consistence of the soil — flows into the stormwater tanks with a certain delay. Therefore the rainfall could be summed up and this integrated quantity could than be used as a new feature of the input data. How can it be expressed that, given $w_2 > w_1$, rainfall from time $t - w_2$ has less influence than $t - w_1$ on $l(t)$? We define the pre-processing operator *leaky rain*

$$\sum_{i=0}^{T} \lambda^i \cdot r(t-i) \tag{3.4}$$

where $\lambda \in [0, 1]$, T is the length of the integration and t is the current time. The formula can be efficiently computed by Fast Fourier Transformation (FFT) even for large datasets.

3.1.6.4 E4: embedding of leaky rain only

Surprisingly the rainfall embedding is no longer advantageous when leaky rain embedding is taken into account. This fact can be clearly deduced from Fig. 3.4, where the RMSE is nearly monotonously increasing with higher rainfall embedding dimensions, given a constant

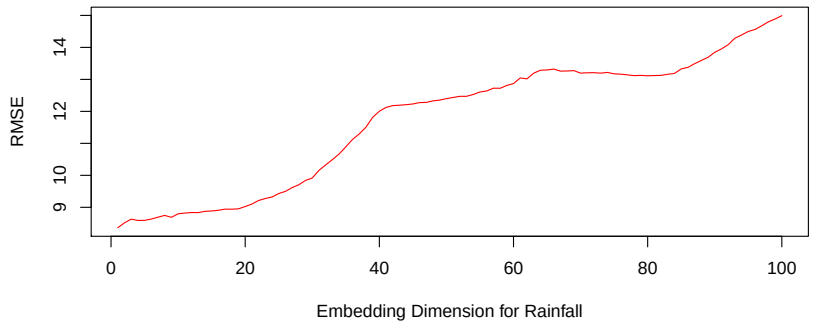


Figure 3.4: Progressively increasing the rainfall embedding dimension W_{rain}, while keeping leaky rain embedding dimension W constant at 43.

leaky rain embedding dimension of $W = 43$. Besides that, the modeling process is slower when higher embedding dimensions are used, because the SVM has to cope with more input dimensions and much more data.

Learning from this observation, we omit the rainfall embedding W_{rain} and concentrate on optimizing the embedding dimension for the leaky rain embedding. Therefore, we performed a similar embedding dimension experiment for leaky rain. Results are shown in Fig. 3.5. The plot shows clearly that there is an almost monotonous decrease of prediction error when using embeddings up to 40 dimensions. The prediction error increases again for more than 40 dimensions. This effect could be explained by two processes. First, the information gain decreases with increasing embedding dimension, because the influence of rainfall on the target decreases with increasing time lag. This may lead to a model with worse generalization capabilities. Secondly, the model is fitted to a higher number of input dimensions, including disturbing factors as noise, measurement errors, etc. Summarized, these factors may lead to more complex models which are prone to overfitting.

Using SPO for tuning the parameters depicted in Tab 3.5 with given region of interest (RoI), SPO delivered an embedding dimension of 43 which is not the global optimum, but is very close to it. Possible improvements on this result can be achieved by increasing the function evaluations in the SPO settings or incorporating a local search strategy on the SPO parameters for fine-tuning.

3.1.6.5 E5: two leaky rain functions

Leaky rain has shown to be an adequate pre-processing function to simplify learning of the target function. However, the true function might be more complex due to factors not incorporated in our simple leaky rain function. Therefore, we employed a more complex pre-processing by using two leaky rain functions at the same time (Fig. 3.6). We tuned the

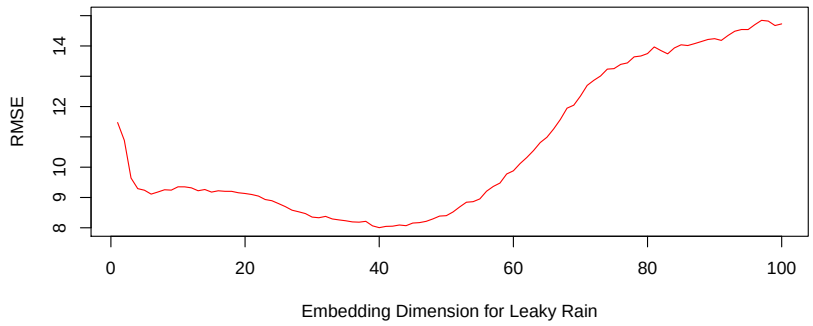


Figure 3.5: Progressively increasing the leaky rain embedding dimension W. The optimal embedding dimension is at $W = 40$ which could be well approximated by SPO. No rainfall embedding was used here.

Table 3.5: SPO parameter for SVM and leaky rain pre-processing

Parameter	Description	RoI
γ	SVM radial basis kernel parameter	$[0.005, 0.01]$
ϵ,	parameter for the insensitive loss function	$[0.005, 0.01]$
T	window size for the calculation of the leaky rain sum	$[50, 120]$
λ	leaky rain kernel parameter (cf. Equation 3.4)	$[0.00001, 0.3]$
W	embedding dimension for leaky rain	$[5, 50]$

parameters λ, T, and W independently for each of the two functions by SPO leading to a new parameter configuration as shown in Tab. 3.6.

After integrating the two kernel functions in our model, compared to a single leaky rain functions, the RMSE slightly improves from 8.09 to 7.80, clearly outperforming the INT2 model again. Note that one leaky kernel function uses a larger λ value and a shorter embedding dimension (less than half the size of the first embedding dimension), so that both functions seem to supplement each other (Fig. 3.6).

In this analysis we used Support Vector Machines (SVMs) as a regression tool for predicting fill levels of stormwater overflow tanks. The results of generic pre-processing are summarized in Tab. 3.7. We can give a postive answer for research question **Q1**. A SVM model is remarkably better than the special-purpose model INT2 under the assumption that i.) pre-processing of the data and ii.) tuning of the SVM *and* the pre-processing parameters is incorporated. By the tuning we could also determine how important certain features are. E.g., for the stormwater problem the embedding of rainfall did not lead to a better performing prediction model. It seems that rainfall itself does not contain enough accessible information to do accurate predictions of the fill levels. The summarized results in Tab. 3.7

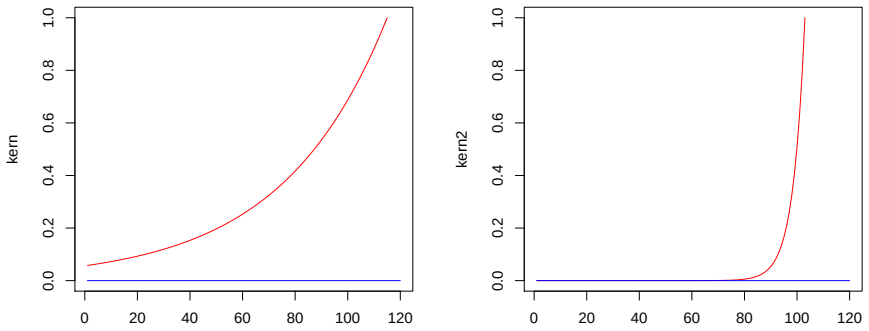


Figure 3.6: Red curves: The two leaky rain kernel functions obtained by SPO. Blue lines: Zero level.

Table 3.6: Best SPO parameter configuration for all tuned parameters evaluated on the test set. The region of interest (RoI) bounds are the values found after preliminary runs.

Parameter Type	Parameter	Best Value found	RoI
Embedding	W_1	39	$[5, 50]$
	W_2	16	$[5, 50]$
	T_1	114	$[50, 120]$
	T_2	102	$[50, 120]$
	λ	0.0250092	$[0.00001, 0.3]$
	λ_2	0.225002	$[0.00001, 0.3]$
SVM	γ	0.0116667	$[0.005, 0.01]$
	ϵ	0.0116667	$[0.005, 0.01]$

show that there is a large reduction of prediction error by using the more sophisticated leaky rain input feature: The RMSE with leaky rain embedding decreases from 14.98 to 10.14. It is an interesting side effect, that we can find out by tuning experiments, which parameters are helpful, confirming research question **Q2**.

As a point of discussion, it has to be criticized that our model parameters were tuned using the test set for evaluation by the optimizing algorithm (SPO). Actually, the prediction error on the test set should only be used for evaluating the model and not for a parameter tuning. Later in this thesis, we analyze if our model is robust to oversearching by introducing an unbiased estimator for time series analysis.

3.2 Learning slow features

Feature processing has a long tradition in ML, data mining and statistical learning. Especially in tasks like time series analysis feature processing plays a key role. Here it is often unclear, how features can be derived from the data. Additionally, time series data are different compared with normal regression and classification tasks. As a consequence it is difficult to apply standard feature processing methods from ML, since they are usually proposed

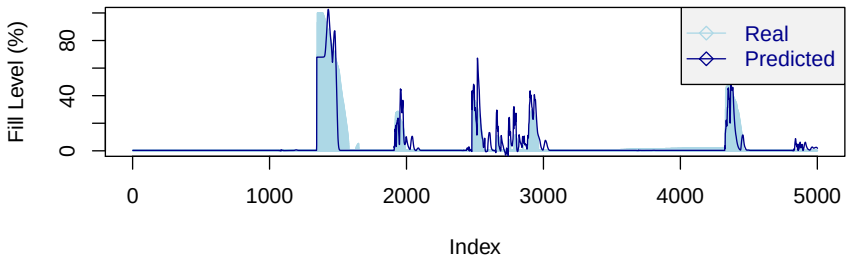


Figure 3.7: Prediction by SVM with two leaky rain embeddings optimized by SPO.

Table 3.7: Precision comparison of all models on testset

Model type	RMSE	Theil's U
Naïve prediction	33.34	1.00
E1: Only Rainfall	18.07	0.54
E2: Rainfall Emb. w/o Leaky	14.98	0.45
E3: Rainfall and Leaky Rain	10.14	0.30
INT2	9.00	0.27
E4: Leaky Rain Embedding	8.09	0.24
E5: Two Leaky Embeddings	7.80	0.23

for classification or regression tasks and do not respect the time dependencies. Thus it is necessary to include special pre-processing — or as another option — to incorporate special-purpose models. However, in ML the majority of research is performed for standard classification and regression. But we think, that time series will become moreover important in the forthcoming years, as most industrial processes involve time as a measurement. Algorithms for modeling time series data include amongst others time series models as ARMA, or GARCH. Unfortunately these methods were only proposed for one univariate time series, but not for modeling multivariate time series. In general, all learning algorithms like Random forest or SVM are able to model multivariate time series. But as a precondition, these methods need to incorporate feature processing, which can be time-consuming and are not necessarily off-the-shelf working well. A method which is able to perform feature processing and time series modeling in one process is the SFA algorithm, which was introduced in Sec. 2.3.3. In a first study, we want to test how good it performs for a real-world classification task.

3.2.1 Research questions

We use SFA to classify gestures obtained from a Nintendo Wii console. The question of interest is, if SFA can be helpful in generating features for the gesture signals:

Q1 Is it possible to construct features with SFA, that a simple Gaussian classifier is able to achieve results comparable to other state-of-the-art learning algorithms, but in shorter computation time?

Q2 How good is the generalization quality of the learned features, measured by the classification accuracy of gestures of unseen persons?

The Slow Feature Analysis has been introduced in Sec. 2.3.3. We give detailed information about the acquisition and preparation of the data in Sec. 3.2.2. The experimental results are described in Sec. 3.2.4 and we give a brief conclusion in Sec. 3.2.5.

3.2.2 Introduction to gesture recognition

The integration of accelerometers in electronic devices such as mobile phones or game controllers for gesture recognition has become popular in consumer electronics. While in the 1990s, the recognition of gestures required a high demand on hardware and was only present in the labs of research institutes, gesture recognition has now made its way into the homes. Even though there already exists a large number of games for the Nintendo Wii game console, recognition of complex gestures still remains challenging. In spite of the large number of classification approaches using the Wiimote controller with the infrared device, see, e.g., the work of Lee [162], we focus here on approaches where no infrared device is used. Although thereby the task becomes more difficult, we are not so dependent on the Wii structure and can easily transfer our approach to other devices. I.e., this can be important, when infrared cannot be used for some reason and only acceleration based sensors are available. Related work has been proposed by Schlömer *et al.* [213], who present a classification tool based on a Hidden Markov Model [200] and Liu *et al.* [168] who applied personalized gesture recognition to a user authentication problem. Rehm *et al.* [204] describe classification of Wii gestures with different methods and expose the influence of cultural behaviours in gestures. Malmestig and Sundberg [171] use the Wiimote to recognize gestures for sign language recognition with good results.

3.2.3 Gesture recognition using accelerometer data

Nintendo invented the Wiimote Controller (Fig. 3.8) as a Bluetooth compliant game controller for the Nintendo Wii console. In its basic version an acceleration sensor, the *accelerometer*, is implemented inside the device. An infrared device for locating the position of the controller during games is also integrated in the Wiimote. The Wii Motion Plus device is an additional device for the Wiimote, based on two gyrometers for calculation

of angular velocities. In order to keep the analysis as simple as possible with respect to the sensor and hardware side, we neither use the sensor information from the Wii infrared sensor (sensor bar), nor from the Wii Motion Plus sensors. As observed in other studies by Schlömer *et al.* [213] and Rehm *et al.* [204] we expect to get comparably good and accurate recognition rates with the sensor data solely taken from the accelerometer.

Figure 3.8: Nintendo Wiimote Controller. The controller is designed as game controller for the Nintendo Wii game console. It contains several buttons, an infrared sensor and an accelerometer. The device is wireless, the signals are transfered via Bluetooth.

3.2.3.1 Data recording and preparation

The dataset consists of five different gestures recorded from ten test persons with the Avetana Bluetooth library[2] together with a modified implementation of the Wiigee framework by Poppinga [195]. Although the number of test persons is rather small, the data shows both varieties and similarities. The acceleration values for several patterns of the frisbee gesture are exemplarily plotted in Fig. 3.9. The accelerometer sensor data were recorded with their timestamps at a rate of approximately 100 Hz. The detection rate was not always equal to 100 Hz, because the data cannot be equidistantly delivered from the Bluetooth interface. In a preliminary data preparation step we interpolated the accelerometer data at 300 equidistant time points between the first and the last timestamp. A first view on the data revealed that the patterns usually vary strongly in execution time and amplitude; even execution times from the same person seem to be rather volatile. In Tab. 3.8 we present a summary of the execution times of a single gesture set, resulting from the ten test persons. Due to the large variance we conduct a smoothing of the gestures by taking the mean of each 10 consecutive data points, resulting in a set of $n = 30$ points for each accelerometer dimension $x_{acc}(t)$,

[2] http://sourceforge.net/projects/avetanabt/

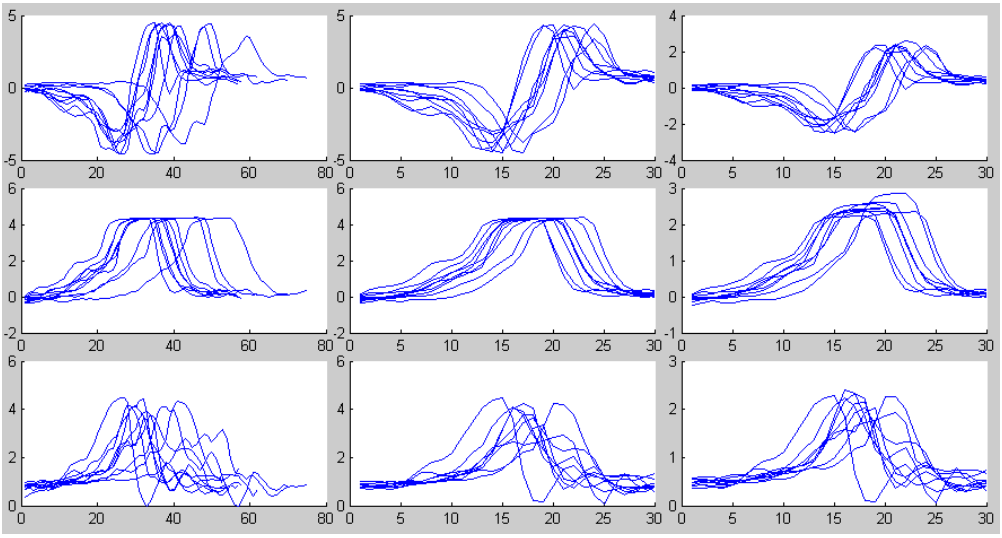


Figure 3.9: Sensor data for all gesture patterns from a frisbee throw obtained from one person. *Left column:* x_{acc}, y_{acc} and z_{acc} acceleration sensor values before time- and amplitude-normalization. *Middle column:* Sensor values after time normalization. *Right column:* Final curves for classification after amplitude normalization.

Table 3.8: Absolute execution times (in seconds) for five example gestures performed by ten test persons. Each gesture was recorded ten times.

Gesture	Minimum	Median	Maximum	Std.Dev.
Circle	0.773	1.607	4.914	0.343
Throw	0.416	0.957	2.306	0.471
Frisbee	0.376	0.883	2.036	0.353
Bowling	0.530	1.325	2.303	0.377
'Z'	0.980	1.607	4.280	0.453

$y_{acc}(t)$ and $z_{acc}(t)$. Another preliminary observation was that the gestures from different persons usually vary in the size of the amplitude. Hence, as an additional operator we use an amplitude normalization by dividing the accelerometer data of each gesture by its standard deviation. In Fig. 3.9 the difference between the gesture signals before and after data preparation can be seen. It is clearly visible that despite the normalization steps were performed, there is still considerable variation within the same gesture type from the same person. Although there is only a small difference between the time-normalized gestures in the middle column and the gestures with amplitude normalization in the right column, the results became slightly better with amplitude normalization for all classifiers.

In order to produce one vector for each gesture we concatenate the multivariate time series data from the three acceleration sensors $x_{acc}(t)$, $y_{acc}(t)$, $z_{acc}(t)$ into a single pattern $\vec{X}(t) = (x_{acc}(t) \oplus y_{acc}(t) \oplus z_{acc}(t))$, where $\oplus$ denotes concatenation, $\vec{X}(t)$ has the dimension

$3n = 90$. We concatenate the raw gesture execution time in seconds as the last dimension, making $\vec{X}(t)$ finally a $3n + 1 = 91$-dimensional vector.

Since 91 dimensions as input to SFA lead to very large processing times and memory requirements, we reduced the dimensions as a final data preparation step by PCA to the n_{pp} dimensions which carry most of the variance in the training data. Usually we let n_{pp} vary between 3 and 20, since higher values are computationally expensive and lower values have a negative impact on the classification. In the case of RF we do not need this dimension reduction.

3.2.4 Experimental analysis

In this section we perform a classification of the recorded gesture signals using a random forest classifier to analyze the performance of the SFA feature construction.

3.2.4.1 Experimental setup

The gestures recorded for recognition were performed ten times by each person. The participants had to push and hold a button on the Wiimote while performing the gestures. The recorded gestures were passed through the data preparation steps as described in Sec. 3.2.2. These pre-processed patterns were passed into the SFA and RF classifiers. All SFA calculations in this thesis were performed with the extended Matlab toolkit `sfa-tk` V2.0 from [153, 16]. For RF we used the *randomForest* R-package.[3]

The gesture set used in our work is thought to be almost realistic for the application in games, since a set of five common gestures is used:

- Circle,
- Throw,
- Bowling,
- Frisbee,
- Z (the letter 'Z' painted in the air)

Both SFA and RF need training patterns as input. Because the classification highly depends on the chosen division of training and test data we differ in the following between several recognition tasks with respect to test and training sets:

(A) *Random sampling:* Classification on the recorded gestures with random sampled test and training set. A 10-fold cross-validation is used to verify the results.

[3] `http://cran.r-project.org/web/packages/randomForest/index.html`

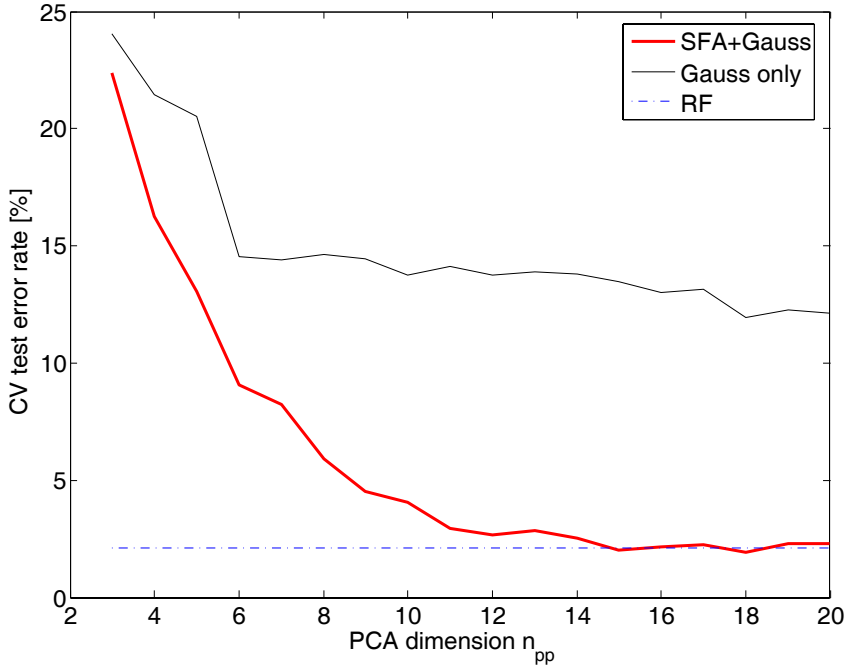


Figure 3.10: Error rates achieved with SFA for different pre-processed dimensions n_{pp} used after PCA preprocessing. Shown are the averages out of 10 runs with different seeds for the CV fold generation. We show for comparison the mean RF error rate 2.09% from Tab. 3.9 (independent of n_{pp}).

(B) *Recognition of unseen persons:* Splitting of training and test set by leaving all gestures from a certain person out of the training set (holdout set). The partitioning is done for all persons sequentially.

3.2.4.2 Random sampling

In figure 3.10 and table 3.9 we show results of a 10-fold cross-validation (CV) from ten independent runs on randomly sampled training data. The 716 gestures were divided into 10 folds containing 71 or 72 records. Each fold in turn was considered as test data while the remaining data were training data. The test error rate on unseen gestures converges quickly to values around 2% as the PCA-reduced dimension attains $n_{pp} = 12$ or above. To measure the impact of the SFA features we directly trained a Gaussian classifier on the same pre-processed input of dimension n_{pp} and tested its performance on the same unseen test data. For $n_{pp} \geq 12$ the pure Gaussian classifier is worse by a factor of 6, showing the strength of the feature combinations found by SFA. We also show in Fig. 3.10 and Tab. 3.9 the results of RF (blue dash-dotted line). RF is slightly better for $n_{pp} = 12$, but for $n_{pp} = 15$ and above SFA and RF are very similar.

Table 3.9: Error rates obtained from ten runs of 10-fold cross-validation on random sampled gesture data. Values in bold are best.

Classifier	Best	Mean	Worst	Std.Dev.
SFA(n_{pp} =12)	2.37	2.82	3.21	0.25
SFA(n_{pp} =15)	1.68	**2.03**	**2.24**	0.18
RF	**1.54**	2.09	2.37	0.30
Gauss	13.55	14.02	14.39	0.22

3.2.4.3 Classifying gestures from unseen persons

In classification with random sampling of training and test sets it is indirectly assumed that the training data is representative for the test set. However, in gesture recognition it is of high importance that gestures of persons who never occurred in the training set can also be recognized by the classifier. Rehm *et al.* [204] highlighted that gestures can be influenced by the expressivity of the user. Factors for expressivity are, e.g., speed, space used for the gesture and the cultural background. Due to the large differences between several persons this might be a quite difficult task for any classifier. Nevertheless this experiment can be of high relevance for the game industry, no person-specific calibration will be needed in case of a good classification.

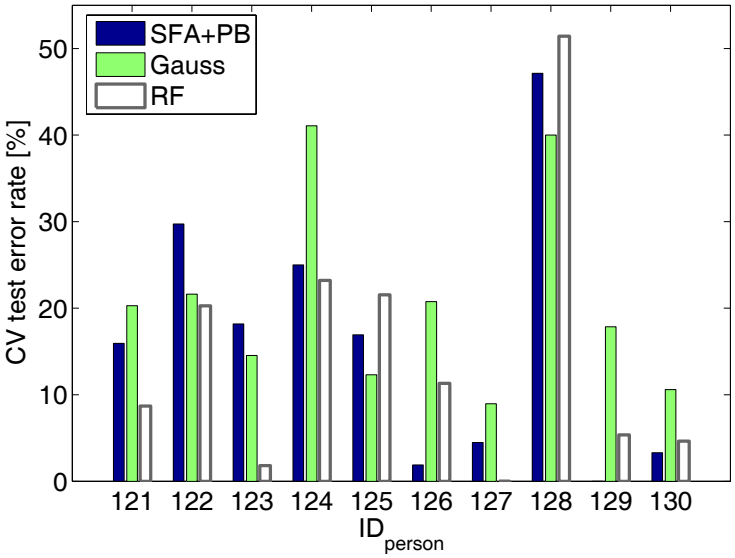
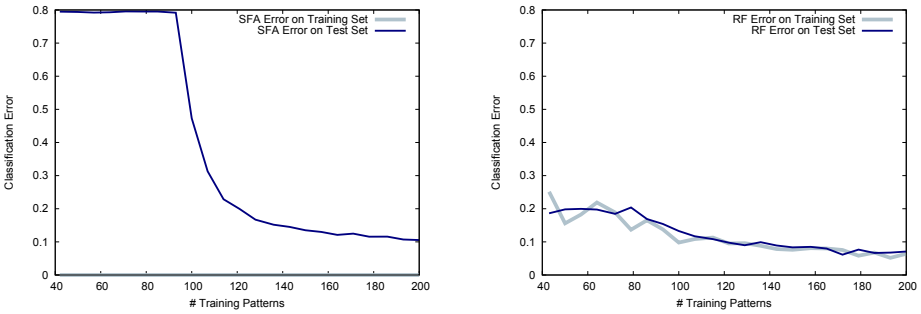


Figure 3.11: Error rates when classifying gestures of persons unseen in the training set (SFA and parametric bootstrap (PB), Gauss: Gaussian Classifier, RF: Random Forest).

In Fig. 3.11 we show the results of a cross-validation experiment for SFA with parametric bootstrap (SFA+PB), a Gaussian classifier and RF when the gestures of each person in



(a) SFA error rates (b) RF error rates

Figure 3.12: (a) Error rates achieved with SFA in the interval of $[40, 200]$ training patterns (parameter setting: $n_{pp} = 12$). The test set error suddenly increases when the number of patterns is too small for a sufficient rank of the covariance matrix (< 90 patterns), while the error on the training set stays constantly near zero. (b) The RF error rates also increase with fewer training patterns, but do not rise to an unnatural high level in contrast to SFA. Additionally, the training set error is a good predictor for the test set error on unseen data.

turn are put completely into the test set. As expected, the error rates are considerably higher than with random sampling. Tab. 3.10 shows the error rate when averaging over all gestures which are 15.3% for SFA, 14.8% for RF, and 19.6% for Gauss. But the most striking attribute of Fig. 3.11 is the great variety of error rates between persons. The gestures of some persons from our dataset could be extremely well classified for both SFA and RF, while others, most prominently gestures of the person with ID 128, were hard to classify for all algorithms. Presumably this person has very different characteristics when performing gestures, but further research is necessary to get more insights for this.

3.2.5 Conclusion

We present the best error rates in Tab. 3.10 for the tasks (A) and (B) from Sec. 3.2.4 for our three algorithms. For SFA we chose settings $n_{pp} = 12$, $\sigma_{nc} = 0.8$ plus $N_{copies} \in \{0, 200, 200\}$ in both cases (A), and (B). For RF we set 500 trees, and parametrized the algorithm with $mtry = 3$. All these settings were considered to be good working and obtained from some preliminary runs. In case (A) the results are averaged and the standard deviations out of 10 runs were added. In case (B) we performed likewise 10 runs, but the test set was fixed in this case to the data of unseen persons (a variant comparable to the holdout set error). With the enhancement of parametric bootstrap the resulting CV test set errors from SFA+PB and RF are similar. With the constructed feature set they are considerably better than a Gaussian classifier trained on the original features, which supports the benefits of SFA.

In the evaluation of the unseen persons, only one cross-validation index (defined by the person ID) is present. Here, the standard deviation is determined with respect to the

Table 3.10: Best cross-validation test set error rates for the three algorithms SFA+PB, RF and Gaussian classifier.

Settings	CV test set error		
	SFA+PB	RF	Gaussian
(A) random sampling (716 records)	(2.9±0.3)%	(2.1±0.3)%	(14.0±0.6)%
(B) unseen persons (716 records)	(15.3±14.9)%	(14.8±15.3)%	(19.6±11.3)%

Table 3.11: CPU times for SFA+PB, RF and a Hidden Markov model [213]. N is the number of records for each task. The CPU times were obtained on a standard desktop computer, namely an Intel® Core2 Duo CPU T7700 running at 2.4GHz on Windows OS. Best values (shortest CPU time) are written in bold font.

Task	N	CPU time (sec.)			
		RF	SFA+PB		HMM [213]
			n_{pp} =12	n_{pp} =15	
train	74	0.55	**0.52**	0.82	60
train	716	5.60	**5.16**	8.00	–
apply	64500	6.61	**1.09**	1.89	–

different persons. It is very large (approximately as large as the CV error itself), because there is a large inter-person variation as already described in connection with Fig. 3.11.

But the classification can benefit from a parametric bootstrap enhancement [151] which was originally designed when few training patterns are available, making the SFA algorithm infeasible. If we run classification on unseen gestures without bootstrap ($N_{copies} = 0$ instead of 200) we get a CV error rate of 17.7% instead of 15.3%. We assume that the variation introduced by the noisy copies of the parametric bootstrap is beneficial for the generalization to unseen persons.

We compared our results with another gesture recognition engine in order to exclude that our gesture set is perhaps "easier" than others. The Hidden Markov Model [213] is included in the software library Wiigee [195] and we used it on a small set of our gestures (74 patterns, randomly selected). The error rate on this training set was 26% which has to be compared with the RF training set error of about 17% from Fig. 3.12(b) for the same training patterns (and with 15% for SFA+PB). Other tests with larger training sets or on unseen test patterns are planned in the future. We conclude however that our gestures do not seem to be fundamentally "easier" for other classifiers. As another comparison we performed the "same-person CV error"-task of [213] on our data and with our SFA+PB model and got a $(2.3 \pm 1.5)\%$ CV error rate where [213] reports 10%.

In Tab. 3.11 we show the CPU times for SFA and RF. Both, RF and SFA, are quite fast

algorithms, since the training takes only about 0.5 seconds for 74 records and 5 seconds for 716 records. This is faster by a factor of 100 compared with the Hidden Markov recognition engine in the Wiigee package [195], which we tested with the same 74 patterns. The training time for the Gaussian classifier is neglectable, actually it is contained in the reported SFA time. The times for SFA include the extra 200 parametric bootstrap records. They are based on an unoptimized Matlab implementation, so there might be possible improvements. Since a trained SFA model has a very simple structure, we found that applying a trained model to new gestures (last line in Tab. 3.11) is for SFA 3–6 times faster than for RF.

We applied Slow Feature Analysis (SFA) to a time series classification problem originating from gesture recognition. The feature construction by SFA improved the Gaussian classifier, which is included in the Matlab toolkit, by a factor of 2–6. Koch *et al.* [151] discovered, that small training sets can lead to overfitting by SFA, which could not be observed with larger training data. For this reason they propose an enhancement to SFA for the case of marginal data, which is based on parametric bootstrapping. Summarizing, SFA can be used as a powerful feature construction algorithm for time series problems. Koch *et al.* [151] obtained results which were equal to or better than the results of a RF classifier, which is really a notable result affirming research question **Q1**. Especially the fast training and prediction times of SFA have to be highlighted here, which could not be beaten by RF as demonstrated in the runtime comparison. This is an indicator that SFA can be profitably applied for feature pre-processing, and can be seen as a promising alternative to linear methods like PCA. The prediction of unseen test persons appeared to be more difficult for all learning algorithms. Although SFA did not perform much worse than RF and SVM, this task needs further experimentation, partially affirming research question **Q2**.

3.3 Conclusions

We highlighted the influence of features on the prediction performance of learning algorithms. Features can be the input attributes of the data, or can be constructed based on the original input features, e.g., by using any projection or aggregation method.

One of the most famous feature processing methods is probably *feature selection*. Feature selection can be used for reducing the number of input attributes of large data sets. Furthermore it is a nice solution to remove irrelevant or redundant information in the data. From a more target-oriented perspective, feature selection aims at increasing the accuracy of learning algorithms. We described differences between methods which are fast to evaluate, that is the filter approaches, and more expensive wrapper approaches. Here the user has to decide how much time can be spent for feature selection, and how good the feature set should be. Wrapper methods sometimes provide a better feature set, but are also more expensive, because the learning process has to be applied several times. Here, filters can be helpful, which only perform a short analysis of the data. However, filters are

sometimes not competitive with the more expensive wrappers, and another advantage is that wrappers select features according to the result of a specific learning algorithm. This can be advantageous, especially when the learning algorithm is deteriorated by misleading features.

Another important step in creating good performing learning algorithms is feature construction. Sometimes the input attributes are not appropriate for predicting the target, but combinations or transformations of the input attributes can lead to better features. Therefore, many methods are available in machine learning, e.g., PCA as a representative for linear projections. Creating the monomials of input attributes constitutes a very simple feature projection, which is sometimes already sufficient to improve the prediction. The main disadvantage of monomials is, that the dimensionality of the feature space grows exponentially with an increase in the number of input attributes. For this reason, seldom monomials are built for all input attributes. Here, ranking procedures of feature selection can help to decrease the total number of features generated by the method.

The Slow Feature Analysis (SFA) can generate more sophisticated features compared with rather simple strategies like PCA and monomials. SFA emerged as a method from neuroscience and was originally developed in context of unsupervised learning [254]. We used SFA as pre-processing method for a time series classification task. Here, SFA could achieve comparable performance like state-of-the-art learning algorithms as random forest (RF) or Support Vector Machines (SVMs) but in much shorter time. As a new aspect we discovered, that the size of the training set must be larger than the dimension of the expanded function space of SFA given by a hyperparameter $xpdim$. Otherwise we observed a negative behaviour of the method, resulting in overfitting to the data, that is worse generalization performance. As soon as additional patterns were added by using a resampling technique from statistics, the concept could be learned again. As a consequence we recommended two approaches to circumvent this issue:

i) adding artificial training patterns by using bootstrapped data or

ii) decreasing the expanded function space of SFA, whose dimension is $xpdim$.

In general both options should work well. However, the first option should be preferred if possible, because a decreased function space governed by $xpdim$ can lead to significantly worse accuracy rates. That is, the advantage of SFA compared with PCA or other linear techniques is lost by decreasing the feature space dimension.

The experiments on stormwater prediction showed that a tuning of the pre-processing parameters (here, the two different leaky rain embeddings) and the model parameters at the same time leads to the best performing model. So far, tasks for time series regression were mainly solved by using standard techniques like ARIMA or GARCH [31]. We showed how feature processing can handle time series data as a normal regression task. The pre-processing operator used is generic and can be re-used for similar problems. The crucial

technique responsible for the good results is termed *embedding*, and consists of adding earlier values of the time series as new features. By doing this, the time dependency of the records is relaxed, which makes it possible to apply any supervised learning algorithm instead of being restricted to specific time series models. A learning algorithm based on SVR could learn the target function very well, using tuning via SPO and the embedding operator for pre-processing only. It showed that the results were a little bit overfitted to the training data, which requires further analysis.

Concluding the feature processing we can say that also combinations of feature projections and feature selection can be helpful in practice. E.g., after applying a PCA, a feature selection of the principal components with the highest variance can reduce the feature number, without losing much information. When simple feature projections do not give sufficient improvements, more powerful non-linear techniques can be helpful. Here we have proposed to use techniques like Genetic Programming or Slow Feature Analysis which can be adapted to arbitrary complexities.

Chapter 4

Parameter tuning

The prediction quality of learning algorithms is largely affected by their parameter settings. Parameter tuning can be performed by defining a functional mapping from a parameter setting and a certain training set to the prediction performance of the learning algorithm. As a benefit of this effort, a fair model selection [35, 263] can be done for the tuned models.

This chapter is structured as follows: in Sec. 4.1 we give a short review of existing tuning approaches for ML parameters from the scientific literature. According to this we mention important factors and ML parameters in Sec. 4.2. In Sec. 4.3 we perform tuning experiments on different case studies, including two real-world problems, firstly the stormwater problem introduced in Sec. 3.1.2, and second a prediction task for acid concentrations in biogas plants. We compare the performance of the tuning approaches. Also a selection of datasets from a public data repository is considered for this comparison. Finally, in Sec. 4.4 we describe an approach for evolving a substantial part of the SVM classifier, namely the kernel function.

4.1 Related work

Dubrawski and Schneider [65] tuned parameters for Bayesian local weighted regression [181] using a Racing [173] approach, where poorly performing regions of the search space are less often evaluated.

In the SVM field a plethora of work in tuning algorithm parameters is available. Early work included the optimization of SVM kernel parameters for binary-class problems. E.g., Campbell *et al.* [36] proposed an incremental strategy for optimizing SVM kernel parameters. Later, Chapelle *et al.* [37] used a gradient-based approach to determine optimal kernel parameters. Gold and Sollich [97] optimized the regularization parameter C and the kernel parameter γ for L_2 soft-margin SVM. Cohen *et al.* [46] recommended to use direct search strategies, e.g., the Simplex strategy by Nelder and Mead [185], due to possible non-differentiabilities of the error function. Keerthi *et al.* [138, 139] integrated the BFGS method [32, 81, 99, 219] to improve the efficiency of the optimization. However, it is

known that local search methods can easily converge to local optima, when they are applied to multimodal problems. Although Keerthi *et al.* [139] stated, that local optima did not occur in their observations, Duan *et al.* [64] showed, that already for a single tuning parameter the optimization problem can become multimodal. The same result was observed by Imbault and Lebart [124]. They proposed to use global optimization based on Genetic Algorithms and Simulated Annealing for determining optimal hyperparameter settings. Likewise Chen *et al.* [40] and He and Jiang [110] report on using Genetic Algorithms for SVM parameter tuning with good performance. Sun *et al.* [236] used a performance measure for tuning based on the distance between the two classes in a binary classification problem. Although most often binary classification was considered in experimental analysis, de Souza *et al.* [55] incorporated Particle Swarm Optimization (PSO) [140] for tuning, with a special focus on multi-class problems. Christmann *et al.* [43] compared the performance of several algorithms for hyperparameter tuning. Friedrichs and Igel [88] also integrated more complex kernel functions in the tuning. Their approach was based on CMA-ES to optimize the hyperparameters and they demonstrated a superior performance compared with grid search. Glasmachers and Igel [93] presented an improved approach for general Gaussian kernels. Although they mention a decreasing prediction error in their experiments, their approach suffers from an increasing number of support vectors. Later, Glasmachers and Igel [94] used the uncertainty handling CMA-ES (UH-CMA-ES) [106]. The overhead caused by the re-evaluations in the UH-CMA-ES could be justified by better performance of the approach in an empirical study [94].

Although it is sometimes claimed differently, Strobl *et al.* [232] showed empirically that for ensemble methods like RF some parameter values are worth to be tuned. Additionally Segal [218] reports of a study, where RF was trained on artificial and real-world datasets and he sometimes observed a remarkable overfitting. Therefore, Segal proposed to tune all relevant RF parameters. For boosted trees and RF Ganjisaffar *et al.* [90] considered a MapReduce [56] technique to enable efficient parameter tuning using parallel computing.

4.1.1 Delimitation from existing work

The main difference between the mentioned approaches and the tuning approach described in this thesis is the focus on systematic parameter optimization of the tunable parameters. Besides that we see a need to tune *multiple parameters of the learning process* at the same time, and not only the parameters of the learning algorithm. Tuning, especially in the SVM field, is sometimes restricted to binary classification problems. We think that this is very restricting, although SVMs were originally proposed for binary classification. However, tuning should be possible for multi-class problems and arbitrary performance measures. To our knowledge there has not been a systematic comparison of model-assisted optimization techniques and other approaches in supervised ML. For this reason we propose to respect

the following points:

i) use model-assisted optimization (e.g., SPO or EGO) for tuning in order to reduce the function evaluations

ii) include the whole *learning process* in the tuning, and not only the hyperparameters of the learning algorithm

iii) make it possible to interpret the importance of the parameters

4.2 Hyperparameters

Hyperparameters are the parameters of a learning algorithm. In this section hyperparameters are described, each having a certain influence on the performance. The importance and influence of the parameters can vary from problem to problem, here we only enumerate parameters which are important for the quality of the prediction models.

A learning process requires good settings of underlying parameters in order to receive satisfying results. Since most learning parameters need to be set corresponding to the properties of the data, no prior knowledge can be used in general. Here, methods for global optimization come into play offering solutions for such optimization tasks.

This section is structured as follows: first we present tuning parameters of learning algorithms in Sec. 4.2.1. Then we introduce additional parameters for feature pre-processing in Sec. 4.2.2 and for post-processing in Sec. 4.2.3.

4.2.1 Parameters of learning algorithms

Some learning algorithms require specific parameters to be set, some do not. E.g., a linear model for regression does not require any additional parameter setting — it works out of the box. However, this does not hold for all learning algorithms. Examples of learning algorithms with additional hyperparameters are Random Forest (RF) and SVMs. More complex learning algorithms usually require hyperparameters and are more powerful, but otherwise they need careful settings to produce good results. In this thesis we focus on tuning parameters for RF and SVM, because they are state-of-the-art learning algorithms. However, other learning algorithms can be tuned in the same way. In Tab. 4.1 an overview of hyperparameters for the RF and SVM learning algorithms are shown.

Although RF is a very robust learning algorithm, it is nevertheless sometimes necessary to tune its hyperparameters. Common tuning choices are the number of trees [232]. Depending on the application and data also other RF parameters might become important, i.e., parameters *mtry* or *sampsize*.

Related work of parameter tuning (see Sec. 4.1) mostly involved the tuning of SVM hyperparameters. The reason is that if SVM parameters are set wrong, the predictions of the trained model can be a complete random guessing. Besides the parameters of the

Table 4.1: Parameters of machine learning algorithms.

Learning algorithm	Parameter	Type	Description
SVM			
	$k(x, y) \in \{\text{linear, radial, polynomial}\}$	factor	kernel function
	γ	numeric	RBF kernel parameter
	C	numeric	cost, regularization parameter
RF			
	mtry	integer	number of splits tried per tree node
	ntree	integer	number of trees in ensemble
	sampsize	integer	sample size used for each tree

kernel function, the kernel function itself can also be seen as a hyperparameter of the SVM. Although the first step in learning with SVM should be to test different kernel functions, we keep the kernel function fixed and always use the RBF kernel. The effect of choosing other kernel function is shown in Sec. 4.4, where kernel functions have been evolved using GP. For the RBF kernel, the kernel width is defined by parameter γ, which can be also found under name σ, and the regularization parameter C or $Cost$, sometimes also referred to as ξ.

In general, the accuracy in ML usually depends on different factors:

- quality of data: presence of noise and missing values, general informative value of input attributes
- dimension of data: number of records multiplied with number of attributes
- accuracy of learning algorithm
- pre-processing of the data
- parameter settings

In Fig. 4.1 the learning is depicted as a process from the raw input data to a prediction model. This process includes the basic steps of a generic ML process. It is important to distinguish training and test sets, here shown by holdout set sampling. A clean evaluation of a prediction model includes the generalization performance and not the performance obtained on the test set. For feature processing the methods of Chapter 3 can be applied to select a subset of relevant features, or to transform the raw features by projection methods like SFA or PCA. It has to be noted, that these methods sometimes require additional parameters. E.g., for SFA the $pprange$ parameter for the basis feature set must be defined.

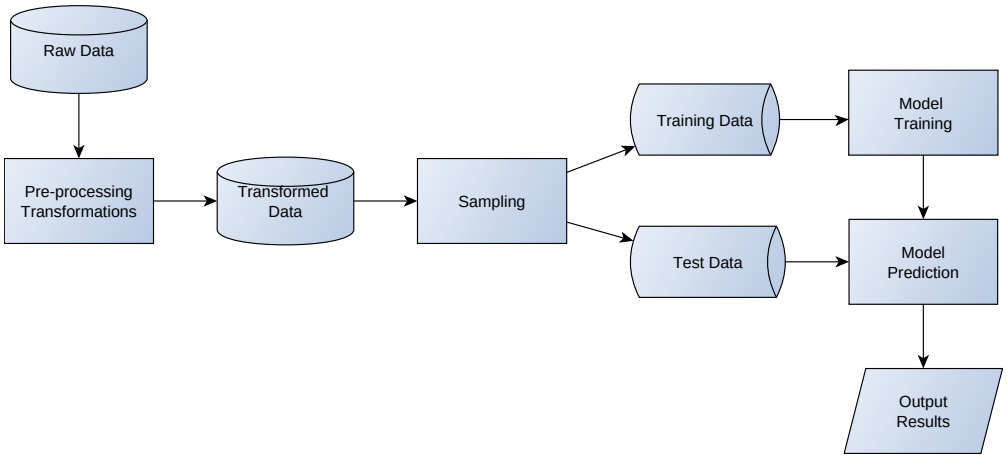


Figure 4.1: Supervised machine learning process. The raw data is first pre-processed. Pre-processing can be any combination of feature processing methods. Examples are feature selection or Principal Component Analysis (PCA) leading to transformed data. Afterwards this data is split into training and test data (sampling). Then only the training data is passed to the learning algorithm, while the test data is solely used to calculate the performance of the learned prediction model.

After the feature processing the learning algorithm is initiated using the sampled training data. In a final step it is also possible to alter the predictions by using class weights or other techniques. Possible approaches for post-processing are therefore described in Sec. 4.2.3.

4.2.2 Pre-processing parameters

Pre-processing certainly belongs to one of the most important steps of the learning process. Pre-processing incorporates all feature processing methods from feature construction and feature selection, as has been already described in Chapter 3. For many tasks feature processing is the key towards successful use of learning algorithms, because a dataset with good descriptive features builds the fundament of successful ML.

An important parameter in pre-processing is the number of features to be selected by a filter method fi. One possibility would be to take the best $j \in \mathbb{N}$ features according to the ranking fi returns. Instead of the absolute number of features, sometimes a concept named $xperc$ is alternatively chosen. The parameter $xperc$ describes the relative percentage of information content of all features. Let us assume that we have ranked features $F_1, F_2, ..., F_N$ using a filter method fi (see Sec. 2.2.1). The ranks are denoted by $R_{fi} = (r_1, r_2, ..., r_N)$. For a value $xperc$ the feature subset is now determined using a greedy selection strategy in algorithm 3.

In feature construction methods like PCA the number of principal components to be selected is an important parameter. Here, also an absolute value could be set. The principle is the same as for the filter method. Feature processing methods like GP reveal a lot of different parameters like the function set, the genetic variation operators, population size

Algorithm 3: Feature selection using filter method *fi* and *xperc*

Define $xperc \in [0, 1]$
$S_{fi} = \emptyset$
Sort R_{fi} by decreasing rank according to filter *fi*
Set $sfi = 0$ and $i = 1$
while $sfi \leq xperc$ ***and*** $i \leq |R_{fi}|$ **do**
  $S_{fi} := S_{fi} \cup \{R_{fi}[i]\}$ (add feature $R_{fi}[i]$)
  $sfi := \sum S_{fi} / \sum R_{fi}$
  $i := i + 1$
end
return feature subset S_{fi}

and so on. Because it is difficult to preset the values for a new ML task, the pre-processing parameters can be plugged into the tuning.

4.2.3 Post-processing parameters

In classification it is usually assumed, that all classes are equally important. However, sometimes we are interested in learning a model, which is capable of classifying class $k \in \{1, \ldots, K\}$ correctly. When only a few training examples are available for class k, it can be difficult to learn the correct concept for this class. Here it might be helpful to define a cost matrix.

A cost matrix provokes positive costs, when a pattern $\vec{x}$ belongs to class i, but is classified as class j. Therefore in the following equation a cost matrix is defined as $C(i, j, \vec{x})$ and the probability of classifying pattern $\vec{x}$ as class j is written as $P(j|\vec{x})$:

$$\sum_{j=1}^{K} P(j|x)C(i, j, x) \tag{4.1}$$

Cost functions have been intensively studied in ML, e.g., Domingos invented the MetaCost algorithm [61], and Elkan propagates a similar approach [68] for imbalanced class sets. A taxonomy for cost-sensitive learning was proposed by Turney in [242]. All approaches enable to use cost-sensitive learning even with non cost-sensitive classifiers.

Now cost-sensitive learning can be connected with tuning. E.g., the Random Forest and SVM learning algorithms both provide probabilities $P(j|\vec{x})$ after any prediction. A vector $C = (c_1, c_2, ..., c_K)$ of class weights can then be used to improve the classification result, even after the outputs of the class patterns have been predicted by the classifier. Therefore, the probability $P(j|\vec{x})$, is multiplied with the vector C:

$$\arg\max_j C_j \cdot P(j|\vec{x}) \tag{4.2}$$

The result of Eq. 4.2 defines the prediction of pattern $\vec{x}$. The weights $c_1, c_2, ..., c_K$ can be considered as additional tuning parameters. Because tuning sometimes leads to large,

undesired changes of the vector components, the weight vector is normalized after each tuning step on a $[0, 1]$ scale:

$$c_j := \frac{c_j}{\sum_{i=1}^{K} c_i}, j \in \{1, \ldots, K\} \tag{4.3}$$

4.3 Tuning of machine learning processes

It is important to understand that all elements of a ML process, starting from feature processing, to the training of the learning algorithm, to post-processing comprise the final prediction model. If one part of this process does not work properly, this can negatively affect the whole prediction model.

When elements of the learning process are selected or replaced, this can have a direct effect on the set of the tunable parameters. As soon as a certain process operator is replaced by another, the hyperparameters should be re-tuned, even when only little changes have been made (e.g., the selection of a different feature selection operator).

The task to be solved can be defined as follows: a set of parameters $\vec{x} := (x_1, ..., x_n) \in S \subseteq \mathbb{R}^n$ shall be optimized for an arbitrary learning process. For a dataset $\mathcal{D} = (\vec{X}_1, Y_1), (\vec{X}_2, Y_2), \ldots, (\vec{X}_m, Y_m)$ the learning algorithm uses training data $\mathcal{L} \subseteq \mathcal{D}$ to create a prediction model M. Then the prediction model M is used to predict test data $\mathcal{T} \subseteq \mathcal{D}$. Usually the dataset is splitted into training and test data, that is the combination of both training and test data is equal to the original dataset again. For evaluation purposes we use the holdout error (see. Sec. 2.1.3). The reason is, that the holdout error can be quickly calculated. In tuning, learning processes need to be built very often. Other performance measures like CV or LOOCV give good estimates of the real generalization error, but are expensive on large-scale ML tasks. For this reason we use the holdout error as performance measure, but with a repeated resampling in the optimization loop, so that still a good estimate of the generalization performance can be achieved. The prediction accuracy y (in percent) defines the performance (which can also be written as prediction error $y_{err} = 100 - y$) of the learning algorithm and is optimized by the tuning algorithm:

$$f(\vec{x}, alg, \mathcal{L}, \mathcal{T}) \rightarrow y \in [0, 100] \tag{4.4}$$

The above equation can be understood as a training of a specific learning algorithm alg using training data $\mathcal{L}$ and hyperparameters $\vec{x}$. As a result of this learning phase, the machine learning model M is returned. This model is then finally applied to predict test patterns $\mathcal{T}$ from which the model performance y can be calculated and passed to the optimization method. Note, that the fitness y *only gives an estimate* of the real generalization performance. The value of the prediction error is dependent on the training and test data passed to the evaluation function. The output of the model will correspond to this selection. Although the model M can be deterministic to parameter settings $\vec{x}$ under constant $\mathcal{L}$ and $\mathcal{T}$, the

output of f still can be subject to noise, because in every new evaluation call new sets $\mathcal{L}$ and $\mathcal{T}$ should be sampled (iterated resampling) and can lead to other outputs and other minima of f.

Our aim is to find the parameter $\vec{x}^{(*)}$ which gives the best value f using a fixed number of evaluations or budget, of resampled training $\mathcal{L}$ and validation sets $\mathcal{T}$.

4.3.1 Stormwater prediction

We give a detailed example for a tuning process. We performed experiments for the stormwater prediction task, which has already been tackled in Sec. 3.1.3. The study is based on the earlier experiments, but now a special focus is given to measure *oversearching* effects caused by the tuning. Biases towards the training data can occur due to the repeated evaluation of models during the tuning process. This is also known as *oversearching* in the literature.

A distinction between *oversearching* and *overfitting* was given by Quinlan and Cameron-Jones [198]:

> Overfitting refers to the construction of a theory tailored to the data that has high (but misleading) apparent accuracy. By analogy we use the term oversearching to describe the discovery by extensive search of a theory that is not necessarily overcomplex but whose apparant accuracy is also misleading.

For this reason we have to take care of oversearching effects as a main issue caused by systematic parameter tuning.

4.3.1.1 Research questions

The research questions can be defined as follows:

Q1 Is it possible to move away from domain-specific models without loss in accuracy by applying modern learning algorithms and state-of-the-art parameter tuning methods on data augmented through generic time-series preprocessing operators?

Q2 Does parameter tuning lead to oversearching in time series regression, and can the effect be measured using multiple mini time series?

4.3.1.2 Experimental setup

In the following we describe the experimental setup for the stormwater data. We briefly recapitulate the embedding and leaky rain pre-processing operators analyzed in Sec. 3.1.6, but now we tune the hyperparameters of these operators using different tuning algorithms, to enable a fair comparison of these methods.

Time series pre-processing

ML methods can be profitably applied for time series regression. Examples were given by [142, 182, 183, 210, 240]. Although common ML methods like SVM or RF are not directly able to handle time series data, the time series problem can be transferred to a normal regression problem, by incorporating feature processing methods. New features can be added, which are based on historic time steps in the time series. For the stormwater problem it turned out that good results could be obtained with the following time series operators (also see Sec. 3.1.6 for our earlier analysis of pre-processing operators for the stormwater data):

- Embedding of the input data [135]. Here, the fill level of stormwater overflow tanks $l(t)$ can be represented by a function F on historic input features, more precisely by the rainfall $r(t)$ up to $r(t-D)$, where t indicates time and $D \in \mathbb{N}^+$ is the embedding dimension:

$$l(t) := F(r(t), r(t-1), r(t-2), ..., r(t-D)) \tag{4.5}$$

- Leaky integration of rainfall (termed *leaky rain* l) which is computed as follows:

$$L(t) = \sum_{i=0}^{W} l^i \cdot r(t-i) \tag{4.6}$$

  with $l \in [0, 1]$ and $W \in \mathbb{N}^+$ is the length of the integration window. Best results with SVM and leaky rain pre-processing were obtained with two different leaky rain functions with two different settings for l and for W.

Tuning setup

We use sequential parameter optimization [11] implemented in SPOT [12] and compare it with other optimization techniques: the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and L-BFGS-B. The region of interest (RoI) settings were obtained by Koch *et al.* [147, 150].

Summarizing, the tunable parameter set contains nine variables, where six values are for the leaky rain pre-processing and the rest for the SVM parameter setting. A radial basis kernel was used as kernel function for the SVM since it gave best results in preliminary runs. The RoI for the variables is shown in Tab. 4.2.

The time series data was split into four consecutive parts of 5000 records each, named Set 1 to Set 4. As training set we chose Set 2 as in previous chapter. For each of the sets the hyperparameters were optimized by the search strategies SPOT, CMA-ES and L-BFGS-B.

Table 4.2: ROI for the stormwater problem

Parameter	RoI	Type	Description
D_1	[2,60]	int	Embedding dimension 1
D_2	[2,60]	int	Embedding dimension 2
$leaky_1$	[0.005, 0.3]	float	Leaky decay parameter 1
$leaky_2$	[0.005, 0.3	float	Leaky decay parameter 2
W_1	[50,120]	int	Leaky window size 1
W_2	[50,120]	int	Leaky window size 2
γ	[0.005, 0.3]	float	RBF kernel width
ϵ	[0.005, 0.3]	float	ϵ insensitive loss-fct.
C	[1,10]	float	regularization term

4.3.1.3 Results

In the earlier experiments described in Sec. 3.1.6 we used the error gathered on the test set as the objective function value for the hyperparameter tuning with SPOT. In the real world this value is unknown and thus gives the tuned model a certain bias towards the test data. In order to perform a fair comparison and to show the benefits of parameter tuning in a more realistic setting, we ran a test using a different objective function for tuning the model parameters. Otherwise the test set error might be biased, since the model has been tuned and tested on the same set. In Tab. 4.3, 4.4, 4.5, we present the results of training a SVM model on Set 2, and using Sets 1, 3, 4 (rows) as validation sets within the tuning. Finally we mutually evaluated the tuned models on all these sets (columns) with SPOT, CMA-ES and L-BFGS-B respectively.

It can be seen that SPOT outperforms all other tuners. The bad result of the quasi-Newton search method L-BFGS-B can be explained by the fact that the method's appropriateness is somewhat dubious considering the fact that finite difference approximations have to be calculated for the gradient and the fitness function is non-differentiable from a theoretical standpoint as it involves integer parameters. It can also be suspect to premature local convergence. On some sets the CMA-ES also produces good results but has to cope with oversearching issues.

Especially the results obtained from the SPOT tuning are better than domain-specific models such as the INT2 model, whereas a positive answer can be given for research question **Q1**. Other tuning algorithms like CMA-ES or BFGS also improve the generalization error, but certainly require more function evaluations or restarts. For the CMA-ES and BFGS tuning algorithms the RMSE is considerably lower when validation set (for tuning) and test set are the same. This can be easily seen by having best values always on the diagonal of the tables, indicating too optimistic tuning results. This means that parameter tuning can lead to oversearching effects, which coincides with research question **Q2**. We quantify this

Table 4.3: Results of SPOT tuning on the stormwater problem. In each row of the table, SPOT tunes the RMSE on validation set 1, 3, 4 leading to different SPOT-tuned parameter configurations. These configurations were applied to the test sets 1, 3, 4 (columns) to make the results comparable. Note that set 2 is missing here, because it was used for training only. Each experiment was repeated five times with different seeds and we show the mean RMSE; bold-faced numbers are best values on the test set and the numbers in brackets indicate standard deviations.

		Test		
		Set 1	Set 3	Set 4
Validation	Set 1	**9.03** (0.66)	15.13 (0.71)	11.63 (1.25)
	Set 3	14.35(2.41)	**9.57**(2.29)	14.42 (3.58)
	Set 4	10.87 (0.15)	12.92 (0.36)	**7.27** (0.51)
	S_t	12.61	14.03	13.02
	V_t	39.7%	46.5%	79.1%

effect by evaluating the following formula: let R_{vt} denote the RMSE for row v and column t of Tab. 4.3, 4.4, 4.5. We define

$$V_t = \frac{S_t - R_{tt}}{R_{tt}} \quad \text{with} \quad S_t = \frac{1}{2}\left(\sum_{v=1}^{3} R_{vt} - R_{tt}\right) \tag{4.7}$$

With S_t we evaluate the mean off-diagonal RMSE for the columns $t = (1, 2, 3)$ which is an indicator of the true strength of the tuned model on independent test data. The diagonal elements R_{tt} are considerably lower in each column of Tab. 4.3, 4.4, 4.5. In case of no oversearching, a value of V_t close to zero would be expected, whereas values larger than zero indicate oversearching.

Table 4.4: Same structure as in Tab. 4.3, but with CMA-ES as tuning algorithm.

		Test		
		Set 1	Set 3	Set 4
Validation	Set 1	**8.07** (1.13)	17.0 (2.37)	13.61 (2.25)
	Set 3	17.90 (2.20)	**9.40**(3.38)	16.55 (1.49)
	Set 4	11.04 (0.24)	13.87 (1.40)	**7.48** (1.48)
	S_t	14.47	15.43	15.07
	V_t	79.4%	64.3%	101.5%

The SPOT tuned learning algorithms always perform best when validation set and test set are the same (diagonals), but the oversearching effect is rather small. However, the best results obtained by CMA-ES and L-BFGS-B seem to be too optimistic, because although none of them has the lowest RMSE on the diagonals, they perform clearly worse on the off-diagonals. Instead, SPOT gives the smallest RMSE on the off-diagonals, just with one exception on validation set 3 and test set 4, where L-BFGS-B is slightly better. This

underlines the robustness of SPOT in tuning learning processes, although a small amount of oversearching is measurable in any optimization method.

Table 4.5: Same structure as in Tab. 4.3, but with L-BFGS-B as tuning algorithm.

		Test		
		Set 1	Set 3	Set 4
Validation	Set 1	**8.32** (1.70)	18.63 (1.99)	13.35 (1.83)
	Set 3	17.32 (2.66)	**10.07**(3.28)	13.39 (3.26)
	Set 4	11.43 (1.17)	15.42 (1.67)	**8.41** (2.73)
	S_t	14.38	17.02	13.37
	V_t	72.9%	69.0%	59.0%

4.3.2 Acid concentrations

The goal of this benchmark is to classify acid concentrations solely from spectroscopy information of a fluid of a biogas plant [257]. In the acid concentration problem the user has defined five classes, each denoting a certain range of acid concentration. For classes $c = (1, \ldots, 5)$ the dataset has imbalanced class members $N_c = (228, 1528, 1880, 731, 70)$ which is depicted in Fig. 4.2. Earlier work for this dataset incorporated GerDA [233], as described by Wolf *et al.* [257]. GerDA is based on Boltzmann machines [229, 111, 15] and learns feature combinations in an unsupervised fashion.

4.3.2.1 Research questions

Q1 Can a tuning of SVM parameters can achieve a similar or even better performance than GerDA [256], the so far best approach?

Q2 Which influence has the optimization budget for the result ($nEval \in \{50, 100, 200\}$)?

4.3.2.2 Experimental setup

The acid concentration problem is a five-class problem, where the user-defined goal is to maximize the mean class accuracy

$$MCA = \sum_{c=1}^{5} \frac{1}{N_c} \sum_{i=1}^{N_c} U(\vec{X}_i) \tag{4.8}$$

where $U(\vec{X}_i)$ is 1 for each correctly predicted record $\vec{X}_i$ and 0 otherwise.

The dataset contains 4437 records with 212 attributes, each attribute denoting a sample point from the spectral curve. Training set (3328 records) and independent test set (1109 records) were defined by Wolf *et al.* [257, 256]. The results in Wolf *et al.* [256] showed that good classification can be obtained if class weights are taken into consideration.

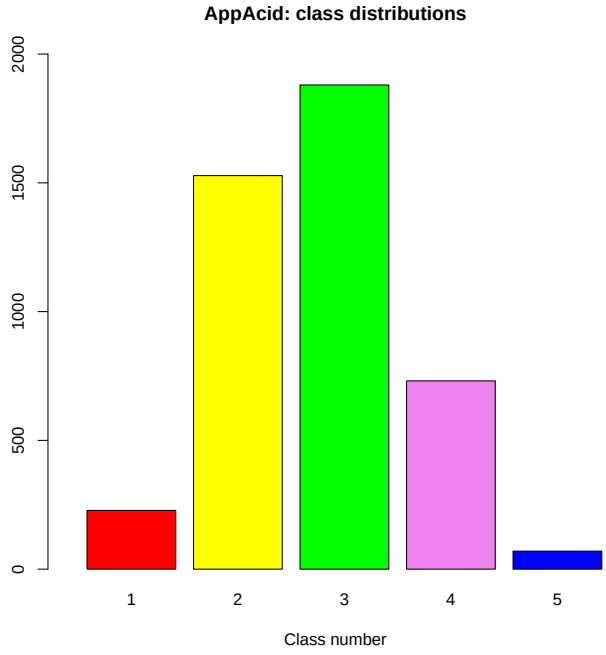


Figure 4.2: Class distribution for the acid concentration problem. Classes 1 and 5 have much fewer instances than the other classes.

Class weights are necessary due to the heavy imbalancing of classes. Fig. 4.2 shows the distribution of the data instances for the five classes. It is desired to classify all classes correct, and here especially class 5 with only 70 records is important. The reason is that class 5 defines a critical plant state and therefore has a higher importance, than e.g., class 3, where a stable plant state is attained.

We performed experiments using SVM as classifier. As an alternative to SVM we tested also Random Forest (RF) which gave similar results [154]. Two simple pre-processing steps were taken into account:

1. The attributes in the dataset show a very high correlation. This is of no surprise, because the attributes stem from a discretization of the spectral curve and thus adjacent attributes have similar values. A PCA was performed in order to reduce the dimensions and to obtain a better class separation. PCA can be optionally switched on in the TDM framework [154] so that no further implementation is necessary.

2. To aid the SVM classifier in finding non-linear feature combinations with relevance to the classification goal we added monomials of degree 2 spanning all combinations of the first $N_{PC} = 8$ principal components with highest eigenvalues.

In Tab. 4.6 the ROI settings for the acid concentration problem are shown, including standard SVM parameters for regularization (C) and the RBF kernel (γ), as well as additional parameters for the class balancing for the acid concentration problem problem and feature selection as a result of the PCA pre-processing. More details on the selected parameters are given in [155].

Table 4.6: ROI for the acid concentration problem

Parameter	ROI	Type	Description
$xperc$	[0.05, 1.00]	float	Fraction of Features Taken as Input
$CUTOFF$	[0.01 0.40]	float	Voting scheme
$CLASSWT$	[0.05 1.0]	float	Class weight vector
γ	[0.005, 0.3]	float	RBF kernel width
C	[1,10]	float	Regularization term

The experiments in Fig. 4.3 were created by repeatedly executing the following procedure five times:

- Each tuner generates an initial design, that is random settings for the hyperparameters within their predefined RoIs created by an optimized Latin hypercube design.
- During tuning only the training set (3328 records) was used. This set was further split randomly into 80% of the data used for training an SVM with hyperparameters provided by the tuner and 20% used for validation, i.e. for measuring the model strength (mean class accuracy) and reporting it back to the tuning algorithm. This random sub-sampling as well as the determination of the infill point leads to non-deterministic variations for the tuner, although the training process of the SVM itself is deterministic. Each design point is evaluated twice ($maxRepeats = 2$) and the tuner aggregates the two results by taking the average.
- A second source of randomness lies in the pre-processing parameter $xperc$: Given the pre-processed inputs a filtering is perfomed by measuring the random forest importance (RFI). Then a greedy selection strategy determines the best variables according to the procedure in Alg. 3. Since the variable ranking is based on RFI, it is also subject to random variations.
- At the end of the tuning process (which is stopped after $nEval \in \{50, 100, 200\}$ model trainings, i.e. $25, 50, 100$ design points), the best set of hyperparameters is

taken, used for a final SVM model training on the full training set (3328 records) and the model strength (mean class accuracy) is evaluated on the 1109 unseen test data records.

Table 4.7: Tunable parameters, their region of interest (RoI) and best results obtained for the acid concentration problem ($N_{eval} = 200$). There were 12 parameters to tune, since $CUTOFF[5]$ was not tuned but set by a sum constraint.

	RoI		Best Value				
Name	Low	High	i: 1	2	3	4	5
$xperc$	0.05	1.00			0.237		
$CUTOFF[i]$	0.01	0.40	0.216	0.023	0.335	0.389	0.232
$CLASSWT[i]$	0.05	1.00	0.849	0.431	0.246	0.468	0.814
γ	0.001	0.80			0.195		
C	0.00	100			54.6		

4.3.2.3 Results

We see from Fig. 4.3 that the SPOT tuning has a slight oversearching effect, is comparable to the so far best GerDA results, and is considerably better than the CMA-ES tuner. Although the SPOT tuning does not clearly outperform GerDA, it can be seen as a promising alternative, since it requires only a fraction of the computation time of GerDA, which ran several days on a high performance computer. Concluding we can state that research question **Q1** is true, although we only achieve small improvements by using SPOT. An interesting result is the variation of the available budget for the tuning. SPOT could attain the result of GerDA with a very small budget of 50 function evaluations. This is a very promising observation for affirming research question **Q2**, meaning that already with small budgets good parameter settings can be obtained.

In an experimental study Konen *et al.* [155] compared the influence of feature processing and selection for the acid concentration problem. Tab. 4.8 shows the influence of feature processing on the prediction performance. Four different pre-processing operators have been incorporated in this study:

- feature construction by PCA,
- calculation of monomials of degree $d = 2$ as additional input attributes (denoted as 'monomials')
- feature selection by random forest importance (denoted by 'RFI')
- and feature selection by a Genetic Algorithm (denoted by 'GA')

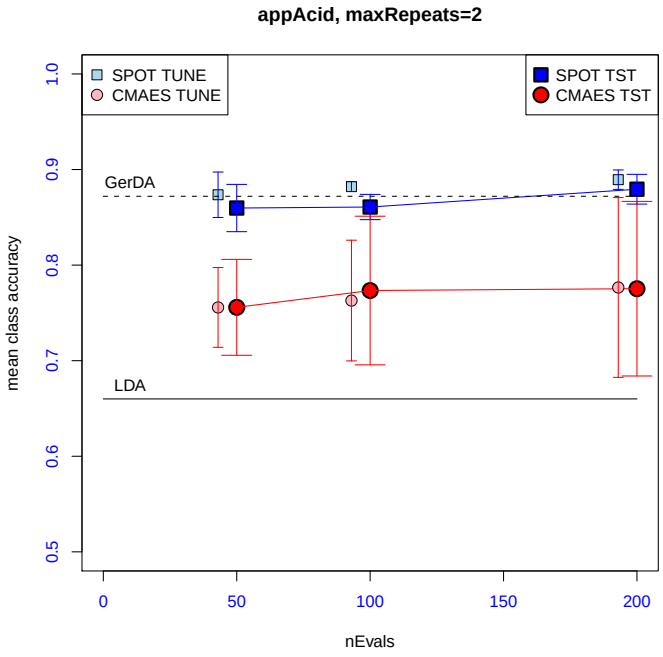


Figure 4.3: Results of SVM tuning in the acid concentration problem . The number of model trainings ($nEval \in \{50, 100, 200\}$) is deliberately set to low values, since this training is the most time-consuming part of the tuning process. Error bars denote standard deviations from 5 repeated experiments with different random seeds. Tuning with SPOT is in all cases better than tuning with CMA-ES. The tuning results for SPOT are slightly higher than the independent test results (oversearching).

The additional feature construction by monomials gives a 3.2% decrease in accuracy, without the PCA a considerable larger decrease (6.5%) can be observed. The largest decrease in classification performance (7.3%) is caused by a turned off feature selection (FS), see line 6 in Tab. 4.8. Here, FS-RFI reduces the feature set to about 40 out of 257 features, while the GA feature selection (FS-GA) selects even less features, between 8 and 19 in the best individuals of five GA runs.

Both feature selection methods reveal comparable performances. This is however only the case if a biased initialization procedure was used to generate the starting population: the first 15 PCA features with the largest eigenvalues were selected with higher probability than the other features with lower eigenvalues. If otherwise a starting population with all features was incorporated, then the GA would usually stop in a local minimum with roughly half of the features selected and with a MCA of only 85%. The advantage of the biased procedure can clearly be seen in the best individuals of five GA runs: The principal components with the highest and second highest eigenvalue are selected in every best individual, and monomials between principal components with highest eigenvalues are also

Table 4.8: Class accuracy MCA of best tuning solution (budget $nEval = 200$) on the acid concentration problem when different feature processing elements are activated. FS-SRF: feature selection based on the sorted RF-importance, FS-GA: GA-based feature selection.

	PCA	monomials	FS-RFI	FS-GA	class accuracy
1	X	X	X	-	$(89.95 \pm 0.41)\%$
2	X	X	-	X	$(89.47 \pm 0.52)\%$
3	X	-	X	-	$(86.72 \pm 0.77)\%$
4	-	X	X	-	$(83.38 \pm 0.78)\%$
5	-	-	X	-	$(82.90 \pm 1.35)\%$
6	X	X	-	-	$(82.60 \pm 0.92)\%$
7	-	-	-	-	$(82.59 \pm 0.42)\%$

selected more frequently compared to other features. Although the biased GA approach was slightly worse compared with RFI, a smaller number of features can be advantageous, e.g., leading to better generalization performances.

4.3.3 Comparing tuned and manual settings

In a further study we analyze, how good hand-tuned paramters are compared with systematically tuned settings. In Table 4.9 we show the CV error of a SVM trained on the gesture classification problem described in Sec. 3.2. Previous results for the same problem are also published in [151]. Although the reported results were not bad in the previous work (CV error rate ranged from 1.54% (using Random Forest) to 3.21% (SFA, $npp = 12$) we already improved with the SVM classifier and hand-tuned kernel parameters ($\gamma = 0.01$, mean CV Error rate from ten runs at 1.37% error). As kernel function the radial basis function (RBF kernel) was used. The CV error of 1.37% was not the optimal value for the SVM. For this reason we applied SPO to tune the γ parameter of the RBF kernel function. Surprisingly the average results with the SPO tuned kernel parameters were improved by over 20% in contrast to the manual approach.

Table 4.9: Comparison of hand-tuned parameters and systematic parameter tuning for the gesture recognition problem (Sec.3.2).

	Best	Mean	Worst	Std.Dev.
SVM[hand]	1.26	1.37	1.67	0.16
SVM[SPOT]	0.7	1.091	1.401	0.22

4.4 Kernel evolution

The importance of appropriate SVM parameters has been shown earlier in this thesis. Other work in SVM tuning mainly includes the optimization of the kernel parameters for fixed kernel functions. Therefore see the overview in Sec. 4.1. Now we want to evolve the kernel

function itself and not only its hyperparameters. For this, Cortes *et al.*[50] proposed a gradient-descent algorithm for learning polynomial combinations of kernels.

Other work is often based on GP performing symbolic regression. As an example Howley and Madden [117] compared standard kernel functions with a tree-based GP approach, which produced better results. Their evolved kernel functions are guaranteed to be symmetric, but can be non positive semi-definite (PSD). This property is required to guarantee the termination of the solver inside SVM. A special fitness function for avoiding overfitting based on margin maximization of the learned hyperplane is used. Diosan *et al.* [60] also use GP for evolving kernels and mind the PSD property the of kernel functions. In their work a larger GP function set compared to [117] is used, but results are only evaluated on few benchmark datasets. Sullivan and Luke [234] respect the PSD property of SVM kernels. They show that GP kernels can improve the accuracy, but remark the large computational overhead produced by the evolutionary process. Gagne *et al.* [89] use a co-evolutionary approach for the kernel evaluation. Kernels obtained by their GP approach are ranked by a nearest neighbour classification algorithm instead of SVM, since the PSD property cannot be guaranteed with the chosen function set. Summarizing these publications we can conclude that good results can be obtained using tuned or evolved kernels, but special care must been given to

- the PSD property of the kernel functions,
- the fitness function, e.g., the evaluation of intermediate kernels,
- the runtime of the algorithms.

Also, by either considering a smoothed version of the CV error [139] or a likelihood function of the hyperparameters [95] one can switch to fast gradient-based methods. Although all cited authors in this section usually compare their algorithms to one or a few alternatives we try to give a comprehensive study including all approaches.

4.4.1 Research questions

Q1 Is it possible to improve the prediction quality of a SVM learning algorithm, by replacing standard kernel functions by means of Genetic Programming (GP)?

Q2 Do GP optimized kernel functions differ from standard kernel functions?

This work is based on the analysis of Koch *et al.* [148] using a more complex approach based on symbolic regression. We investigate the possibility to use GP as a symbolic regression technique for evolving kernel functions. The main reason to choose GP was, that it is a generating technique with large degrees of freedom, and that it has become more and more applicable along with the increase in computational power in the last years.

Earlier work included three possible options to optimize a set of base kernels $k_1, k_2, ..., k_l$:

1. Optimize the composition kernel k_μ with an optimization strategy as ES by learning a linear combination of the base kernels:

$$k_\mu(x,y) = a_1 k_1(x,y) + a_2 k_2(x,y) + ... + a_l k_l(x,y) \tag{4.9}$$

   where $a_1, ..., a_l$ are real numbers

2. Optimize a set of kernel functions by Genetic Programming: The kernel base set is passed into a GP type system. GP learns the new kernel by means of mutation and recombination. Note that base kernels may be changed.

3. A non-linear learning of a kernel system. In this case a non-linear (!) function (polynomial) is learned by GP to optimize the kernel function, the base kernels itself are not changed.

In the earlier approaches, basic SVM kernel functions were optimized by tuning the kernel parameters, e.g., γ and C. This can be already seen as an advantage over simple hand-tuning. E.g., the authors of the libsvm package recommend to start with a simple kernel selection and tune via grid search [118]. Now we aim at evolving kernel functions, to obtain specific kernel functions for new datasets. In this work, we use GP as an evolutionary system for improving existing kernel functions.

4.4.2 Kernel evolution with Genetic Programming

We consider GP as a function optimizer [89] for evolving kernel functions of SVMs. The goal is to use GP to detect new elements for kernel machines, which is an important part in defining a kernel-based learning algorithm. The data and control flow of the GP algorithm used in this work is shown in Figure 4.4.

GP has been succesfully applied for symbolic regression tasks earlier, especially for producing interpretable feature sets, e.g., using Pareto GP [226, 225, 157]. Instead of limiting the tree depth Pareto GP could be used to explore symbolic kernel functions with varying complexity. However, this is another field of research and exceeds the scope of our aim.

The difficulty is to guarantee the positive semi-definite property of the kernel function [180, 235]. Cortes *et al.* [48, 49] presented some properties of kernel functions and describe the closure of kernel functions. In a more recent work Cortes *et al.* [50] analyzed the idea of creating ensembles of kernel functions combined with a non-linear weighting scheme. Our objective is to accomplish well suited non-linear weights by learning a weight function with a tree based GP system written in R called 'RGP' [80].

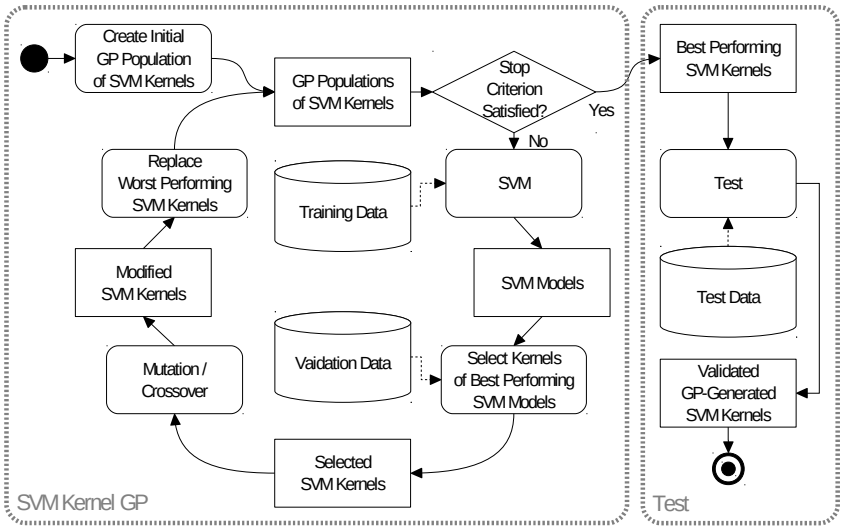


Figure 4.4: General outline of kernel evolution by GP: Starting with a randomly generated population of positive definite SVM kernel functions represented as symbolic expressions, the algorithm selects a random subset, termed a tournament set, for evaluation. Each kernel in the tournament set is employed in training a SVM model on a training dataset. These models are then evaluated on an independent validation dataset. The kernels of the best performing SVM models in the tournament are then modified by mutation and crossover. The resulting kernels and replace worse performing kernels from the tournament set in the population. This loop is repeated until a stop criterion is satisfied. In the final test stage, the best performing kernels are validated on an independent test dataset and returned as an result.

4.4.2.1 GP search space

Tab. 4.10 defines the strongly typed GP search space for SVM kernel evolution. The used building blocks (constants, input variables, and operators) allow the definition of standard support vector kernels like linear, polynomial and RBF kernels, with the only exception of the sigmoid kernel. We omitted the inclusion of trigonometric functions to keep the search space reasonably simple. The search space is further restricted by an expression tree depth limit of 10 levels.

It can be easily seen that the strongly typed GP search defined in Tab. 4.10 includes kernels that violate the PSD criterion. We therefore reduce the search space further by only accepting individuals that comply to the following criteria:

- A valid kernel expression k must contain both input vectors $\vec{X}_i$ and $\vec{X}_j$.
- k must contain at most 4 symbolic constants.
- The kernel matrix M_k of k must not contain numerical problems.
- M_k's eigenvalues must be non-negative (up to a small numerical margin of $1e-16$), i.e., M_k must be positive semi-definite.

Table 4.10: Strongly typed GP search space for SVM kernel evolution. All possible building blocks for each building block class are shown. The variables $\vec{x}$, $\vec{x}_i$ and $\vec{x}_j$ denote real vectors of the SVM input data dimension D, the constants c_1, c_2, c_3, c_4 denote symbolic constants. The types of building blocks are given as type expressions, e.g., $\mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ is the type of an operation that maps two real vectors to a scalar real value. A center dot $\cdot$ denotes a placeholder for a building block of suitable type. The division operator is protected division. Note that some operators, such as the $\times$ operator, are "overloaded" to work with multiple operand types.

Building Block Class	Type	Building Blocks
Constant	$\mathbb{R}$	$\{c_1,\ c_2,\ c_3,\ c_4\}$
Input Variable	$\mathbb{R}^D$	$\{\vec{x}_i,\ \vec{x}_j\}$
Scalar Function	$\mathbb{R} \to \mathbb{R}$	$\{\exp(\cdot),\ \cdot^2\}$
Scalar Operator	$\mathbb{R} \times \mathbb{R} \to \mathbb{R}$	$\{\cdot + \cdot,\ \cdot - \cdot,\ \cdot \times \cdot,\ \cdot \div \cdot\}$
Vector Operator	$\mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}^D$	$\{\cdot + \cdot,\ \cdot - \cdot\}$
	$\mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$	$\{\cdot \times \cdot,\ \|\| \cdot - \cdot \|\|^2\}$
	$\mathbb{R} \times \mathbb{R}^D \to \mathbb{R}^D$	$\{\cdot \times \cdot\}$
	$\mathbb{R}^D \times \mathbb{R} \to \mathbb{R}^D$	$\{\cdot \times \cdot\}$

Table 4.11: General settings for all kernel evolution by GP experiments.

Setting	Value
Algorithm	Strongly Typed GP
Total Budget (Evaluations)	2500
Design Factor	3
Population Size	20
Tournament Size	8
Population Initialization	Type-safe Random Growth
Type-safe Variation Operators	Replace Function Label, Replace Node by New Subtree, Uniform Subtree Crossover
Breeding Tries	5,000
Extinction Prevention	True (retries up to a time limit of ten minutes)
Number of Runs (n)	20

A breeding strategy is applied to the population initialization and individual variation operators to ensure with high probability that only individuals conforming to these criteria are present in the population. When an individual is created or modified by a genetic operator, the result is checked for compliance. If it violates one of the criteria, the operator is retried, up to maximum number of tries given as the algorithm parameter "Breeding Tries". If a genetic operator exceeds this limit, its last result is returned. This means that the population can contain individuals that may not comply to our constrains.

Fig. 4.5 introduces the typed GP variation operators used in this study by means of examples. Formal definitions of these standard operators are given in [193]. The parameter settings of the GP system are listed in Tab. 4.11.

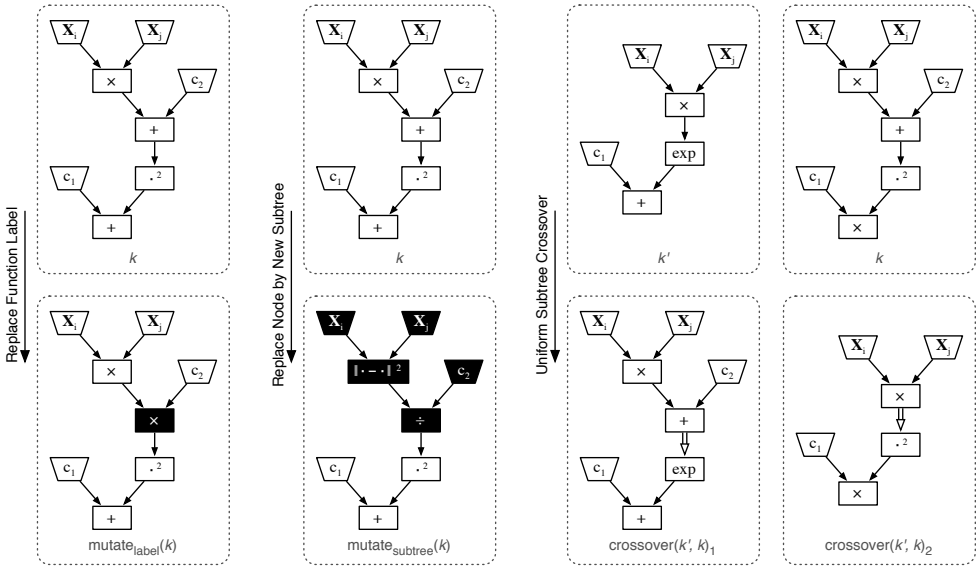


Figure 4.5: Standard typed GP variation operators explained by means of example: Two different mutation operators and a single crossover operator are employed. The first mutation operator "Replace Function Label" chooses a node by uniform random selection to be replaced by a node of matching type. Nodes affected by a variation operator are shown in inverse (white-on-black). The second mutation operator "Replace Node by Subtree" selects a node by uniform random selection to be replaced by a newly initialized subtree. These mutation operators enable the GP algorithm to reach every well-typed expression. The crossover operator "Uniform Subtree Crossover" chooses an edge as crossover point by uniform random selection in each of the two parent trees, then swaps the subtrees at the crossover points between the parents, creating two children trees. Edges at crossover points are shown as white arrows.

4.4.2.2 GP fitness function

The quality of an individual k, which denotes the quality of the symbolic representation of a support vector kernel in our case, is evaluated by our fitness function in four steps:

1. It is checked if k complies to our criteria defined above. If it does not, a bad fitness $(+\infty)$ is assigned to k. This step is necessary because invalid individuals resulting from failed breeding attempts might be present in the population.

2. As the performance of a kernel is often highly sensitive to its constants as well as to the regularization parameter C, a Latin hypercube design (LHD) D of size $d \cdot (\mathrm{nconst}(k) + 1)$ is created by Latin hypercube sampling, where $\mathrm{nconst}(k)$ is the number of constants in k and the design factor d is given as an algorithm parameter.

3. For each design point in D, C as well as the constants of k are set accordingly and the performance of a Support Vector Machine is evaluated based on a single validation holdout set (1/3 of the training data created in the outer resampling), giving a vector

$\vec{Y}$ of performance values. Performance is measured by mean misclassification error (MMCE).

4. The pointwise minimum of $\vec{Y}$, i.e. the performance of k based on the best design point in D, is returned as k's fitness.

The description above applies to classification problems. For Support Vector Regression, we add another parameter ϵ to the LHD and change the performance measure from MMCE to root mean squared error (RMSE). See Sec. 2.1.2.3 for a definition of ϵ.

4.4.2.3 Evaluation and performance measures

The evaluation of a Support Vector Machine concerns two different stages. While we need to be able to compare different kernels or parameterizations during model building, we also have to evaluate the combined fitting and optimization process itself. The latter is straightforward: After we have decided upon a specific kernel function and its parameters by training using sub-sampling of training data, we evaluate on the remaining observations (holdout set) to avoid optimistic bias. This process is repeated a couple of times in order to maximally utilize the limited amount of data. The performance measure for evaluation is dependent on the target of the application and will be closely connected to the loss function chosen in Eq. 2.11. For classification often the misclassification error (i.e., 0/1-loss) and for regression either the root mean squared error (RMSE) or the mean absolute deviation is selected. To compare kernels or parameterizations during model selection we could in general do the same (resulting in nested resampling), and this approach is followed by quite a lot of authors. But one should be aware that especially for kernel construction by Genetic Programming a large number of these evaluations will be needed. Therefore, if possible, less expensive alternatives should be taken into account. One is to use fast implementations for calculating the leave-one-out error. The other is to exchange the natural performance measure during model selection by one which is computationally cheaper, see Duan *et al.* [64] for a comparison of possibilities.

4.4.3 Kernel closure properties

Kernel functions compute dot products in high-dimensional spaces without explicitly mapping into these spaces. A kernel function must satisfy several mathematical properties so that Mercer's theorem [176] holds. One property is that kernel functions must be positive semi-definite (PSD). Kernels stay PSD under certain operators. Let us assume we have two kernels k_1 and k_2 which are PSD. Then the following kernel functions are also PSD:

- Closure under sum: $k_1(\vec{X}_i, \vec{X}_j) + k_2(\vec{X}_i, \vec{X}_j)$
- Closure under product: $k_1(\vec{X}_i, \vec{X}_j) \cdot k_2(\vec{X}_i, \vec{X}_j)$

- Closure under positive scalar multiplication: $a \cdot k_1(\vec{X}_i, \vec{X}_j)$, with $a > 0$
- Closure under exponentiation: $\exp(k_1(\vec{X}_i, \vec{X}_j))$

For a more detailed description of closure properties we refer to Cortes *et al.* [47, 49]. However, Schölkopf [214] showed that even when the PSD condition is violated, some non-PSD kernel functions can be used successfully in practice for learning tasks. Probably the most well-known kernel function, which is not strictly PSD, is the sigmoid kernel:

$$k(\vec{X}_i, \vec{X}_j) = \tanh(\alpha \vec{X}_i^T \vec{X}_j + C) \tag{4.10}$$

Vapnik [245] showed that this kernel is not PSD for some settings of α and C. It is of large interest, how results are affected, if one uses a PSD kernel satisfying Mercer's theorem, or if one uses a non-PSD kernel. A kernel obtained by evolutionary techniques such as GP can of course easily destroy the PSD property. The variation operators can change a PSD kernel into a non-PSD kernel, depending on the chosen GP function set. As non-PSD kernels often also create technical problems for the employed SVM optimizer, we try to maintain the PSD property in the evolutionary process. This is achieved by incorporating special variation operators which respect the closure properties. These operators are implemented through a combination of strongly typed GP and breeding.

4.4.4 Experimental analysis

In this section first results of our support vector kernel evolution scheme for different datasets are reported. To make it possible to compare our results with earlier work in the field of SVM kernel evolution, we used datasets from the UCI machine learning repository [84] for evaluation. These results are linked to the parameter tuning approach discussed in this chapter by comparing them with results obtained with tuned RBF kernels (tRBF). In the tRBF experiments, the parameter for regularization (C) and the RBF kernel parameter (γ) were simply tuned by a grid search with discrete settings $2^{-8}, \ldots, 2^{-7}, \ldots, 1, \ldots, 2^8$ for both C and γ. In the GP experiments we used a single holdout set (33% length) for validation during kernel evolution, while in the tRBF experiments we used 5-fold cross validation during tuning. In both experiments, the reported test performance is obtained by 20-fold sub-sampling (80% training, 20% test).

Tab. 4.12 shows an overview of the test performance of evolved kernels (GP) and grid search tuned RBF kernels (tRBF), measured by the mean misclassification error (MMCE) on UCI datasets. They are also shown graphically in Fig. 4.6. The test performance of GP-generated and tRBF kernels is generally quite similar on all considered datasets. We tried a further tuning for the parameters of the best GP-generated kernels as well as for the regularization parameter (C) via a much larger design after each GP run was finished, which gave only marginal improvements.

Table 4.12: Results obtained on test datasets with GP evolved custom kernels ($N_{eval} = 2500$, $N_{runs} = 20$) compared with results obtained with grid search tuned RBF kernels (tRBF) on UCI test datasets. All test problems use mean misclassification error (MMCE) as performance measure.

	Dataset	Best	Mean	Worst	SD
GP	Ionosphere	0.0282	0.0845	0.1831	0.043
	Glass2	0.1212	0.2545	0.3939	0.075
	Heart	0.0556	0.1516	0.2593	0.049
	Liver	0.1884	0.2929	0.3623	0.045
	Pima	0.1883	0.2300	0.2727	0.024
tRBF	Ionosphere	0.0282	0.0599	0.1127	0.023
	Glass2	0.0606	0.2015	0.3333	0.074
	Heart	0.0926	0.1593	0.2593	0.051
	Liver	0.2029	0.3043	0.4493	0.053
	Pima	0.1688	0.2403	0.3117	0.033

Tab. 4.13 shows the best kernels found in 20 GP runs, i.e., the kernels giving the best performance on validation data. Note that standard kernels like the linear kernel and polynomial kernels are rediscovered by GP search.

Table 4.13: Best GP evolved kernels for UCI test datasets found in 20 runs ($N_{eval} = 2,500$). These kernels where selected based on their performance on validation data.

Dataset	Best GP Evolved Kernel
Ionosphere	$k(\vec{X}_i, \vec{X}_j) = c_1 + \frac{(\vec{X}_i^T \vec{X}_j)^2}{[((c_2(\vec{X}_i^T \vec{X}_i))\vec{X}_j)^T \vec{X}_j]^2}$
Glass2	$k(\vec{X}_i, \vec{X}_j) = [\vec{X}_j^T((c_1(\vec{X}_i^T \vec{X}_i))\vec{X}_j)]^2 + (\vec{X}_i^T \vec{X}_j)^2$
Heart	$k(\vec{X}_i, \vec{X}_j) = \vec{X}_i^T \vec{X}_j$
Liver	$k(\vec{X}_i, \vec{X}_j) = c_1 + (c_2 \vec{X}_i^T \vec{X}_j)^2$
Pima	$k(\vec{X}_i, \vec{X}_j) = c_1 \vec{X}_i^T \vec{X}_j$

4.5 Conclusions

We conclude this chapter of tuning in machine learning with a summary about the most important results.

4.5.1 Tuning of learning algorithm parameters

The optimization of algorithm parameters has a large impact on the prediction accuracy in ML. The biggest concern in performing tuning is probably the high computational costs caused by the evaluation procedure. Learning processes should be evaluated by unbiased evaluations of the prediction model. Here, the most time-consuming part is the training of the learning algorithm. But also other parts like pre-processing can increase the computational

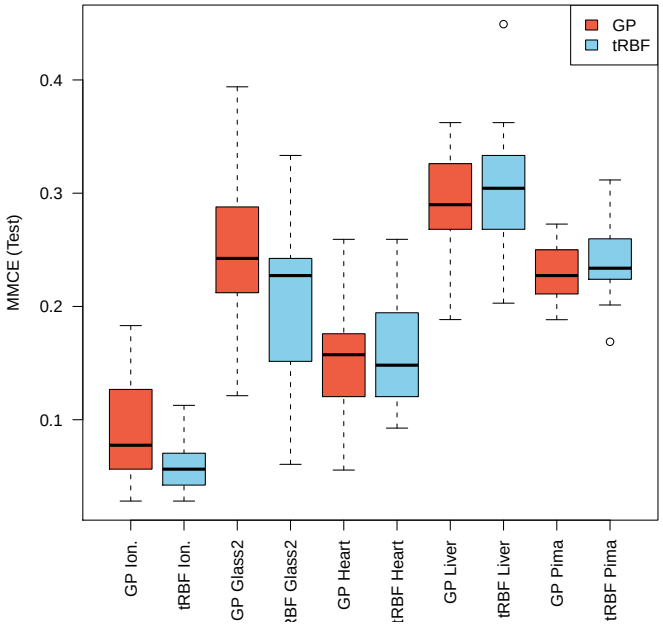


Figure 4.6: Boxplots for comparing the test MMCE of GP evolved custom kernels (GP) ($N_{eval} = 2,500$, $N_{runs} = 20$, no tuning) with the test MMCE of grid search tuned RBF kernels (tRBF) on UCI test datasets.

costs. For this reason a substantial requirement for tuning is, that the number of function evaluations does not exceed a certain limit or budget. This budget is usually defined by the user and can vary from task to task. Due to the large-scale data prevalent today, learning tasks have become more complex, making tuning of learning processes sometimes infeasible. Here, methods like SPO or EGO can be a solution towards systematic parameter tuning, because they require only a small number of function evaluations. Instead a surrogate model of the parameter space is learned, which enables to perform the optimization on the cheaper surrogate.

However, another burden remains, that is that non-deterministic outputs occur. E.g., when a ML process is evaluated twice, the evaluations can differ. The reason is that although the learning algorithm itself can produce deterministic outputs (like a SVM), the selection of training and test data of the unbiased evaluation leads to non-deterministic objective function values. A solution to this can be to incorporate specific surrogate models, which are able to handle uncertainties within the surrogate model. Here, the re-interpolation approach by Forrester [82] has been mentioned for Kriging surrogate models, as well as to use replicated evaluations as known from DoE and DACE. We think that the re-interpolation

can lead to a better exploration, but nevertheless keeps stable results on very noisy tasks.

The resampling of data in the tuning of learning algorithms is important to avoid biases towards some specific data. Oversearching or overfitting effects can be handled by this, in order to produce good generalizing models. In Sec. 3.1 the problem of oversearching has been introduced and has been later discussed in Sec. 4.3. In the latter case, methods for avoiding oversearching have been incorporated, which were based on resampling using holdout sets for both validation and test data.

The tuning itself can be seen as an important step for comparing prediction models. Model selection is only possible for tuned learning algorithms, which is one of the main reasons why tuning must be integrated in the learning process. Besides the tuning of learning algorithm hyperparameters, we have also shown how learning algorithms are influenced by other parts of the learning chain, e.g., the feature processing. As a main contribution, we propose to integrate all parts of the learning process in an outer tuning procedure and not to decouple any part from the rest of the learning process.

In a detailed experimental study we showed advantages of tuning of learning processes. A prediction model for time series regression based on SVM outperformed special-purpose models created by experts. The tuning of the parameters of this SVM model was performed by a model-assisted optimization procedure devoted to the field of efficient global optimization (EGO). EGO starts with an initial distribution of design points, that is a set of parameters for the learning process. This initial design is evaluated on the real objective function, which incorporates the training of a prediction model and the evaluation on some unbiased validation data. The error obtained on the validation data is taken as performance of the parameter setting.

The tuning of standard support vector kernels by various optimization methods gave improvements over hand-tuned parameter settings and rule-of-thumb values. Here, especially the SPOT-tuned SVM kernels showed a remarkably better prediction accuracy than the other tested optimization algorithms like CMA-ES or derivative-free search. While this is not too surprising for the quasi-Newton search, the CMA-ES case deserves perhaps further discussion. We think that two reasons might be responsible for this:

1. The relatively small number of function evaluations (up to 200) is not favorable for the CMA-ES mechanism which needs usually more function evaluations to adapt its covariance matrix.

2. Another problem for CMA-ES is the existence of relative tight RoI-borders in our tuning problems. If a border constraint is violated, CMA-ES adds a penalty term for solutions violating the border constraint, which often leads to a minimum exactly at the RoI border and lets the CMA-ES solution stick there. The probability to get stuck at such a border minimum can be reduced with frequent restarts [4] of CMA-ES (a feature which is not yet available in the current CMA-ES R-package), but this is not

likely to help in our case, since restarts would further diminish the number of function evaluations available for each start.

It is a nice feature, that SPOT is not affected by both effects and can therefore lead to good results with a relatively small number of function evaluations. This has the added benefit of modest computation times, making this approach feasible for use in real-world problems like the acid concentration problem and the stormwater problem. Computation times might be further reduced by using a smaller subset of the training data for parameter tuning. The robustness of parameter settings obtained on reduced training set sizes is discussed in Chapter 5.

4.5.2 Evolution of SVM kernels

The evaluation of the GP-based system for kernel evolution results in less favorable results. First of all it should be mentioned that a non-trivial technical overhead is involved in building such a system, as a (potentially strongly-typed) GP toolbox needs to be coupled with a SVM implementation, which allows for custom kernel functions. In our case we took specific care to generate only PSD kernels, but numerical problems still resulted in occasional "freezes" of the SVM optimizer [136], making it necessary to include a mechanism to stop these processes in our evaluation framework (extinction prevention).

Regarding the optimization results it is certainly interesting that our GP system can data-dependently recover default kernels, but no significant improvements over a default approach with a RBF kernel are obtained. With this result we have to negate research question **Q2**, stating that other kernels than default kernels can be obtained using GP. In the cases where the GP system performs slightly better, most of the time a linear kernel is recovered, which could have been easily included in our default evaluation. And the computational cost of GP is still a considerable burden even for smaller datasets. Furthermore, the default kernels are usually tried first for a reason in SVM modeling, as they correspond to either polynomial decision boundaries or a local model with data-dependently placed radial basis functions, which are appropriate for many problems.

We would like to point out that our results are qualitatively on the same scale as the results of other authors of comparable papers [89, 60], but we come to a more cautious interpretation when comparing to default kernel results. Although we find the contributions of the afore mentioned authors very valuable and interesting in their own right, we would like to discuss a few examples which illustrate why their results might look better in their comparisons:

1. Diosan *et al.* [60] only mention that the default kernels are optimized, but give no details. For the RBF kernel they report an accuracy of about 80.5% on the Ionosphere data set, while one of their GP variants achieves about 91.5%. The last result is

comparable with ours, while the former is not. A simple kernel parameter tuning by grid search produces an accuracy of about 94%.

2. Gagne *et al.* [89] always set the bandwidth of their default RBF kernel to 10 and only tune the C parameter. It is highly unlikely that this setting is best across the different datasets they evaluate. They achieve significantly worse test set performance for all three datasets that are common to both their and our study (Ionosphere, BUPALiverDisorders, PimaIndiansDiabetes) for the RBF kernel than in our comparison experiment. It should also be noted that they use a kernel-nearest-neighbor classifier instead of a SVM in their GP system; a reason for this might have been the technical difficulties we mentioned above.

3. Sullivan and Luke [234] report extremely narrow confidence intervals for their error estimations, leading to significant improvements although the differences in mean performance are quite small. This includes the conclusion that their kernel evolution achieves a significantly better performance even on the Iris data set. Since it is hard to verify how they achieved these values (our error estimators have much larger standard deviations), we would like to mention the well-known fact that for resampling (like cross-validation) the obtained error samples have a non-trivial dependence structure, leading to an underestimation of the standard deviation. One should also keep in mind the well-known distinction made in statistics among significant differences and relevant ones. We further tested their best obtained non-default kernels for Ionosphere, for which they report an accuracy of more than 98%. We found that it did not perform significantly better than a tuned RBF kernel in an unbiased estimation.

Although we could not find proofs to corroborate research question **Q1**, we still assume that GP-based discovery of custom kernels might give significant improvements on problems that are more difficult to solve with standard kernels, such as time series prediction and classification problems or problems from functional data analysis. As these problems often involve a large amount of data, at least when compared to our UCI test datasets, it might be beneficial or even necessary to examine and optimize the effectiveness of the GP search process on the search space of PSD kernels. A first casual observation of the GP search process and of the genotype of the best performing individuals gives the impression of a highly random navigation of the search space, probably caused by low causality of the genetic variation operators. In other words, a single variation (mutation or crossover) step often leads to a drastic change in kernel behavior and performance, rendering efficient evolutionary search nearly impossible. A redesign of the GP search space for PSD kernels that takes causality into account explicitly might help to alleviate this problem [206, 62, 251].

Chapter 5

Improving the efficiency of tuning

In this chapter we describe approaches, which can diminish the computation time of parameter tuning. As tuning can be very expensive for complex ML tasks, we want to pay more attention to the effectiveness of the methods. In general we analyze three different approaches for saving computation time, that is

1. Tuning with limited budgets
2. Efficient sampling during the optimization
3. Automatically setting the number of repeated objective function evaluations within SPO, by using optimal computing budget allocation (OCBA) [39]

While the first and third approach aim at saving computation time by the optimization procedure, the second approach is proposed for diminishing the computation time of the objective function. Nevertheless all three approaches can be efficient solutions towards practical use of parameter tuning.

We give a brief summary about tuning with limited budgets in Sec. 5.1. As a new investigation proposed in this thesis we analyze the effect of sub-sampling during tuning on the classification accuracy in Sec. 5.2. In Sec. 5.3 we introduce an approach based on the optimal computing budget allocation (OCBA) to reduce the number of function evaluations.

5.1 Tuning with limited budgets

The tuning of hyperparameters and pre-processing parameters can be expensive in terms of required computation time. Usually, feature selection approaches, as the wrapper approaches described in Sec. 2.2.2 or other tuning runs, need many model evaluations. As a consequence, tuning is seldom performed in practice or parameters are just set by hand.

An often used strategy to limit the computational budget is to limit the evaluations for the optimization procedure. In Sec. 4.3.2 we performed a series of experiments on the acid concentration task. It showed that the budget has an effect on the tuning results. For

SPOT, better results were obtained, when the budget was increased. The opposite was the case with CMA-ES, where the error on the test data suddenly became worse compared with lower budgets. The number of necessary iterations for a tuning task has to be given for each problem. In general, limiting the budget is not recommendable, since this usually impedes the optimizer to converge too early. Nevertheless computation time can be saved using model-assisted optimization, like efficient global optimization (EGO). Here only a few parameter sets are evaluated on the objective function, but the main part of the optimization is performed on a surrogate model, or response surface methodology (RSM), of the real objective function. Konen *et al.* [155] showed, that using these strategies very restrictive budgets like 50–200 evaluations for more than six parameters are necessary to calculate competitive parameter sets.

5.2 Efficient sampling for tuning experiments

Although Computational Intelligence (CI) methods can generate good and robust solutions for global optimization problems, it is known that they sometimes require a large number of function evaluations. Unfortunately, ML tasks are usually very expensive to evaluate and quick solutions are requested by the users. In this section we investigate how computation time can be saved in order to make CI methods more applicative for ML tasks.

5.2.1 Research questions

We propose the following hypotheses:

Q1 Are the results of parameter tuning more subject to noise, when smaller fractions X of the training data are used?

Q2 Are the optimal design points found by an efficient optimization strategy nevertheless competitive (as long as the training set size is not *too* small)?

The question **Q1** aims at analyzing the variance of the tuning results, when the training set size is decreased. If **Q1** can be answered affirmatively, we should be able to measure this effect directly by an increased variance of the prediction error. If we can answer **Q2** affirmatively, considerable computation speedups in tuning ML models should be possible, without a too high decrease in generalization performance.

5.2.2 Related work

Previous work in analyzing the effects of the chosen sample size has been mainly done in fields like statistics and machine learning. A good overview of different strategies for data sampling can be found in Cochran [44]. A description of resampling methods for optimization in data mining has been given by Bischl *et al.* [18]. Raudis and Jain [202] and Jain and Zongker [125] discuss the influence of sample sizes on the results of feature

Table 5.1: Datasets used for the training set size experiments.

Dataset	Records	Min. Training Size	Max. Training Size	Validation and Test Size	Number of Parameters
Sonar	208	8	124	41	2
Glass	214	8	128	42	2
Liver	345	13	207	69	2
Ionosphere	351	14	210	70	2
Pima	768	30	460	153	2
AppAcid	4400	176	2640	880	12
DMC-2007	50000	2000	30000	10000	7

selection, which is in a certain way related to the parameter optimization task in this section. In statistics, sample sizes are frequently discussed in terms of statistical studies like significance tests [164]. In machine learning, sampling strategies like the bootstrap [66] have been well analyzed and are often applied in practice. However, to our knowledge no study exists where the size of the underlying training data is diminished during parameter tuning. The analysis in this chapter is based on the work of Koch *et al.* [149].

5.2.3 Experimental analysis

In this section we describe the experimental setup and the methodology of our empirical study.

5.2.3.1 Datasets

All experiments presented in this study were performed using datasets from the UCI repository [84], which can be regarded as the simpler datasets, the DMC 2007 data from the Data Mining Cup 2007 [33], and the real-world dataset for predicing acid concentrations which has been introduced in section 4.3.2. In Tab. 5.1 an overview about the datasets and their sizes is given. We present the minimal training set size for each dataset, and also the sizes of the test and validation sets. Each model is evaluated a) during tuning on the validation data and b) after tuning on the independent test data. The sizes for the test and validation sets are equal and stay constant all the time at 20% of the total dataset size. In Tab. 5.1 these sizes are given as *Validation and Test Size*.

Every time a fixed set of 20% of the patterns was set aside prior to tuning for testing purposes. From the remaining 80% of the data (subset D_{train}), we use a fraction X from $X_{min} = 5\%$ to $X_{max} = 75\%$ for training and a fraction of 25% for validation (which is 20% of all data). See Tab. 5.2 for illustration. At the end of tuning a best design point is returned. Using this best design point, we ran a final "full" training with all data in D_{train} (80%) and evaluated the trained model on the test set (20%). Since the training, validation and test sets were drawn at random we repeated all of our experiments ten times.

Table 5.2: Splitting of data. We use fractions from 4% to 60% of the data for model training, 20% for validation during the tuning and the remaining 20% for independent testing.

Training $\longleftrightarrow$	not used	Validation	Test

While a first benchmark of tuning algorithms has been performed by Konen *et al.* [155], it remained unclear, whether the results also hold for smaller training set sizes. Now we also compare the sequential parameter optimization toolbox (SPOT) as a tuning algorithm with a Latin hypercube design (LHD) as a simple, but robust sampling heuristic. LHD is based on the following procedure: We chose random and roughly equally distributed design points from the region of interest and evaluated the design three times. Again, the best point is taken for the 'full' training as above.

5.2.3.2 Tuning setup

SPOT can be controlled through various settings, which have to be adapted slightly for each task. We briefly present our settings for the experimental study here (Tab. 5.3). With 150 function evaluations for the UCI experiments we chose a rather large number of evaluations compared to the other (real-world) datasets. Our aim was to achieve a good and clear convergence to an optimum. For this reason we considered to analyze simpler datasets first, since complex datasets require much more time for such experiments. Nevertheless we also set a number of 200 function evaluations for AppAcid, which proved to be a good and sufficient number in one of our last studies [155]. It has to be noted that the dimensionality of the parameter space for AppAcid is 12, while it is only 2 for the UCI benchmarks.

Table 5.3: SPOT Configuration

Setting	UCI	AppAcid
function evaluations	150	200
initial design size	10	24
sequential design points	3	3
sequential design repeats	$\{1, 3\}$	2

As region of interest (RoI) for the UCI datasets we set quite large ranges (as we have enough evaluations for these benchmarks). The range of γ is in $[0.0, 1.0]$ and the range of cost C in $[0.0, 10.0]$. For the other applications (AppAcid, DMC2007) we relied on the same settings as in our previous experiments, see [155] for more information.

5.2.3.3 Resampling

First of all, k subsets $D_1, \ldots, D_k \subset D_{train}$ of the complete training data D_{train} lead to models $M_1, \ldots, M_k$ which are presumably not equal when the training data subsets are not

equal, although hyperparameters γ and C are identical (on the same training data SVM is deterministic). Different strategies are possible for sampling the training subsets during tuning:

(A) Choose a subset $D_1 \subseteq D_{train}$ *once* and use it for the whole parameter optimization process. We call this option *parameter tuning without resampling*.

(B) In each call i of the objective function, choose a new subset $D_i \subseteq D_{train}$ for training. If a design point is evaluated repeatedly, e.g., n times, we choose different subsets $D_1, D_2, \ldots, D_n$, train models for them and use an aggregation function (here: the mean) to return an objective function value. We call this option *parameter tuning with resampling*.

5.2.3.4 Results

In Fig. 5.1 and Fig. 5.2 we present boxplots of the SPOT hyperparameter tuning for the Sonar and AppAcid datasets respectively. We distinguish between the error achieved on the validation data when using only a smaller training set fraction X (Error VALI Set) and the independent test set error when a complete re-training with the optimal parameter setting and the complete training data is performed (Error TEST Set). Note that the results on the test set can be better than the error obtained on the validation set (the tuning result), which is caused by using the complete training data when measuring the error on the test set. The results shown in Fig. 5.1 and Fig. 5.2 were optimized using SPOT with sampling strategy (B) and a total number of 3 repeated evaluations for each design point. The validation error in both plots is clearly increasing if the training set size X is reduced from $X = 75\%$ to $X = 5\%$. However, the same does not necessarily hold for the test data. While the mean errors and their variances are large for small training fractions, they are roughly constant and small for all $X \geq 15\%$.

This is a promising result, as it may allow us to use only a sub-sample of the complete data during tuning. As we can see in Tab. 5.4, good tuning results with a very small number of training data (here $X = 10\%$) were observed as well for all other datasets. If we take the best parameters from such a tuning process and re-train *once* with the complete training data, we obtain results on independent test data which are competitive with a much more time-consuming "full" tuning.

We observed, that the prediction accuracies for models trained on different data are very noisy. Thus, the chosen sampling strategy (see Sec. 5.2.3.3) does have an impact on the final results. A comparison of the sampling strategies (A) and (B) showed that we can achieve the most stable results using strategy (B). While it is easier to obtain a good parameter setting for a specific training/validation set with strategy (A), this does not hold for sampling strategy (B). When strategy (A) is used, the optimization always uses some training data which is never changed during the optimization. With deterministic learning algorithms like

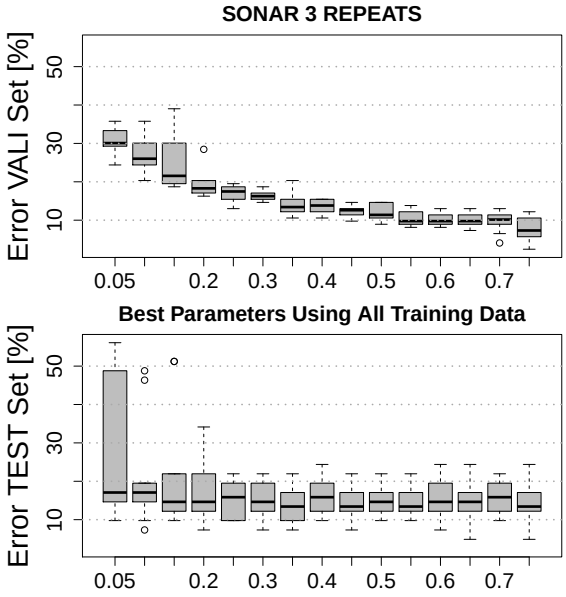


Figure 5.1: Tuning results with SPOT for Sonar dataset using 3 repeats. The x-axis shows the training set fraction X.

SVM, this leads to a deterministic optimization problem, which means to minimize the error always on the same validation set. In strategy (B) instead, the training and validation sets *change in each iteration*. This has an impact on the learned prediction model, and also on the error on the validation set (the training set and validation sets of two consecutive iterations are not comparable). While this strategy seems to be disadvantageous at first, it has to be noted, that we force the learning algorithm indirectly to be good generalizing. With strategy (B) the parameter setting must perform well on different validation sets, otherwise the setting will be discarded soon by the optimization procedure. For this reason, this strategy is not prone to overfitting as is strategy (A), although it might be valuable to get more stable results by testing parameter settings multiple times, avoiding to discard a good setting by mistake. The least is known from design of experiments (DoE) [76], and is achieved by integrating replicates, meaning that each parameter setting is evaluated multiple times. Another option is to introduce noise estimating methods, which are unfortunately only available for some surrogate models, e.g., Kriging and the re-interpolation method by Forrester *et al.* [82].

Due to the high noise levels usually occurring in ML we recommend a number of repeated evaluations greater than 1 (from now on we always set the repeats to 3 in this study). We think that repeated evaluations are an important factor to circumvent wrong decisions of the optimizer operating in a noisy environment.

Regarding the comparison of SPOT and LHD we show in Fig. 5.3 that SPOT usually

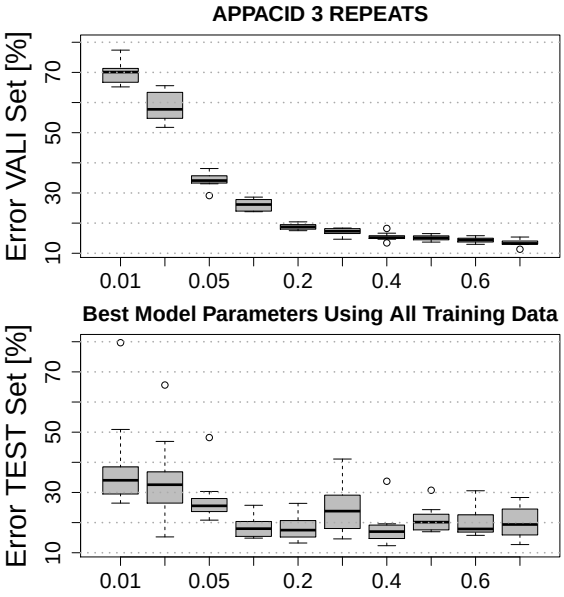


Figure 5.2: Tuning results with SPOT for AppAcid dataset using 3 repeats. The x-axis shows the training set fraction X.

yields better parameter settings when trained with a fraction of the data, especially for $X \leq 10\%$. Results degrade for very low training set sizes (1% or 2%). They are however competitive for all $X \geq 10\%$ (SPOT) and $X \geq 20\%$ (LHS): the tuning results are as good as with a tuning on the "full" training data and the models generalize well on unseen data.

5.2.3.5 Computation times

The characteristics of the computation times for different training set sizes are shown in Fig. 5.4 for the AppAcid dataset. Note that the curve which appears to be linear here can have a different appearance for other datasets. The roughly linear slope has its origin in the model building process for the acid concentration task. This process includes several other operators beneath SVM training, e.g., pre-processing (Principal Component Analysis and feature selection). In other cases the pure SVM computation time might increase quadratically with the number of training samples, leading to even larger computation time savings.

5.3 Optimal computing budget allocation

Chen *et al.* [39] developed a strategy for estimating the optimal budget for repeated design point evaluations. In EGO this strategy was proposed together with the SPOT environment in [9]. The principle in DoE is, that design points are repeatedly evaluated. There are some advantages and disadvantages for performing more than one evaluation of the same design

Table 5.4: Test-set error rates (mean and standard deviation of ten runs) for all datasets investigated. We show results when using small and full training data during tuning, after re-training once with the best parameters found. In case of DMC-2007 we show relative gain instead of error rate.

Dataset	$X = 10\%$ median (std.dev.)	$X = 75\%$ median (std.dev.)
Sonar	17.07 (14.3)	13.41 (5.5)
Glass	28.57 (7.7)	28.57 (7.6)
Liver	36.23 (4.3)	31.88 (6.5)
Ionosphere	5.71 (2.4)	5.71 (2.4)
Pima	23.20 (2.5)	23.20 (3.9)
AppAcid	17.99 (3.3)	19.38 (5.5)
DMC-2007	14.73 (2.0)	15.66 (1.0)

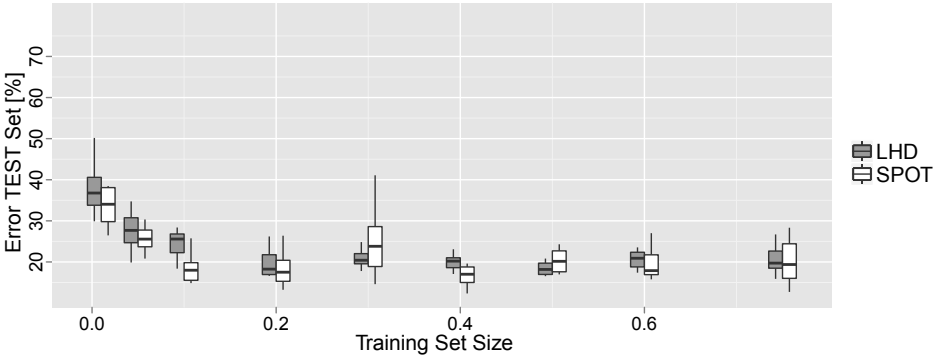


Figure 5.3: Comparison of SPOT and LHD algorithms on AppAcid.

point. Replications are useful, when the underlying objective function is noisy. This is usually the case in parameter tuning for machine learning model selection. Replications can help to get more robust surrogate models, especially after the initial design has been built. As an example in some engineering applications researchers need to perform analyses of the process before further steps are undertaken. However, other researchers think it is not necessary to perform repeated evaluations. Another solution is to evaluate more sequential points instead of spending evaluations in the beginning of the search. This approach rather aims at handling the noise by the surrogate model. This can be done using the re-interpolation technique by Forrester (cf. Sec. 2.5.5), where model uncertainties are considered by deploying an uncertainty band for the evaluated design points. It is difficult to give a clear indication when to use repeats and when not. Chen *et al.* [39] proposed a technique for finding the best number of replications for designs called optimal computing budget allocation (OCBA).

Bartz-Beielstein *et al.* [10] integrated OCBA in the SPO toolbox and analyzed it on five noisy test problems. They showed, that SPO using OCBA could outperform three

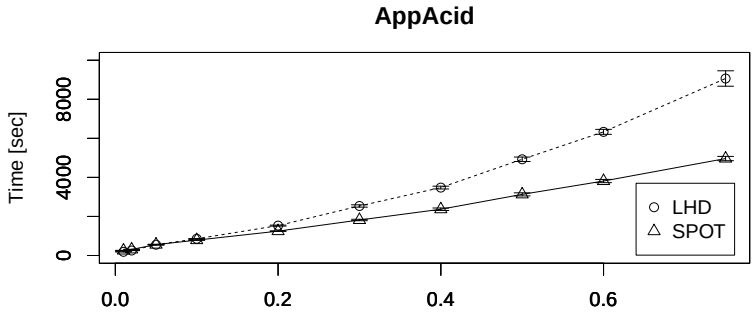


Figure 5.4: Computation time on the acid concentration task as a function of training set size X.

other optimization methods, including CMA-ES [107], Simplex search [185] and Simulated Annealing [143]. Also the robustness of SPO was increased by incorporating OCBA, leading to less outliers in the tuning results.

5.4 Conclusions

We showed that tuning with a low training set size very often leads to good results: an astonishing small fraction of the training set (10–15%) was sufficient to find with high probability a good model when performing one "full" training with the best design point parameters. The resampling strategy (B) (see Sec. 5.2.3.3) might be a crucial ingredient for this success with small training samples, but further research is needed to justify this claim.[1]

In the same way Koch *et al.* [149] investigated seven different datasets with sizes from 208 to 50,000 records. For bigger datasets like the acid concentration tasks large speedups (factor six to ten) are possible as Fig. 5.4 shows. Summarizing the results, we can conclude that both research questions **Q1** and **Q2** can be affirmed for the datasets used in this study. In the future we plan to search strategies for selecting the right sample sizes adaptively as a dynamic parameter during tuning.

[1] We note that Daelemans *et al.* [53] got negative results with a small training sample, but only on a specific word disambiguation task and without resampling.

Chapter 6

Landscape analysis

In Evolutionary Computation (EC) the response or output of an objective function is usually written as *fitness*. We can view the number of responses as a landscape of fitness values and their corresponding parameters. Sometimes it is beneficial to plot the fitness landscape, to get new insights into advantages or disadvantages of the modeling. Unfortunately, plotting is restricted to small parameter spaces, because for higher dimensions no visualizations can be given without making use of dimension reduction strategies. In this Chapter we analyze the fitness landscapes of Kriging surrogate models and detect interesting behaviour even for small parameter dimensions.

6.1 Related work

Model-assisted optimization can provide a promising alternative to classical optimization methods, especially when the objective function is expensive. Surrogate modeling techniques like Kriging allow to reduce the number of real objective function calls by learning a surrogate model of the real target function. For tuning ML models, it was shown that model-based optimization techniques give the most stable results, even when the model is only trained on a subset of the training data (Sec. 5.2). However, up until now no deeper analysis of the surrogate-model quality was undertaken. We try to get a better understanding of the optimization process: how accurate are surrogate-based optimization techniques? Are the approximated landscapes realistic or do they show a strange behaviour in certain regions? And furthermore: does the focussing on good solutions lead to inferior accuracy in regions which were initially disregarded because the quality was worse compared with other regions?

Earlier work on fitness landscape analysis is mainly available in the field of algorithm selection [205]. In algorithm selection, one is interested in determining the optimization algorithm which solves best the instances of a certain problem class (or cluster of problems). In [134] Jones and Forrest proposed a fitness distance correlation (FDC) measure for predicting the performance of Genetic Algorithms on both deceptive and non-deceptive problems. Recent work often uses certain *features* of the instances. This can be a feature of

the problem itself, or more complex features obtained in hill-climbing runs on the problem instances. E.g., Merz and Freisleben [179] classified problem instances according to their difficulty and name this *fitness landscape analysis*. For further reading we refer the reader to Mersmann *et al.* [177, 178], Bischl *et al.* [19], and Abell *et al.* [1].

6.2 Research questions

The sequential parameter optimization (SPO) with Kriging has shown to be good-working for ML parameter tuning [147, 150, 155]. Although Kriging performed well, it remained unclear, if the surrogate models are precise or if improvements of the surrogate model fits can even lead to better solutions. Recently, Koch and Konen [149] showed that the fitted fitness landscapes of certain Kriging methods sometimes are highly decorrelated from the real landscapes. Imprecise regions of the search space could be observed throughout all of the optimization runs. In this chapter give a deeper insight in the modeling process. The points of research are:

Q1 Does noise in the target function pose problems for Kriging-based landscapes, if not the right Kriging method is selected and parameters are not chosen carefully?

Q2 Can we propose both methodical and heuristic solutions for detecting and improving the goodness of the landscape fit?

Systematic optimization of ML parameters includes methods for global optimization. Glasmachers and Igel [94] presented an approach where CMA-ES [108] can handle model uncertainties and noise which is frequently present in ML. Today, surrogate-based optimization techniques (see Keane [137] for a comprehensive overview) and especially Kriging have become more and more important. Many comparative studies of surrogate-modeling techniques have been proposed, e.g., Kim *et al.* [141] report a superior performance of Kriging, while Jin and Chen [126] indicate earlier that the design size and distribution of the samples is crucial to receive an accurate surrogate-model. Noise can be a problematic factor for Kriging surrogate-models, since standard Kriging is only an interpolating technique. Due to the data sampling, optimization problems in ML usually have noisy objective function values, thus the surrogate-models should also be able to handle the uncertainty and noise. Therefore, Jin *et al.* [127] and Zhao and Xue [261] analyzed the quality of surrogate-modeling approaches for optimization under uncertainty. Especially Kriging with an additional nugget estimation [261] should be studied for such tasks. In our study we used SPO by Bartz-Beielstein *et al.* [11] which makes it possible to switch between various state-of-the-art surrogate-modeling techniques easily.

6.3 Experimental analysis

We performed an optimization of the ML hyperparameters γ and C of Support Vector Machines (SVMs) [246]. For visualization we have restricted ourselves to these two parameters. Nevertheless our findings are assumed to be also valid in higher dimensional spaces. In our experiments we used the Tuned Data Mining in R (TDMR)[1] framework [154] as a tuning framework. As surrogate-model for SPO we used Kriging based on Gaussian Processes (*Gausspr* from the Kernlab package[2] in R [199]) and the *Maximum Likelihood Estimates of Gaussian Processes* (MLEGP).[3] The surrogate model MLEGP was parametrized for usage with additional nugget estimation to handle noise, while *Gausspr* was used as standard interpolating Kriging operator.

Fig. 6.1 shows a comparison of the surrogate surfaces for a small ($X = 10\%$) and large training set size on the Sonar dataset [84]. We used a surrogate model based on Gaussian processes [136] for tuning and plotting. When only few training data were used (top left and bottom left plot in Fig. 6.1), the LHS and SPOT landscapes are both relatively flat, with shallow minima near $\gamma \approx 0$. These minima are however a good indicator for the minima obtained when we perform the tuning with the complete training set size (top right and bottom right). These plots both exhibit a clear and deep minimum near $\gamma \approx 0$, relatively independent of the cost term. The landscape for $\gamma > 0$ is however very different. With SPOT, we obtain very spiky surrogate models (Fig. 6.1, bottom right). This especially occurs in regions where only few design points are sampled (these are the regions presumably not containing the optimum). We think that when there is a region with a high density of points but with much noise in the objective function, Gaussian processes assume a very small correlation length leading to spikes in the low-density regions. Overall this leads to a less robust surrogate model.

LHS with its equal-density sampling in design space does not have this problem: The landscape (Fig. 6.2, upper right) exhibits a low curvature and is stable in all experiments. Nevertheless the main issue of LHD sampling, which is the bad scalability for higher dimensions, remains, and this is the reason why this sampling method is less preferable for higher-dimensional search spaces.

In Fig. 6.2 we show the contour plots of a parameter tuning run on the Sonar dataset[4] for interpolating Kriging and Kriging with added nugget estimation. It can be seen from the plot that a Latin hypercube design together with a simple interpolating Kriging surrogate-model (left: *Gausspr*) nicely models the whole parameter space. However, the same does not hold when SPOT is used for the design point generation. Here, the surrogate-model without interpolation (middle) is completely deteriorated. A thing to note is that the design points

[1] `http://cran.r-project.org/web/packages/TDMR/index.html`
[2] `http://cran.r-project.org/web/packages/kernlab/index.html`
[3] `http://cran.r-project.org/web/packages/mlegp/index.html`
[4] `http://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks)`

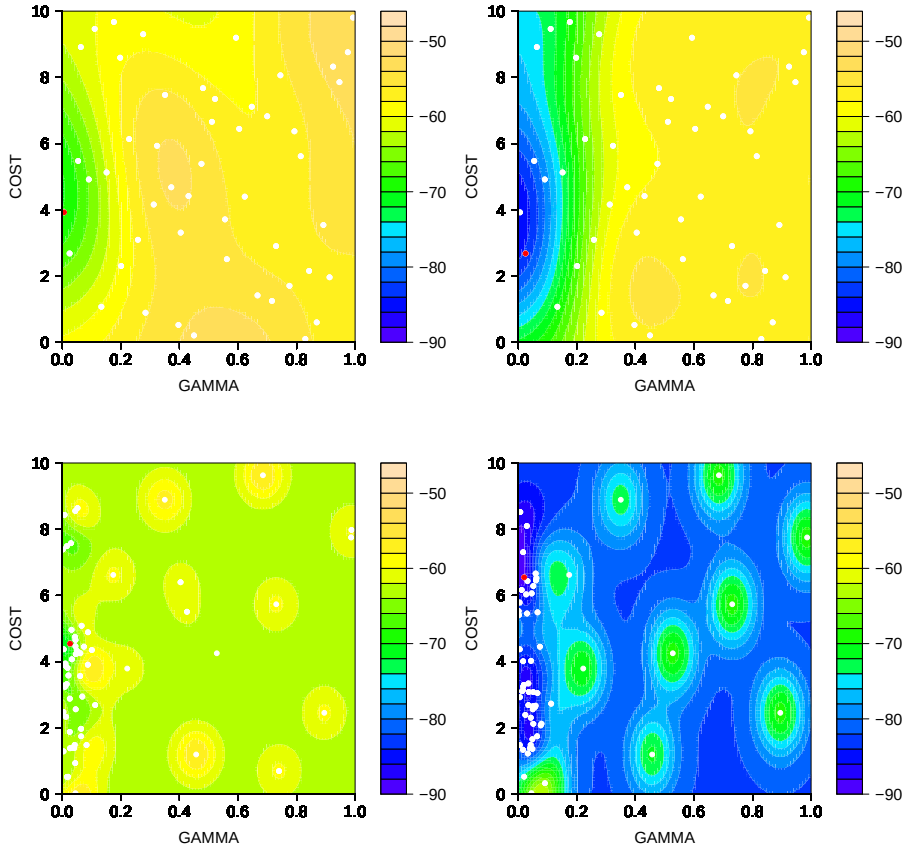


Figure 6.1: Optimization of the SVM parameters γ and C on the Sonar dataset using LHS (top) and SPOT (bottom). The left plots show a tuning with few training data, whereas the right plots show a tuning with the complete training data. For SPOT a Kriging surrogate model without nugget estimation was used. The red points depict optimal solutions found by the tuning algorithms.

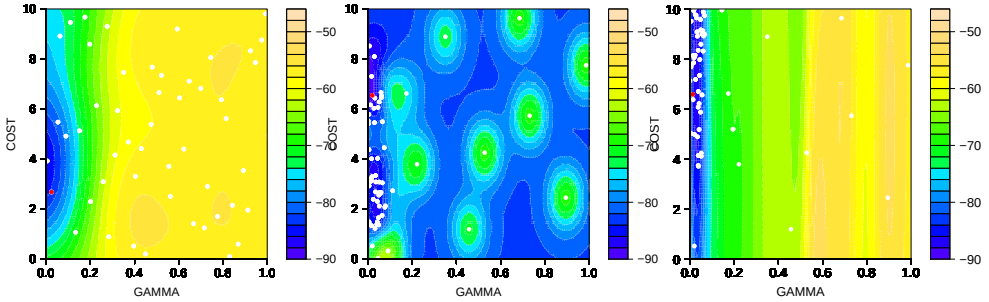


Figure 6.2: Contour plots for SVM parameter optimization on the Sonar dataset. Left: LHS and simple Kriging without nugget estimation. Middle: SPOT design and simple Kriging without nugget estimation. Right: SPOT design and Kriging with nugget estimation. We evaluated a total number of 50 design points (white dots), each point repeatedly evaluated 3 times, taking the mean as aggregation function. The red points are optimal.

are mainly aggregated in the optimal region (low γ values) as can be seen by the white points (the design points). This is caused by the selective pressure of SPOT, where good solutions are selected with higher probability. As a consequence, the model fit with the simple interpolating Kriging exhibits a too small correlation length. In regions of the search space without sampled points the surrogate model quickly approximates the mean value of all available solutions, an often biased value. A solution to this problem can be to use a a non-interpolating Kriging model. As soon as a nugget term is added, the landscape appears to be more convenient (right plot in Fig. 6.2).

Non-uniform distributions of design points

The non-uniform distribution of the SPOT sampled design can lead to an inexact surrogate model as described earlier. The easiest way to handle this imbalanced design space is to use a Kriging method with nugget term. Other solutions also aim at avoiding a too early concentration of sequential design points in the expected optimal basin. E.g., for interpolating Kriging models, the expected improvement (EI) [133] criterion can be used as infill criterion, probably leading to more exploratory steps. For non-interpolating Kriging models, also expected improvement can be maximized by using the re-interpolation technique as proposed by Forrester *et al.* [82]. A more detailed study on infill criteria for noisy Kriging methods has been given by Picheny *et al.* [192]. Other possibilities are to cluster solutions in the expected optimal region, and to select a subset of these clusters, or to avoid placing too dense design points in the initial design of experiments (DoE). Unfortunately these proposals also reduce the accuracy in the optimal basin and are difficult to parametrize. Therefore, using the nugget estimation is the most preferable way to generate an accurate model.

6.4 Conclusions

We analyzed the fitness landscapes of Kriging surrogate models. All landscape plots presented in this chapter were obtained from a parameter optimization of SVM learning parameters. For tuning we used EGO performed by SPOT using Kriging surrogate models. Two different variants of the Kriging models were incorporated: a) interpolating Kriging, and b) Kriging using a smoothing procedure by adding a regularization constant (here denoted as *non-interpolating Kriging*). In both cases a greedy infill strategy was chosen, that is a sequential design is generated in the neighbourhood of the best design point. It could be seen that the accuracy of interpolating Kriging variants seems to be biased to regions of the search space which are assumed to be promising in the initial steps. The initial design size was kept constant, but not too many evaluations were spent for the initial LHS. Here we showed that SPOT sampled the majority of the sequential points in the region, which was considered to be the best in the initial design. This led to a clear bias towards the optimal design point obtained from the LHS. As a result, deteriorating effects can occur, because the distribution of the design points is biased leading to a strange looking landscape. The Kriging interpolation could be identified as reason for this behaviour and the misleading accuracies of the underlying fitness landscapes. Thus we can affirmate research question **Q1**, stating that noise can pose problems when not the right Kriging variant is incorporated. As first solution to this problem we showed how non-interpolating Kriging methods could lead to better estimations of the fitness landscape in these regions finally giving a positive answer for research question **Q2**.

Chapter 7

Efficient multi-criteria optimization in machine learning

In the previous chapters we showed that finding good parameters is essential for training a precise classifier. Up to now, this problem has been defined as a single-criteria optimization (SCO) problem, where the classification error or any other quality indicator was minimized. Systematic global optimization was used to optimize ML processes in [148, 149, 155]. Here, SPO [8] with Kriging surrogate models performed best for problems with very restrictive budgets.

However, in most cases the user is not only interested in high-quality prediction models, but also wants that model training and parameter tuning is performed in a reasonable amount of time. These objectives are usually conflicting, leading to the concept of multi-criteria optimization (MCO). Nowadays, model-assisted optimization algorithms are also available for multi-criteria optimization problems [69, 144, 194, 259]. Up to now this paradigm has received little attention in the ML community, and only few work has been spent to optimize the mentioned objectives at the same time.

One reason for this is in the nature of ML parameter tuning, where the evaluated design points are usually noisy. What is the source of noise in ML tasks? Besides the normal noise of the ML data, it is mainly related to the randomness in the selection of training, validation and test data which lead to different results, even for deterministic ML models. A surrogate model has to cope with such noisy responses. The noise in the quality response will be increased if we seek for solutions with low runtime, which is usually connected with smaller training set samples.

Summarizing, the typical challenges of noisy MCO can be formulated as follows: (a) finding a good approximation of the Pareto front, (b) coping with the noise and (c) finding a good approximation with very restricted budgets of function evaluations.

We aim at applying multi-criteria model-assisted optimization variants to solve multi-criteria ML tasks.

For this reason we define research questions in this chapter as follows:

Q1 Is multi-criteria optimization with surrogate modeling (Kriging) possible in the presence of strong noise?

Q2 Is it also possible when the budget for function evaluations (i.e., ML training runs) is very restricted?

Q3 Is it necessary to dampen the noise by averaging over repeated function evaluations, at the price of fewer allowed infills under the given budget?

Q4 Are the multi-criteria model-assisted optimization approaches better in finding good approximation sets than traditional design of experiments (DoE) techniques, e.g., LHS? Are there significant differences between the different model-assisted optimization approaches?

The chapter is structured as follows: in Sec. 7.1 we highlight related approaches. A general introduction of MCO is given in Sec. 7.2.1. In Sec. 7.3 we describe the setup of the study for efficient multi-criteria ML experiments. The experimental results are discussed in Sec. 7.4 and we give a conclusion in Sec. 7.5.

7.1 Related work

Because most supervised ML models like Support Vector Machines (SVMs) [246, 52, 215] are sensitive to their hyperparameter settings, an optimization is required until an optimal behaviour of the models can be guaranteed. For single-criteria ML problems we already discussed related work in Sec. 4.1 and showed that a tuning using SPO performs better on a set of benchmark problems than other state-of-the-art optimization heuristics (see Sec. 4.3). On the basis of these results Koch and Konen [149] discovered that tuning with small fractions of the available training data can lead to good parameter settings (see Sec. 5.1). But any a priori setting of the training set size without special problem knowledge remained virtually impossible. A solution to this issue is to explore a set of solutions, representing alternatives between small and large training set sizes, so that a suitable size is finally delivered to the user.

In ML research, MCO was firstly proposed by Liu and Kadirkamanathan [167]. They optimized a radial basis function network, where two objectives functions were considered to optimize the differences between the real non-linear system and the non-linear model, and another function to emphasize simpler models. Freitas [85] and Jin and Sendhoff [130] give comprehensive reviews about the employment of multi-criteria algorithms in ML. Jin [128] advocates to use Pareto-based approaches, covering both supervised and unsupervised learning.

In these approaches the authors used evolutionary multi-objective algorithms (EMOA). The required number of real function evaluations is considerably higher for these algorithms which can be problematic, because the computation time is usually very limited. Instead Knowles and Nakayama [146] discuss the use of surrogate-modeling techniques for multi-criteria optimization problems. The first EMOA using surrogate models was proposed by Giannakoglou *et al.* [91]. A popular variant of EGO for multi-criteria optimization was given by Knowles [144], the Pareto-EGO (Par-EGO). Later, EGO approaches using the hypervolume as infill criterion were introduced, e.g., the SMS-EGO by Ponweiser *et al.* [194], or the hypervolume-based EI criterion by Emmerich *et al.* [69], also referred to as $\mathcal{S}$-metric based Expected Improvement (SExI) [249]. Another approach based on decomposition is the MOEA/D by Zhang *et al.* [260]. Recently, Zaefferer *et al.* [258] compared SMS-EMOA, a well-known solver without surrogate modeling, and four cutting-edge multi-criteria solvers with surrogate modeling (SMS-EGO, SExI-EGO, MSPOT and MEI-SPOT) on an optimization task without noise. An overview about the properties of multi-criteria EGO variants was given by Wagner *et al.* [249].

There is a large body of work on noisy Kriging-based optimization (NKO) for single-criteria optimization tasks [82, 119, 191, 247]. Picheny *et al.* [192] give a comprehensive overview and perform a benchmark of several NKO-approaches on a variety of well-known hard optimization problems with a steerable amount of additive noise. Compared to this, only few work has been done in the area of NKO for multi-criteria optimization tasks. An exception can be found in Knowles *et al.* [145], but to our knowledge there exists no related work of multi-criteria NKO in ML.

7.2 Methods

In this section we describe the general multi-criteria optimization task and give a short overview about existing algorithms solving such problems.

7.2.1 Multi-criteria optimization

In multi-criteria optimization, multiple objective functions are optimized in parallel:

$$\min_{\vec{x} \in S} \left(f_1(\vec{x}), f_2(\vec{x}), \ldots, f_m(\vec{x}) \right) \tag{7.1}$$

Note that the individual objective functions $f_1, \ldots, f_m$ can be conflicting, so that the sets of the optimal solutions for the functions cannot be unique. We say a point $\vec{x}_1$ *dominates* a point $\vec{x}_2$, if all functions values of $\vec{x}_1$ are smaller than or equal to those of the corresponding objective function values of $\vec{x}_2$, and at least one objective function value of $\vec{x}_1$ is strictly better:

$$\vec{x}_1 \prec \vec{x}_2 \tag{7.2}$$

A point x^* is considered to be Pareto-optimal, if there does not exist any feasible point $\vec{x}^{(d)} \in S$ which would decrease one objective function value without a simultaneous increase in any other objective function value.

In the last years set-based approaches have been established to approximate the Pareto-optimal front [262], and here especially evolutionary multi-objective algorithms (EMOA) have become popular methods in practice. The advantage of set-based approaches is that a group of points distributed over the Pareto front is approximated within a single run of the algorithm, while in single-solution approaches only a single trade-off between the objectives is returned in the end.

7.2.2 Model-assisted multi-criteria optimization

The most important weakness of the usage of evolutionary algorithms is probably that they suffer from a large number of function calls required for converging to good regions of the search space. Especially in multi-criteria optimization this is a clear disadvantage and can result in prohibitive computation times. A frequently shared paradigm in this case is to use surrogate-functions which model the real fitness landscape and can help in performing optimization runs with much fewer real function evaluations [146].

In MCO, surrogate models must be learned for each objective. This set of surrogate models is then used for the prediction of promising new design points using EI definitions for multiple criteria [249].

The following EGO methods are popular in MCO:

ParEGO (Pareto optimization EGO by Knowles [144]): This was the first approach to use model-assisted EGO techniques in MCO. In each iteration a different random weight vector is used to combine the normalized objectives to a scalar function (scalarization). For each previously visited point this scalar function is calculated and a surrogate model is fitted. On this surrogate model the expected improvement (EI) criterion defined in Sec. 2.5.4.2 is used to determine the next infill point.

SMS-EGO ($\mathcal{S}$-Metric Selection EGO by Ponweiser *et al.* [194]): A hypervolume-based infill criterion where the important part of the improvement function is based on the hypervolume increase due to a potential solution.

SExI-EGO ($\mathcal{S}$-Metric Expected Improvement EGO by Emmerich *et al.* [69]): This is another hypervolume-based infill criterion where for each point with objective values and predicted variances given by the surrogate models the exact computation of the expected improvement in the hypervolume is done.

The hypervolume indicator or the $\mathcal{S}$-metric is defined as the Lebesgue measure of the subspace that is dominated by the approximation set. To make this measure finite, the

subspace is cut from above by a reference point r. The reference point is chosen in such a way that all points in the approximation set dominate it. The $\mathcal{S}$-metric is also called a scalar indicator for measuring improvements in an approximation set to the Pareto front.

Since the $\mathcal{S}$-metric indicates good theoretical properties in [249], we consider both the $\mathcal{S}$-metric Selection-EGO (SMS-EGO) and the $\mathcal{S}$-Metric Expected Improvement-EGO (SExI-EGO) as model-assisted algorithms for our experiments on multi-criteria optimization of ML models.

We compare these two EGO-approaches with Latin hypercube sampling (LHS) as a baseline. For LHS the full budget of real function evaluations are spent to build competing optimal LHS in the region of interest. The set of non-dominated solutions among the LHS points is determined and its hypervolume calculated.

7.2.3 Noise handling

Model evaluations in ML are usually subject to noise. Although learning algorithms like SVMs are deterministic, their training depends on the (random) order of feeding examples which can lead to different model accuracies. This can cause wrong decisions during tuning, because the returned objective function values are unstable and have variances significantly larger than zero.

Replications have been introduced in Sec. 2.5.5.1. We investigate below whether replicates have a beneficial effect on the overall quality. As another alternative the re-interpolation procedure by Forrester *et al.* [82] can be used. We compare replications and re-interpolation for multi-criteria tuning in ML.

7.3 Experimental analysis

We performed an experimental study using two objectives f_1 and f_2 for parameter tuning in ML. For the first objective we used the *classification gain* and for the second objective we measured the *training time* of the learning algorithm. It is very likely that these objectives are conflicting: the runtime increases, when more data is used for the learning. However, the classification result should improve in this case.

Different variants of model-assisted multi-criteria algorithms are analyzed to measure the performance of the EGO approach. All algorithms are implemented in Matlab version 7.12.0.635 (R2011a) and always Kriging is used together with different EI criteria based on the hypervolume. The learning itself was implemented in R using the TDMR framework [154].

7.3.1 Experimental setup

As initial design we always used a LHS consisting of n_{init} parameter design points, which were evaluated on the real objective function. We spent $n_{init} = 20$ evaluations for the Sonar dataset from the UCI library [84] and $n_{init} = 50$ evaluations on the acid concentration problem (AppAcid) introduced in Sec. 4.3.2. For Sonar 130 sequential steps were performed,

while we spent 150 evaluations for the AppAcid problem. The maximum number of surrogate model evaluations was restricted to $50,000$ evaluations. As an additional stopping criterion we forced the algorithms to stop when the objective function value doesn't give more than $\varepsilon = 10^{-7}$ improvement. This restriction was necessary, because although evaluations on the surrogate model are usually cheap, some EGO implementations tend to calculate the dominated hypervolume very often, which can be time-consuming also for small objective dimensions m.

For the ML, we used SVM as learning algorithm and the available data were split prior to tuning in 20% test data and 80% for training and validation. The SVM implementation was taken from the *e1071* R package. As kernel for the SVM we chose the radial basis kernel. As shown earlier this learning algorithm is sensitive to the parameter settings of the kernel parameter γ. For the Sonar benchmark we tuned γ together with the parameter $trnFrac$ as another tuning parameter, having a somehow comparable setup as in [149] (see Tab. 7.1). The tuning parameter $trnFrac$ defines the fraction of the training patterns related to the size of the training and validation set. The patterns not considered for training were used for validation purposes. We did not tune the SVM regularization parameter C because in earlier experiments the model was rather insensitive to its settings.

Table 7.1: Region of interest (RoI) for Sonar tuning experiments. See main text in Sec. 7.3.1 for explanation of parameters.

Parameter	Lower	Upper	Type
γ	0.01	0.7	float
$trnFrac$	0.05	0.7	float

For the more difficult AppAcid problem, we tuned seven parameters. Detailed information about the region of interest and the parameters for AppAcid can be found in Tab. 7.2. We employed two objectives for the problems consisting of $f_1 =$ *gain on test set* and $f_2 =$ *training time*. The parameter $trnFrac$ should directly affect objective f_2, while the other parameters should mainly affect function f_1. The region of interest (RoI) was chosen according to settings from preliminary runs and the settings used in [155]. Since the parameters partly affect functions f_1 and f_2, it is of interest, whether the optimization methods are capable to learn the parameter relevances.

All experiments include the run of a multi-criteria optimizer (SMS-EGO, SExI-EGO or LHS), where in each iteration the learning algorithm is called. In each call the learning algorithm takes as input a subset of the data for building a prediction model (both for Sonar and AppAcid) and the hyperparameters set by the optimizer. For each experiment, ten runs were performed with different randomly drawn test and training-validation data samples to get statistically sound results.

Table 7.2: RoI for AppAcid tuning experiments. Parameters γ and $trnFrac$ have the same meaning as in the Sonar task, see Sec. 7.3.1. Parameter $xperc$ controls the feature selection, the algorithm selects a number of the most important features to capture a fraction $xperc$ of the overall importance. The parameters $CUTOFFi, i = 1, 2, 3, 4$ control the weight for each class.

Parameter	Lower	Upper	Type
γ	0.001	0.8	float
$trnFrac$	0.2	0.7	float
$xperc$	0.050	1.0	float
$CUTOFF1$	0.010	0.4	float
$CUTOFF2$	0.010	0.4	float
$CUTOFF3$	0.010	0.4	float
$CUTOFF4$	0.010	0.4	float

7.3.2 Benchmarking

Benchmarking multi-criteria optimization algorithms usually includes the distance of the approximation set to the real Pareto front and the spread of the points over the front. Since the real Pareto front is unknown in our case, we try to introduce a fair benchmarking which nevertheless respects both performance measures. At first we summarize all results by taking the union of the approximation sets from all 10 runs, and selecting its non-dominated subset. We call this resulting set the *reference set*. The distance of the non-dominated sets of each run and this reference set then describes the approximation quality. In the best case an algorithm would produce exactly this reference set, so that the total distance would become zero. Now we only have to define how the distance between the single algorithm runs and the reference set are measured. Since the hypervolume and other quality indicators are controversely discussed in MCO, we additionally incorporate the $R2$ indicator by Hansen and Jaszkiewicz [103].

For a given approximation set A and a set of arbitrary utility functions U, the $R2$ indicator is defined as follows

$$R2(A, U) := -\frac{1}{|U|} \sum_{u \in U} \max_{a \in A} u(a) \tag{7.3}$$

As utility function we used the weighted Tchebycheff distance, but in fact also other metrics (e.g., the Minkowski distance) could be used here:

$$u(z) = u_\lambda(\vec{z}) = - \max_{j \in \{1,..,m\}} \lambda_j |z_j^* - z_j| \tag{7.4}$$

with z^* the utopian point (a point that can never be approximated in the objective space) and $\vec{\lambda} = (\lambda_1, ..., \lambda_m)^T$ a weight vector for the objective functions. Different, randomly chosen $\vec{\lambda}$ constitute the utility function set U. In our experiments we sampled U uniformly at random from the interval $[0, 1]$ as proposed by Hansen and Jaszkiewicz [103]. The size of U is heuristically defined as follows:

$$|U| = \binom{s+m-1}{m-1} \tag{7.5}$$

with

$$s = \begin{cases} 500 & \text{if } m = 2 \\ 30 & m = 3 \\ 12 & m = 4 \\ 8 & m = 5 \\ 3 & otherwise \end{cases} \tag{7.6}$$

Other performance indicators are not considered here, but with similar advantageous properties the averaged Haussdorff distance [216] to the reference attainment surface could be taken.

7.3.3 Sonar

Because Sonar is a small dataset with only 208 records [84], a parameter tuning can be performed quickly to test the performance of tuning algorithms. For simplicity we performed a tuning with only two parameters (SVM kernel parameter γ and parameter $trnFrac$ for the size of the training set within the tuning).

In Fig. 7.1 we show the solution space of ten independent runs of SMS-EGO on the Sonar dataset. We spent 150 evaluations for the tuning. Input parameters were γ and $trnFrac$ with the RoI settings as described in Tab. 7.1. In the upper plot ($r = 1$: one evaluation for each sequential design point) we used the re-interpolation procedure by Forrester *et al.* [82] with the SMS infill critierion. It is possible, that the solutions with a gain value better than -90% are biased due to the random resampling. Nevertheless for SMS-EGO with $r = 1$ the approximation quality appears to be better than in the lower plot ($r = 3$: three evaluations for each sequential design point). Using replicates with a reduced total number of iterations at first results in worse approximation, compared with having more exploration through spending only a single evaluation for each design point.

To highlight the amount of noise of the ML problem (we term the observed noise of the problem *empirical variance*), we analyzed the variances of the obtained parameter settings. Therefore we re-evaluated each solution obtained from the SMS-EGO runs ten times, each time using a new random sample for the training set. We show the results of the *empirical variance* for the two objectives in Fig. 7.2, where the plot of Fig. 7.1 is extended with the variances obtained in the re-evaluation procedure.

We also investigated the quality of the surrogate model predictions. In Fig. 7.3 we compare the obtained solutions during tuning with ten repeated evaluations on the real objective function. In the optimal case all points would lie on the diagonal of the plot.

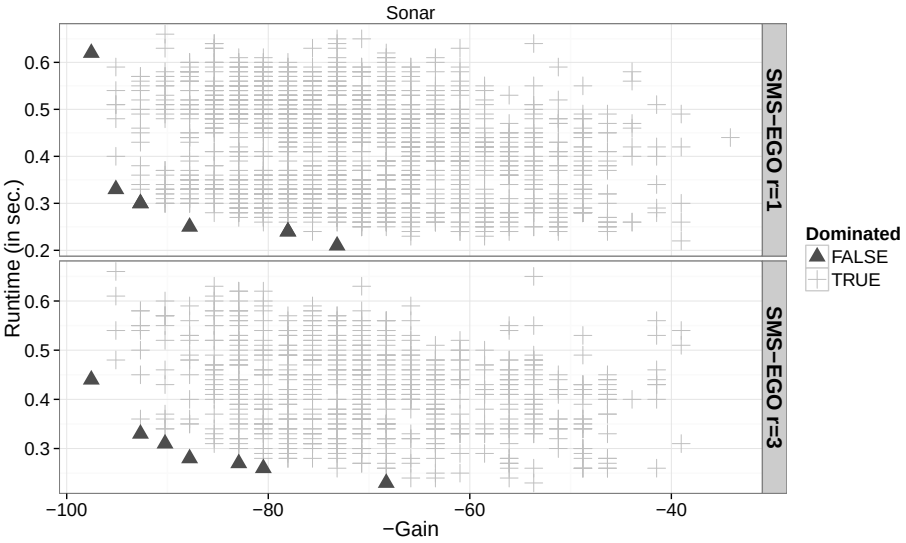


Figure 7.1: Tuning results with 150 real function evaluations on the Sonar dataset. In the plot the aggregated solutions of ten runs in the objective space are displayed. On the x-axis the negative gain values (first objective) are depicted, while the y-axis shows the runtime (the second objective).The initial design size (LHS) was 20. Only a single evaluation was performed in the first experiment (top), while we spent three repeated evaluations for the second experiment (bottom). SMS-EGO was run using a RBF kernel together with re-interpolation. All solutions were drawn together and a non-dominated filtering was applied to these points. The triangles represent the non-dominated solutions of this combined filtering, while the crosses represent the dominated points of the combined filtering.

However, a perfect fit is unrealistic, because a certain amount of noise is always present in the data. But as can be seen from the plot, most of the solutions tend to be close to the diagonal, with some outliers.

In a final comparison all non-dominated solutions of all variants were re-evaluated ten times after the budget was fully spent (here: 150 trainings and evaluations of the Sonar dataset). This was done to remove biases in the non-dominated sets. Then a reference set for all algorithms was determined, consisting of the non-dominated solutions of all algorithms and runs of the re-evaluation. The differences of the obtained quality indicator values for $R2$ and dominated hypervolume and this reference set are shown in Fig. 7.4. Small values indicate good performance of the algorithm, while large differences to the reference set indicate poor performances.

It can be seen from the boxplots that all EGO variants have smaller variances than the LHS solutions. In a t-test we obtained that SExI-EGO with three evaluations (r=3) is statistically significant better than LHS. All other variants did not show significant results, but as the boxplots reveal, they are more stable than the simple LHS baseline approach. Another thing to note is, that although sometimes LHS performs good, this will be in

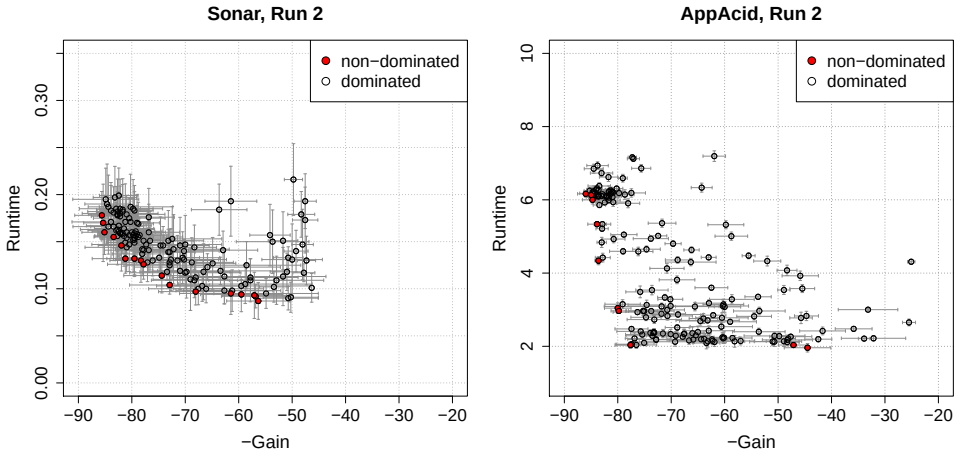


Figure 7.2: Objective space plots showing the empirical variances for the two objectives *–Gain* and *Runtime*. The error bars represent the standard deviations from 10 repeated evaluations (re-training on a new training sample) of a certain parameter point. The gain is obtained on an independent test set. Shown are the variances for Sonar and AppAcid. The variances are bigger for Sonar since the number of records in a sample is smaller.

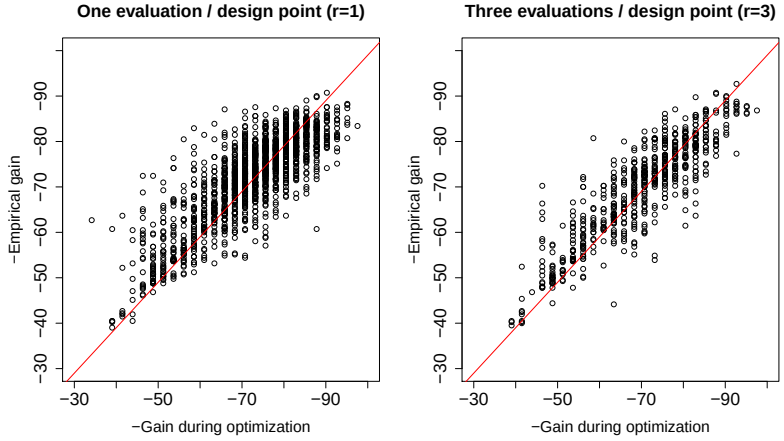


Figure 7.3: Tuning result of SMS-EGO on the Sonar dataset for the objective *gain*. The plot shows the results seen by the tuning algorithm on the x-axis and of a new evaluation, where each design point was evaluated again ten times (yielding the empirical gain). Left: ten runs of SMS-EGO with a single evaluation for each design point ($R2 = 0.67$). Right: ten runs of SMS-EGO, but with three evaluations for each design point in the sequential part of the algorithm ($R2 = 0.77$). The red diagonal line denotes possible situations where the empirical gain would be equal to the gain during tuning.

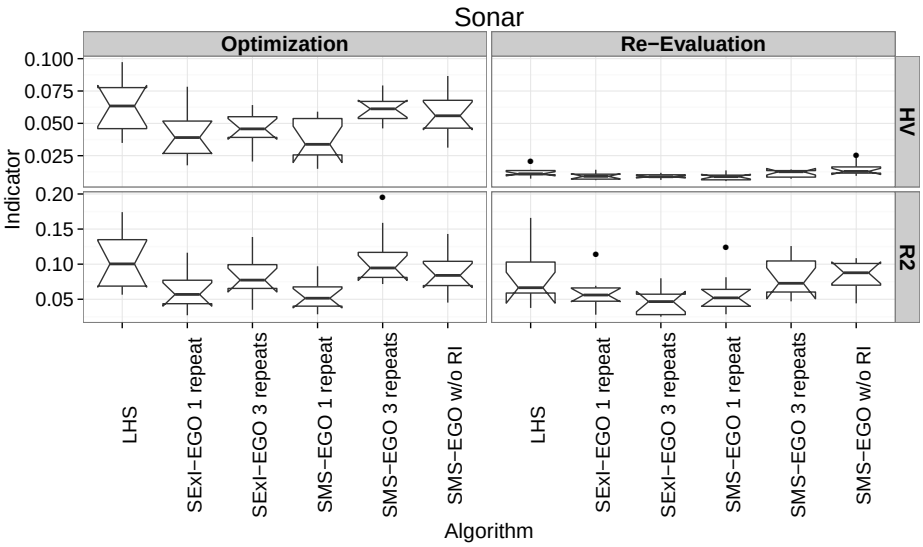


Figure 7.4: $R2$ and HV quality indicators for ten independent runs on task Sonar. The non-dominated solutions obtained after 150 function evaluations were evaluated again ten times, each time using different training/validation splittings to remove biased evaluations. For each quality indicator the mean value of these re-evaluations was calculated. Then we determined the differences between the reference set (set of non-dominated solutions of all runs) and the mean quality indicator values. Thus values of zero would be optimal. We compare the EGO variants both with one function evaluation ($r = 1$) and three function evaluations ($r = 3$). Additionally, SMS-EGO was run without re-interpolation and a single evaluation for each design point (SMS-EGO without RI).

general only true for small parameter spaces. In the Sonar experiment, where we have only two tunable parameters, the performance of the LHS is not so bad when enough evaluations are available. But if there are only a few real evaluations, LHS should be worse than the EGO algorithms. Besides that, it is obvious that as soon as the dimension of the input space increases, LHS will probably fail, or needs a tremendous number of real evaluations. Results for larger parameter spaces are shown later in this chapter when we tune the parameters for the AppAcid dataset.

The behaviour of SMS-EGO, SExI-EGO and LHS during the optimization process is shown as a line plot in Fig. 7.5. It can be seen from the plot that the EGO variants achieve a very fast decrease of the difference to the attainment surface, especially in the first 40 iterations. Although LHS does not include a sequential process, we plot it as a line here. The line for LHS is obtained by evaluating the quality indicators, when $51, ..., 150$ LHS points are considered for the approximation set. It can be concluded that LHS converges more slowly to the reference set compared with SMS-EGO and SExI-EGO.

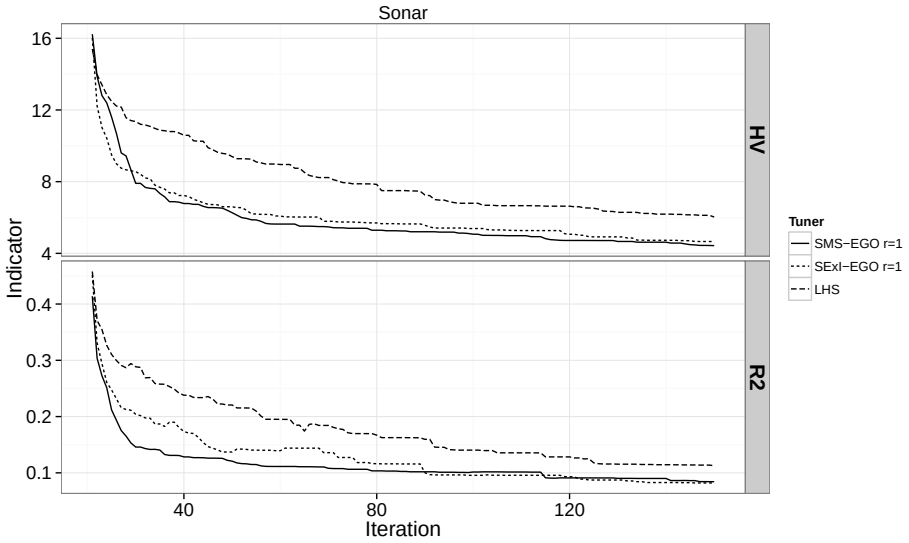


Figure 7.5: Sonar: Mean HV and $R2$ development of ten runs of SExI-EGO, SMS-EGO and LHS. The plot shows that the EGO variants approximate quicker to good HV and R2 approximations compared with LHS. The initial design of size 20 is not plotted for the variants, but we show the situation directly after the initial evaluations.

7.3.4 AppAcid

AppAcid is a dataset for classifying spectrography measurements of a biogas plant (cf. Sec 4.3.2). In our earlier SCO tuning with model-assisted optimization techniques (e.g., Sec. 4.3 or [155]) we obtained a performance in mean classification accuracy of up to 88%.

It was our goal to show the effect of balancing runtime and prediction accuracy. Our approach shows an interesting behaviour of the runtime, because SCO in ML is known to be very time-consuming when state-of-the-art models are considered. It can be seen from Fig. 7.6 that the multi-criteria optimization has no negative effect on the overall model accuracies. As in the single-criteria task before, the optimized SVM classifiers reached a high mean classification accuracy of up to 88%. Interestingly the plot also shows that high computation times do not necessarily lead to good prediction models in terms of accuracy. With wrong settings for *xperc*, the percentage of selected features, even a high runtime can lead to disappointing low gain as the crosses around $Gain = 0.2$ in Fig. 7.6 show.

Similar to the Sonar task, we re-evaluated the non-dominated sets of each run ten times for all variants considered in this study. The differences of the re-evaluated points and the corresponding reference set are shown in Fig. 7.8. For the AppAcid task 200 ML trainings were spent, since more parameters had to be tuned compared with the Sonar task. The different EGO variants are discussed in more detail in the following Sec. 7.4.

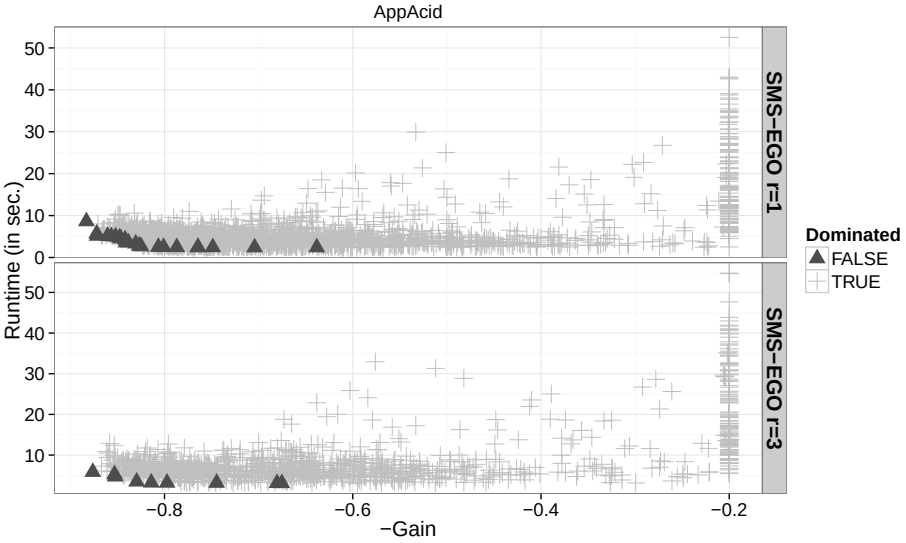


Figure 7.6: Plot of objectives for AppAcid. The mean classification accuracy (*gain*) is plotted on the x-axis and the runtime in seconds on the y-axis. Note, that the negative value for the gain is taken, because all objectives are being minimized.

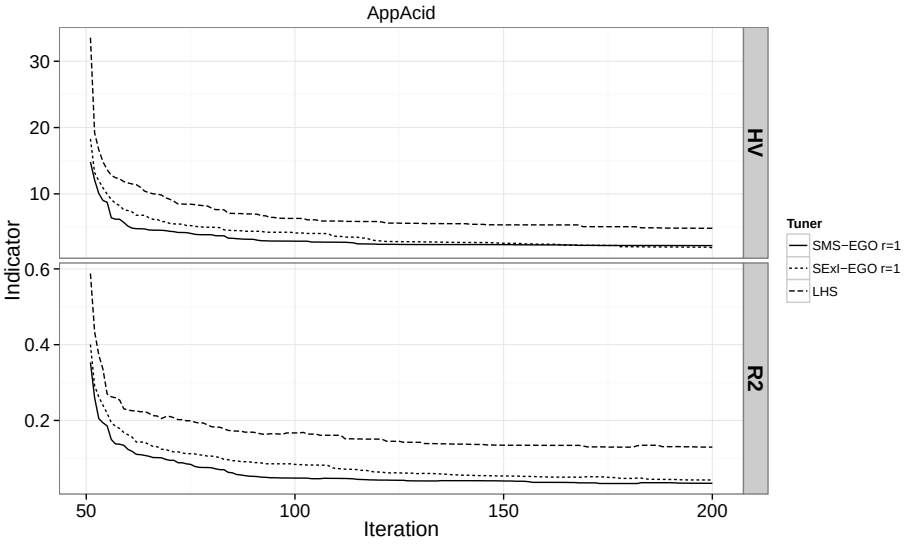


Figure 7.7: AppAcid: Mean HV and R2 development of ten runs of SExI-EGO, SMS-EGO and LHS. The plot shows that the EGO variants approximate quicker to good HV and R2 approximations compared with LHS. The initial design of size 50 is not plotted for the variants, but we show the situation directly after the initial evaluations.

7.4 Discussion

In this section we give a discussion for the multi-criteria tuning using model-assisted optimization algorithms. In Sec. 7.4.1 we review the noise handling of model-assisted MCO and in Sec. 7.4.2 we criticize the runtimes required for the multi-criteria tuning algorithms.

7.4.1 Noise handling

In machine learning it is difficult to obtain a good and reliable estimator of the real generalization error of a prediction model. In our MCO experiments we evaluated the models using the holdout set error (the classification error obtained on a validation set). Although this gives a better estimate than just to consider the training error (which would rather lead to overfitting issues), the objective function values are still subject to high noise levels, because they depend on the randomly sampled training data. Therefore we need to respect special characteristics of ML evaluations, especially highlighting the noisy observations here. We compared three variants to handle noise in optimization under restricted budgets:

(a) the re-interpolation (RI) method by Forrester *et al.* [82], one evaluation of each design point ($r = 1$),

(b) replicates: repeated evaluations of each design point without RI ($r = 3$, this allows only one third of the infills), and

(c) no RI and no replicates ($r = 1$).

The goal is to avoid a too early convergence of the surrogate models to misleading solutions caused by noise.

Variant (a) tries to avoid selecting biased solutions by an internal estimation of the noise. Variant (b) instead explicitly performs re-evaluations to obtain more stable results during the optimization and to give better estimates for the surrogate model predictions. Variant (c) was added as baseline comparison for the special noise handling approaches. In our experiments, the variant using one evaluation ($r = 1$) and noise estimation within the Kriging procedure (re-interpolation) performed worse once on task Sonar (SExI-EGO using three evaluations performed better than SExI-EGO ($r = 3$), Fig. 7.4). Also, SMS-EGO ($r = 3$) performed better on task AppAcid (compared with SMS-EGO ($r = 1$), Fig. 7.8). But it has to be noted, that in both cases the significance level of the t-test did not reach the 95% confidence level for both quality indicators R2 and dominated HV. This means that we cannot give evidence for using $r = 3$ rather than $r = 1$ on any of these tasks or vice versa.

On task AppAcid all EGO variants are significantly better than LHS (t-tests with $p < 0.029$) for the R2 indicator, with the exception of SExI-EGO ($r = 3$). Interestingly, the behaviour for different quality indicators is not necessarily the same, which can be seen

from the result for SExI-EGO (r=1) on task AppAcid. In this boxplot a very large box was obtained for indicator HV, while the box for the R2 indicator is rather small, leading to a significant better result of the EGO approach (t-test for R2 with $p = 0.029$, while the p-value for HV is 0.977).

Very high noise levels were observed for the Sonar task, leading to rather unstable results. As a positive result in Fig. 7.4 the EGO variants reveal smaller boxes, indicating a higher robustness of the optimization compared to LHS. However, in our statistical test (t-test, confidence level 95%) only SExI-EGO ($r = 3$) was significantly better than LHS. The good result of SExI-EGO ($r = 3$) can be reasoned by the usage of repeated evaluations, when the task is very noisy. Instead when the parameter space becomes larger (e.g., task AppAcid), more exploration is required, and the algorithms using replicates do not perform as good (Fig. 7.8). It is also an important achievement that all EGO variants result in more stable quality indicator measures as can be derived from the smaller box sizes in the boxplot in Fig. 7.4.

A point of discussion is the setting of the reference point for the hypervolume indicator. We noticed that this point plays an important role, because it can lead to a perturbation of the results. This most likely occurs, when the reference point is set too close to the points in the approximation set. As a consequence some points do not contribute much to the hypervolume indicator, resulting in a too pessimistic evaluation of the approximation set. Therefore special care must been given to the corner solutions. Such solutions will not contribute to the dominated hypervolume if the reference point is set too close. For this reason we always set the reference point

$$r = (\max_{\vec{x} \in N} f_1(\vec{x}) + C_1, \max_{\vec{x} \in N} f_2(\vec{x}) + C_2) \tag{7.7}$$

where N denotes the set of non-dominated solutions from the reference set and C_i are additional constants for each objective. Here we chose $C_i = 1$. For the R2 indicator we set an utopian point as $(-1, 0)^T$ which is guaranteed to dominate every real solution.

Summarizing we find no significant difference between the noise-dampening variant with $r = 3$ and the other variant $r = 1$ which has more noise but also allows more infills.[1]

RI deals successfully with the observed noise at the $(r = 1)$-level. SMS-EGO with RI performs always slightly better than SMS-EGO without RI, the difference are however not significant in our current experiments. It is matter of future research to reveal whether RI makes a difference compared with simple interpolating Kriging strategies without special noise handling.

[1] As an interesting side remark we note that we *had* a significant difference in the pre-final results: Directly after optimization, the $(r = 1)$-variants seemed significantly better. But this turned out to be only a statistically irrelevant fluctuation of the $(r = 1)$-solutions: After the ten-fold re-evaluation the difference was mainly gone. This emphasizes the importance of sound evaluation procedures in the presence of noise.

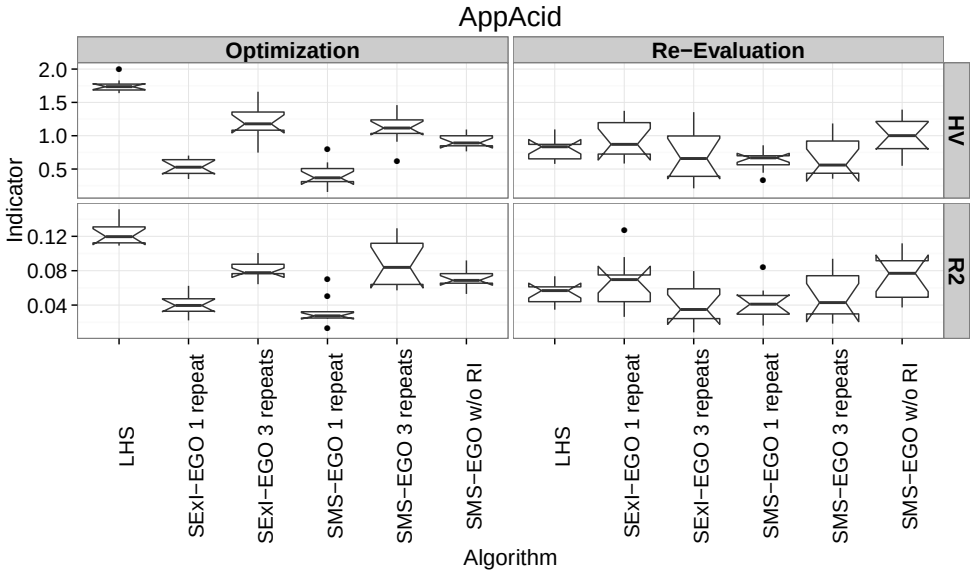


Figure 7.8: R2 and HV quality indicators for ten independent runs on task AppAcid. The non-dominated solutions obtained after 200 function evaluations were evaluated again ten times, each time using different training/validation splittings to remove biased evaluations. For each quality indicator the mean value of these re-evaluations was calculated. Then we determined the differences between the reference set (set of non-dominated solutions of all runs) and the mean quality indicator values. Thus values of zero would be optimal. We compare the EGO variants both with one function evaluation (r=1) and three function evaluations (r=3). Additionally, SMS-EGO was run without re-interpolation and a single evaluation for each design point (SMS-EGO w/o RI).

7.4.2 Parameter space dimension

It is interesting to note the influence of the parameter space dimension on the relative performance of LHS against the multi criteria EGO variants. In the Sonar case (dimension 2) LHS is very close to the EGO results (Fig. 7.4). This is not too surprising, because we spent a large number of 150 function evaluations for the Sonar task. The line plot in Fig. 7.5 shows that the EGO variants could obtain good approximations of the reference set after 50 iterations. This indicates that EGO would also produce good results with less function evaluations, and therefore would probably outperform the LHS. After a sufficient number of function evaluations finally all algorithms achieve a comparable attainment of the reference set (here, after 150 iterations).

In contrast to this, the AppAcid task (dimension 7) shows that LHS cannot keep pace with EGO, because, as Fig. 7.7 shows, the multi criteria EGO indicators now drop much faster: even at iteration 100 (50 initial design + one third of 150), the HV-indicators for both EGO variants are already better than LHS at iteration 200.

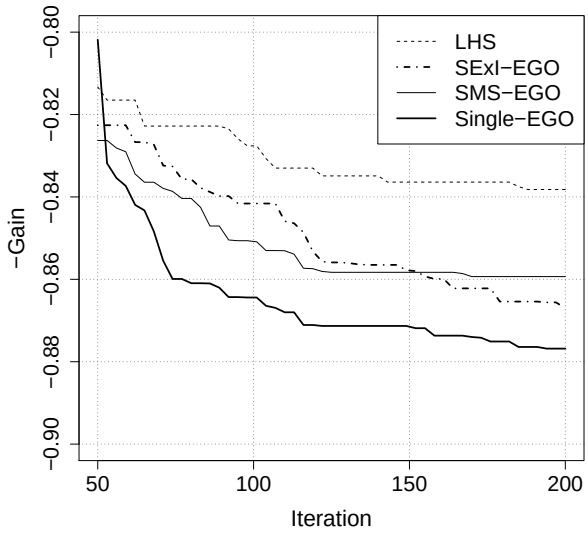


Figure 7.9: AppAcid: Development of the objective *–Gain* during single-criteria EGO (thick line), SMS-EGO and SExI-EGO as two multi-criteria EGO approaches (thin and dash-dotted lines) and LHS as baseline approach. Shown is the average of ten independent runs with different training samples.

7.4.3 Single- and multi-criteria optimization compared

If we perform multi-criteria optimization, is there a large price to pay with respect to the individual goals? If we consider only one objective *gain*, is the best gain delivered from MCO considerably lower than the gain from SCO which concentrates only on this objective? How does the *gain* develop in a restricted-budget optimization?

To answer these questions we performed ten SCO-runs for task AppAcid with the same parameters to optimize, the same RoI and the same budget (50 initial design points, 150 sequential infills) as in the MCO runs. As SCO tuning algorithm we chose the well-known sequential parameter optimization (SPO) [12] within the TDMR-framework [154]. SPO as well uses Kriging as surrogate model, but in this case only for single-criteria optimization of the objective *gain*. We show in Fig. 7.9 the average of ten runs. It is interesting to note that the MCO-gain is only slightly below the SCO-gain which is quite surprising because MCO operates in an objective space bigger by one dimension.[2] But at the same time MCO gives the user more information with respect to the tradeoff between runtime and quality (*gain*).

[2]We note that an earlier SCO-experiment [155] on task AppAcid with method SVM and a slightly different RoI had a mean gain of 86.1%, which is on the same level as the current MCO-results.

7.4.4 EGO runtimes

A disadvantage of hypervolume-based EGO are the high computation times, which can exceed the computation times of the objective function evaluations. E.g., the total runtime of the AppAcid optimization was 15.8 hours with SMS-EGO, while the runtime of SExI-EGO was 143.6 hours (almost 6 days) on a Intel Xeon E5530 CPU running at 2.40GHz. This result was obtained with all settings as described before, having 200 evaluations in total, an initial design size of 50, and one evaluation per design point ($r = 1$). Because the largest runtime for the function evaluations is below one minute, we can conclude that the real objective function cannot be the source for the long runtime. Hence, the optimization on the surrogate model must be responsible for the high computation times. The reasons for this are twofold. At first the Kriging procedure badly scales with increasing parameter sizes. Today, parameter spaces of 20 are challenging. But since we only have 7 tunable parameters, this cannot be the reason. Second, the hypervolume indicator is calculated very often. While the calculation of the hypervolume is NP-hard [29], this bares a real disadvantage for the model-assisted MCO approach. The reason why SExI-EGO has almost a ten times higher runtime, is caused by the underlying implementation. We are aware of the fact that simple modifications of the algorithm can lead to better runtimes. Nevertheless one has to note, that the MCO approach only makes sense when the runtime of the EGO is significantly lower than the runtime of the learning algorithm. In our benchmark this was not the case, but it might become important for really large datasets.

7.5 Conclusions

Summarizing the results, we showed that multi-criteria optimization can help to offer the user a variety of possibile solutions to select from. In the earlier SCO approach the selection was restricted to use the best generalizing model delivered by the optimization procedure. Now, with the multi-criteria approach presented in this chapter, the user is able to select the best models spending a certain limited time budget for the training process.

We applied two hypervolume-based EGO variants, namely SMS-EGO and SExI-EGO, for the first time to noisy ML tasks. It was found that they operate well, even in the presence of considerable noise affirming research question **Q1**. Both algorithms deliver comparable quality in terms of the covered hypervolume and the R2-indicator. SMS-EGO in its current implementation is considerably faster than SExI-EGO.

Both MCO-approaches use Kriging surrogate models to cope with heavily resctricted budgets (**Q2**). Those Kriging models have to work robustly under the presence of noise. Noise is inevitable in most ML tasks due to the variations of randomly drawn training samples and the characteristics of the data. We found it to be crucial to use the re-interpolation technique to cope with the noise confirming **Q3**. Plain Kriging models without re-interpolation (boxplot SMS-EGO without RI in Fig. 7.8) perform worse, because they tend to overfit the noise.

The re-interpolation could deal well with the noise in our benchmark. An additional noise reduction by aggregating repeated evaluations can be necessary, when high noise levels occur. For larger dimensions, repeated evaluations diminish the exploration of the search space, deteriorating the approximation quality due to a reduced number of infill steps.

On the MCO tasks, both Kriging-based EGO techniques performed better than the baseline LHS approach (**Q4**). The advantage increases with the number of dimensions in the parameter space: the difference between LHS quality indicator values and EGO quality indicator values is much higher in task AppAcid (parameter space of dimension 7) than in task Sonar (dimension 2).

In the future we plan to apply the multi criteria approach to other datasets as well, in order to reveal the challenge caused by high dimensions of the parameter space. Additionally, we want to explore the relationship between computation time and classification gain to enable the tuning of ML algorithms in a shorter timeframe.

Chapter 8

Summary

8.1 Contributions of this thesis

In this section we summarize the contributions described in this thesis. Machine learning is a field which has fascinated researchers for many years. It can be seen as a subfield of artificial intelligence, where often the question arises, what computers are able to learn. Today, most desktop computers or even smartphones are capable to learn hidden concepts in the data. But are they also able to find new hidden features and higher-level structures? Or is the knowledge derived from data in ML somehow limited to the data itself — meaning, that up to now supervised learning algorithms can only use features which are provided by the user. The experiments in this thesis showed, that even this can be a challenging task. Efficient tuning algorithms must be used, to solve difficult problems arising in ML. The settings of ML processes are difficult to define and here new research fields seem to emerge. We showed how the incorporation of feature construction methods like Genetic Programming (GP) or the Slow Feature Analysis (SFA) can have an impact on the feature quality. Going deeper into detail here can only be profitable for ML. The variety of experiments exposed several interesting facts, which we now want to summarize in the most relevant contributions:

- **Incorporating pre- and post-processing in the tuning loop (Sec. 4.2, 4.3)** We see the construction of a prediction model as a process, starting with pre-processing transformations like projections in the initial phase. The modeling itself constitutes the main learning phase, and an optional post-processing can be done in the final phase. In each of these steps parameter settings are included, all requiring tuning. We always recommend to optimize these parameters at the same time, instead of tuning them separately from each other. In this thesis it was shown, that each of these parts has a high relevance for the prediction accuracy of the final prediction model. As a consequence we think that tuning must include all relevant parameters to be incorporated in one tuning process.

- **Tuning with model-assisted optimization (Sec. 3.1, 4.3)** The optimization of learning algorithm parameters is expensive, and the runtime can become prohibitively long for large datasets. To mitigate this, we used efficient global optimization for the tuning. In case studies we showed that an optimization even with small budgets (real function evaluations, that is conducting a complete learning process and evaluation on test data) can still yield a good performance. Model-assisted optimization originates from engineering design, where the evaluations of the real function are likewise very expensive. For this reason these methods can be profitably applied to the optimization of learning tasks. One has to be careful that the results of prediction models can be sometimes very noisy, which can be a real challenge for the tuning algorithm. Here, additional techniques were integrated in the tuning process, e.g., the re-interpolation technique, or replications as frequently used in design of experiments (DoE).

- **Time series classification with Slow Feature Analysis (Sec. 3.2)** Time series signals are difficult to classify, especially when they underly high noise terms. Slow Feature Analysis (SFA) extracts the slow varying features of a quickly varying signal and thereby makes it possible to classify such signals. A big advantage of the method is, that it is both fast in training and evaluation of the learned model. This is especially relevant, when a direct response is required, e.g., when a classifier has to be applied in real-time. In our experimental study we classified gestures based on an acceleration sensor (accelerometer) inside a Nintendo Wiimote device. With SFA, we could achieve a considerably high classification accuracy compared with other state-of-the-art learning algorithms like SVM or RF. As a novelty we observed, that SFA requires a sufficient high number of gestures for a good classification, otherwise it merely overfits. In our analysis we indicated that the number of gestures must be larger than the dimension of the expanded feature space set inside SFA. We proposed a strategy based on parametric bootstrapping, that is adding resampled artificial patterns to the training data. Another option can be to decrease the expanded feature space, however leading to a less powerful feature composition. Therefore we recommend to use the first variant, because the latter suffers from a decreased prediction performance.

- **Evolutionary kernel machines (Sec. 4.4)** Choosing the right kernel function is a key step in learning with kernel machines. Although standard kernel functions are available, which work well in most cases, it is of large interest to find an adapted kernel function, which can even be more appropriate. The idea is to learn a kernel function by means of symbolic regression, here using GP. The advantage is, that GP makes it possible to create functions of different complexities using a variety of function sets. However, we observed in a series of experiments, that learning kernel functions for algorithms like SVM is very restrictive. It was possible to detect existing kernel

functions like the linear or polynomial kernel using GP, but the large search space was a barrier to find better working functions at least for our benchmark problems.

- **Time series regression using ML (Sec. 3.1, 4.3)** Time series regression is usually performed with classical methods based on autoregression or moving average. However, sometimes the target variable should not be measured itself, but derived from other measurements. Possible reasons are, that it is expensive to measure the target variable directly, e.g., when chemical analyses need to be incorporated, taking a lot of time and causing additional costs. This situation is ideally suited for learning algorithms, which enable to model such dependencies of the target and input features. Nevertheless learning algorithms are usually not able to model a time series without prior preprocessing. For this reason a technique termed *embedding* was incorporated, which reduces time series problems to standard regression problems. Embedding of features means to add new attributes for the data, based on predecessors of the measured variable. The length of the embedding must be well-defined to achieve good results. We combined this embedding technique with tuning algorithms, for determining the best embedding size and simultaneously optimized the hyperparameters of the learning algorithm. In a case study it was shown, that the tuned learning algorithm performed better than a special-purpose model for the task.

- **Tuning with sub-sampled data (Sec. 5.2)** Tuning can be expensive in terms of computation time. To avoid such expensive tuning runs, we applied a trick, which is based on iterated sub-sampling of the data during the tuning. While it should be clear, that a classifier trained on a subset of the training data probably has not the same performance as a classifier which is trained on the complete training data, we observed that this is not such a big problem for the tuning. The goal of the tuning is to grasp the best tuning parameters, which can be used to build an optimal prediction model. If the performance during the tuning is worse, but the parameters obtained are finally optimal in terms of generalization performance, this is sufficient. We can conclude that if we perform tuning with subsets of the training data, the performance on the validation set is clearly lower compared with using the complete training data. But as soon as we evaluate the best result gathered in the tuning on the final test set (*unbiased evaluation*), we observed that the parameters are not worse compared to using much larger parts of the training data. This really gives evidence for using tuning with sub-sampled data, since the empirical risk is not worsened by the lower performance during tuning.

- **Multi-criteria ML (Chapter 7)** enables a new perspective for the learning task. In most cases the prediction accuracy is considered as single objective in ML. However, as we analyze efficient ML, the learning algorithms should also be efficient in terms of

computation time. It is of large interest, that the learning algorithm can be trained in a reasonable amount of time. Training times lasting longer than days or even weeks are not feasible for most real-world applications, where quick solutions are desired. We propose to use multi-criteria optimization, which can enable a new perspective on ML parametrizations. We defined two criteria in our study, prediction quality and computation time required for building and evaluating the learning algorithm. It was remarkable, that the best result in prediction performance was not worse compared with single criteria tuning. At the same time the analyses showed, that there can be sometimes a large decrease in runtime, with only a minor decrease in accuracy. This is very encouraging for users of ML, as it makes it possible to select between different compromise solutions.

8.2 Discussion

The findings presented in this thesis offer a promising approach to improve prediction models in supervised machine learning. We discuss possible points of criticism in this section.

Overfitting and oversearching

Occam's razor [22, 244] describes the theory that the best description for a concept should be the easiest one when having similar quality. In classification this could be a tree or a linear regression model. However, these learning algorithms are sometimes not capable to create a prediction model, which is powerful enough to separate the data, or fit the function in regression. Reasons for that are, that the data can be often non-linear and noisy. Although simple learning algorithms like linear models or CART can give good hints and are highly interpretable, most people rather stick to more complex models, which are hardly interpretable. Also newer approaches, e.g., Genetic Programming as described in Sec. 4.4, quickly reached their limitations in our experiments. Nevertheless more complex models require tuning, and new obstacles come into play: tuning of state-of-the-art learning algorithms like SVM and RF must be done carefully, because the evaluation inside the tuning can lead to biases towards the training data. This is known as *oversearching* and *overfitting* in the literature [198].

We are aware of the fact that the tuning of learning processes can lead to *oversearching* effects, which has been adressed in Sec. 4.3. Oversearching can be avoided in tuning experiments, e.g., by using resampling, and always having a clear distinction between training, validation and test sets. This distinction is important and helps to retain the generalization performance. We presented the results of repeated experiments, with different random seeds for selecting the data, and thereby tried to pay the highest degree of attention in order to avoid oversearching as far as possible. We think that this topic is important, since tuning should always be implemented with strategies for avoiding oversearching.

Runtimes of tuning experiments

In some tuning experiments it emerged, that runtimes of the tuning algorithms are not neglectable. In our experiments this was especially the case for multi-criteria optimization using Kriging surrogate models. Here, the dominated hypervolume in algorithms like SMS-EGO or SExI-EGO was used as infill criterion, which is known to be computationally expensive when it is calculated very often. It is questionable, whether tuning with hypervolume-based optimization algorithms is still feasible in this case. The high computation times in the multi-criteria experiments showed, that the required times are practical for really large datasets, where the runtime is an important factor.

Dimensionality of the parameter space

Model-assisted optimization using surrogate models is a nice alternative for evaluating the real objective directly. However, surrogate models as Kriging are somehow limited to certain parameter sizes. E.g., a model-assisted optimization in a 50-dimensional space is computationally infeasible for Kriging. Here other surrogate models must be used, or at least the parameter space must be reduced to a lower size.[1]

In the opposite case, tuning of only 2 parameters can easily be done by SPO, but a simple grid search or LHS designs most likely do not perform much worse. However, under very high noise levels, and under very restrictive computational budgets, model-assisted optimization is better suited than simple search strategies. In our experiments model-assisted optimization revealed very stable results for all of the investigated parameter spaces (we tackled real-world problems, and did not include artificially generated benchmarks in this thesis). For this reason it can be argued that for most problem classes occurring in reality, model-assisted optimization is actually a very good choice.

8.3 Conclusions

Parameter tuning of learning processes can be performed efficiently using model-assisted optimization. Combined with techniques for noise handling, the model-assisted optimization outperformed other strategies on a variety of benchmark problems. Tuning itself can be seen as an essential ingredient for improving the performance of learning algorithms. But as the evaluation of learning algorithms can be expensive, model-assisted optimization helps to make the optimization task viable. On our benchmark set the Kriging surrogate models performed best. Their main advantage is that they are almost parameter-free and can model most response surfaces. Even the very high noise levels occurring in ML became tractable by using specific features of Kriging, like the re-interpolation procedure. It was shown, that

[1] It is also questionable, whether 50 parameters in a ML process are reasonable. Nevertheless problems might exist, which require a chained application of pre-processing operators, leading to such a high dimension.

model-assisted optimization not only improves the learning algorithm itself, but the whole learning chain, including feature processing and post-processing operations. By incorporating tuning, also problems, which are not directly suited for statistical learning algorithms — e.g., time series prediction — could be solved with good performance. Methods for feature processing like the Slow Feature Analysis and Genetic Programming were also integrated in a tuning framework and can be regarded as a substantial part of the learning chain. As a new feature, the multi-criteria optimization of runtime and prediction quality opened up a new perspective by presenting users trade-offs of tuned learning algorithms. Summarizing the results, tuning must not be expensive, but can be profitably applied in practice, especially for large-scale learning tasks. From the author's point of view tuning of ML algorithms is the right way to enable comparisons of the algorithms. In the future it must be a goal to simplify comprehensive frameworks like the tuned data mining in R (TDMR) package by Konen *et al.* [155], and to improve them in order to alleviate such comparisons.

Bibliography

[1] T. Abell, Y. Malitsky, and K. Tierney. Fitness landscape based features for exploiting black-box optimization problem structure. Technical Report TR-2012-163, IT University of Copenhagen, 2012.

[2] Y.S. Abu-Mostafa, M. Magdon-Ismail, and H. Lin. *Learning from data*. AMLBook, 2012.

[3] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9(7):1545–1588, 1997.

[4] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, pages 1769–1776. IEEE, 2005.

[5] T. Bäck and M. Schütz. Evolution strategies for mixed-integer optimization of optical multilayer systems. In *Proceedings of the 4th Annual Conference on Evolutionary Programming (EP 1995)*, pages 33–51. MIT Press, 1995.

[6] T. Bäck and H.P. Schwefel. An overview of Evolutionary Algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[7] W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann Publishers, 1997.

[8] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation — The New Experimentalism*. Natural Computing Series. Springer, 2006.

[9] T. Bartz-Beielstein and M. Friese. Sequential parameter optimization and optimal computational budget allocation for noisy optimization problems. Technical Report 01/11, Cologne University of Applied Sciences, 2011.

[10] T. Bartz-Beielstein, M. Friese, M. Zaefferer, B. Naujoks, O. Flasch, W. Konen, and P. Koch. Noisy optimization with sequential parameter optimization and optimal computational budget allocation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 119–120, 2011.

[11] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuß. Sequential parameter optimization. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, volume 1, pages 773–780. IEEE, 2005.

[12] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. The Sequential Parameter Optimization Toolbox. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 337–360. Springer, 2010.

[13] T. Bartz-Beielstein, P. Limbourg, J. Mehnen, K. Schmitt, K.E. Parsopoulos, and M.N. Vrahatis. Particle Swarm Optimizers for Pareto optimization with enhanced archiving techniques. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, volume 3, pages 1780–1787. IEEE, 2003.

[14] T. Bartz-Beielstein, T. Zimmer, and W. Konen. Parameterselektion für komplexe Modellierungsaufgaben der Wasserwirtschaft — Moderne CI-Verfahren

zur Zeitreihenanalyse. In R. Mikut and M. Reischl, editors, *Proceedings of the 18th Workshop on Computational Intelligence*, pages 136–150, 2008.

[15] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[16] P. Berkes. Pattern recognition with Slow Feature Analysis. Cognitive Sciences EPrint Archive (CogPrint) `http://cogprints.org/4104/`, 2005.

[17] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, volume 2, pages 11–18, 2002.

[18] B. Bischl, O. Mersmann, and H. Trautmann. Resampling methods in model validation. In *Proceedings of the Workshop on Experimental Methods for the Assessment of Computational Systems (WEMACS)*, pages 14–31, 2010.

[19] B. Bischl, O. Mersmann, H. Trautmann, and M. Preuß. Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 313–320, 2012.

[20] B. Bischl, O. Mersmann, H. Trautmann, and C. Weihs. Resampling methods for meta-model validation with recommendations for Evolutionary Computation. *Evolutionary Computation*, 20(2):249–275, 2012.

[21] C.M. Bishop. *Pattern Recognition and Machine Learning*, volume 1. Springer, 2006.

[22] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, 1987.

[23] G. Bo and H. Xianwu. SVM multi-class classification. *Journal of Data Acquisition & Processing*, 21(3):334–339, 2006.

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

[25] L. Breiman. Technical note: some properties of splitting criteria. *Machine Learning*, 24(1):41–47, 1996.

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

[27] L. Breiman. Manual — setting up, using, and understanding Random Forests. Technical Report V3.1, University of California, Berkeley, 2002.

[28] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

[29] K. Bringmann and T. Friedrich. Approximating the least hypervolume contributor: NP-hard in general, but fast in practice. *Theoretical Computer Science*, 425:104–116, 2012.

[30] D. Brockhoff, A. Auger, N. Hansen, D. Arnold, and T. Hohm. Mirrored sampling and sequential selection for Evolution Strategies. In *Proceedings of Parallel Problem Solving from Nature (PPSN)*, pages 11–21. Springer, 2011.

[31] P.J. Brockwell and R.A. Davis. *Introduction to Time Series and Forecasting*. Taylor & Francis, 2nd edition, 2002.

[32] C.G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92):577–593, 1965.

[33] C. Buck, T. Gass, A. Hannig, J. Hosang, S. Jonas, J.T. Peter, P. Steingrube, and J.H. Ziegeldorf. Data-Mining-Cup 2007. *Informatik-Spektrum*, 31(6):591–599, 2008.

[34] L.T. Bui, H.A. Abbass, and D. Essam. Fitness inheritance for noisy evolutionary multi-objective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 779–785. ACM, 2005.

[35] K.P. Burnham and D.R. Anderson. *Model Selection And Multimodel Inference: A Practical Information-Theoretic Approach*. Springer, 2nd edition, 2002.

[36] C. Campbell, N. Cristianini, and J. Shawe-Taylor. Dynamically adapting kernels in Support Vector Machines. *Advances in Neural Information Processing Systems*, 11:204–210, 1999.

[37] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for Support Vector Machines. *Machine Learning*, 46(1):131–159, 2002.

[38] B.B. Chaudhuri and U. Bhattacharya. Efficient training and improved performance of multilayer perceptron in pattern classification. *Neurocomputing*, 34(1):11–27, 2000.

[39] H.C. Chen, L. Dai, C.H. Chen, and E. Yucesan. New development of optimal computing budget allocation for discrete event simulation. In *Proceedings of the 1997 Winter Simulation Conference*, pages 334–341. IEEE, 1997.

[40] P.W. Chen, J.Y. Wang, and H.M. Lee. Model selection of SVMs using GA approach. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 3, pages 2035–2040. IEEE, 2004.

[41] V. Cherkassky and Y. Ma. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, 17(1):113–126, 2004.

[42] V. Cherkassky and F.M. Mulier. *Learning from data: concepts, theory, and methods*. Wiley, 2007.

[43] A. Christmann, K. Luebke, M. Marin-Galiano, and S. Rüping. Determination of hyper-parameters for kernel based classification and regression. Technical Report 13/2005, TU Dortmund, 2005.

[44] W.G. Cochran. *Sampling Techniques*. Wiley-India, 3rd edition, 2007.

[45] C.A. Coello Coello, G.B. Lamont, and D.A. Van Veldhuizen. *Evolutionary Algorithms for solving multi-objective problems*, volume 5. Springer, 2007.

[46] G. Cohen, M. Hilario, and A. Geissbuhler. Model selection for support vector classifiers via Genetic Algorithms. An application to medical decision support. *Biological and Medical Data Analysis*, pages 200–211, 2004.

[47] C. Cortes, P. Haffner, and M. Mohri. Positive definite rational kernels. In *Proceedings of the Conference on Computational Learning Theory (COLT)*, volume 1, pages 41–56, 2003.

[48] C. Cortes, P. Haffner, and M. Mohri. Rational kernels. *Advances in Neural Information Processing Systems*, pages 617–624, 2003.

[49] C. Cortes, P. Haffner, and M. Mohri. Rational kernels: theory and algorithms. *Journal of Machine Learning Research (JMLR)*, 5:1035–1062, 2004.

[50] C. Cortes, M. Mohri, and A. Rostamizadeh. Learning non-linear combinations of kernels. *Advances in Neural Information Processing Systems*, 22:396–404, 2009.

[51] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[52] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[53] W. Daelemans, V. Hoste, F. De Meulder, and B. Naudts. Combined optimization of feature selection and algorithm parameters in machine learning of language. In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 84–95, 2003.

[54] W.C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, 1991.

[55] B.F. de Souza, A. de Carvalho, R. Calvo, and R.P. Ishii. Multiclass SVM model selection using Particle Swarm Optimization. In *Proceedings of the International Conference on Hybrid Intelligent Systems (HIS)*, pages 31–35. IEEE, 2006.

[56] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[57] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[58] T.G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

[59] E. Dimitriadou, K. Hornik, F. Leisch, D. Meyer, and A. Weingessel. e1071: misc functions of the Department of Statistics (e1071), TU Wien. *e1071 R package*, pages 1–60, 2006.

[60] L. Diosan, A. Rogozan, and J.P. Pecuchet. Evolving kernel functions for SVMs by Genetic Programming. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA)*, pages 19–24. IEEE, 2007.

[61] P. Domingos. MetaCost: a general method for making classifiers cost-sensitive. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD 1999)*, pages 155–164. ACM, 1999.

[62] S. Droste and D. Wiesmann. Metric based Evolutionary Algorithms. In *Proceedings of the European Conference on Genetic Programming*, pages 29–43. Springer, 2000.

[63] H. Drucker, C.J.C. Burges, L. Kaufman, A. Smola, and V.N. Vapnik. Support Vector Regression Machines. *Advances in Neural Information Processing Systems*, pages 155–161, 1997.

[64] K. Duan, S. Keerthi, and A.N. Poo. Evaluation of simple performance measures for tuning SVM hyperparameters. *Neurocomputing*, 51:41–59, 2003.

[65] A. Dubrawski and J. Schneider. Memory based stochastic optimization for validation and tuning of function approximators. In *Proceedings of the Conference on AI and Statistics*, 1997.

[66] B. Efron. Bootstrap methods: another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

[67] B. Efron and R.J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 1st edition, 1994.

[68] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 17, pages 973–978, 2001.

[69] M. Emmerich, A.H. Deutz, and J.W. Klinkenberg. Hypervolume-based expected improvement: Monotonicity properties and exact computation. In *Proceedings of Congress on Evolutionary Computation (CEC)*, pages 2147–2154. IEEE, 2011.

[70] M. Emmerich, M. Grötzner, B. Groß, and M. Schütz. Mixed-integer evolution strategy for chemical plant optimization with simulators. *Evolutionary Design and Manufacture*, pages 55–67, 2000.

[71] H. Eskandari and C.D. Geiger. Evolutionary multiobjective optimization in noisy problem environments. *Journal of Heuristics*, 15(6):559–595, 2009.

[72] M.J.A. Eugster, T. Hothorn, and F. Leisch. Exploratory and inferential analysis of benchmark experiments. Technical Report 030, Department of Statistics, University of Munich, 2008.

[73] M.J.A. Eugster and F. Leisch. Bench Plot and Mixed Effects Models: First steps towards a comprehensive benchmark analysis toolbox. In *Proceedings of Computational Statistics (Compstat)*, volume 26, 2008.

[74] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and Support Vector Machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

[75] J.E. Fieldsend and R.M. Everson. Multi-objective optimisation in the presence of uncertainty. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, volume 1, pages 243–250. IEEE, 2005.

[76] R.A. Fisher. Design of Experiments. *British Medical Journal*, 1(3923):554–554, 1936.

[77] P. Flach. On the state of the art in machine learning: a personal review. *Artificial Intelligence*, 131(1):199–222, 2001.

[78] P. Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

[79] O. Flasch, T. Bartz-Beielstein, P. Koch, and W. Konen. Genetic Programming applied to predictive control in environmental engineering. In F. Hoffmann and E. Hüllermeier, editors, *Proceedings of the 19th Workshop on Computational Intelligence*, pages 101–113. KIT Scientific Publishing, 2009.

[80] O. Flasch, O. Mersmann, and T. Bartz-Beielstein. RGP: an open source Genetic Programming system for the R environment. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 2071–2072. ACM, 2010.

[81] R. Fletcher and M.J.D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168, 1963.

[82] A. Forrester, A. Keane, and N.W. Bressloff. Design and analysis of "noisy" computer experiments. *Journal of the American Institute of Aeronautics and Astronautics*, 44(10):23–31, 2006.

[83] A. Forrester, A. Sobester, and A. Keane. *Engineering design via surrogate modelling: a practical guide*. Wiley, 2008.

[84] A. Frank and A. Asuncion. UCI Machine Learning Repository, 2010.

[85] A.A. Freitas. A critical review of multi-objective optimization in Data Mining: a position paper. *ACM SIGKDD Explorations Newsletter*, 6(2):77–86, 2004.

[86] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[87] J.H. Friedman. Multivariate Adaptive Regression Splines. *The Annals of Statistics*, pages 1–67, 1991.

[88] F. Friedrichs and C. Igel. Evolutionary tuning of multiple SVM parameters. *Neurocomputing*, 64:107–117, 2005.

[89] C. Gagné, M. Schoenauer, M. Sebag, and M. Tomassini. Genetic Programming for kernel-based learning with co-evolving subsets selection. In *Proceedings of the Conference on Parallel Problem Solving from Nature (PPSN)*, pages 1008–1017. Springer, 2006.

[90] Y. Ganjisaffar, T. Debeauvais, S. Javanmardi, R. Caruana, and C.V. Lopes. Distributed tuning of machine learning algorithms using MapReduce clusters. In *Proceedings of the Workshop on Large Scale Data Mining: Theory and Applications*, pages 1–8. ACM, 2011.

[91] K.C. Giannakoglou, A.P. Giotis, and M.K. Karakasis. Low-cost genetic optimization based on inexact pre-evaluations and the sensitivity analysis of design parameters. *Inverse Problems in Engineering*, 9(4):389–412, 2001.

[92] C. Gini. Variability and mutability. *Journal of the Royal Statistical Society*, 76(3):326–327, 1912.

[93] T. Glasmachers and C. Igel. Gradient-based adaptation of general Gaussian kernels. *Neural Computation*, 17(10):2099–2105, 2005.

[94] T. Glasmachers and C. Igel. Uncertainty handling in model selection for Support Vector Machines. In *Proceedings of the Conference on Parallel Problem Solving from Nature (PPSN)*, pages 185–194. Springer, 2008.

[95] T. Glasmachers and C. Igel. Maximum likelihood model selection for 1-norm soft margin SVMs with multiple parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1522–1528, 2010.

[96] C.K. Goh and K.C. Tan. *Evolutionary Multi-Objective Optimization in Uncertain Environments: Issues and Algorithms*, volume 186 of *Studies in Computational Intelligence*. Springer, 2009.

[97] C. Gold and P. Sollich. Model selection for Support Vector Machine classification. *Neurocomputing*, 55(1):221–249, 2003.

[98] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[99] D. Goldfarb. A family of variable metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[100] G.H. Golub and C.F. Van Loan. *Matrix Computations*, volume 3. JHU Press, 2012.

[101] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[102] L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[103] M.P. Hansen and A. Jaszkiewicz. Evaluating the quality of approximations to the non-dominated set. Technical Report IMM-REP-1998-7, Institute of Mathematical Modelling, Technical University of Denmark, 1998.

[104] N. Hansen. The CMA Evolution Strategy: A Comparing Review. *Studies in Fuzziness and Soft Computing*, 192:75–102, 2006.

[105] N. Hansen, A.S.P. Niederberger, L. Guzzella, and P. Koumoutsakos. Evolutionary optimization of feedback controllers for thermoacoustic instabilities. In *IUTAM Symposium on Flow Control and MEMS*, pages 311–317. Springer, 2008.

[106] N. Hansen, A.S.P. Niederberger, L. Guzzella, and P. Koumoutsakos. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13(1):180–197, 2009.

[107] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in Evolution Strategies: the Covariance Matrix Adaptation. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, pages 312–317. IEEE, 1996.

[108] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in Evolution Strategies. *Evolutionary Computation*, 9:159–195, 2001.

[109] S.S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2007.

[110] W. He, Z. Wang, and H. Jiang. Model optimizing and feature selecting for Support Vector Regression in time series forecasting. *Neurocomputing*, 72(1):600–611, 2008.

[111] G.E. Hinton, S. Osindero, and Y.W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[112] T.K. Ho. Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282. IEEE, 1995.

[113] T.K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[114] K. Hornik and D. Meyer. Deriving consensus rankings from benchmarking experiments. *Advances in Data Analysis*, pages 163–170, 2007.

[115] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, 1933.

[116] T. Hothorn, F. Leisch, A. Zeileis, and K. Hornik. The design and analysis of benchmark experiments. *Journal of Computational and Graphical Statistics*, 14(3):675–699, 2005.

[117] T. Howley and M.G. Madden. The genetic kernel Support Vector Machine: description and evaluation. *Artificial Intelligence Review*, 24(3):379–395, 2005.

[118] C.W. Hsu, C.C. Chang, and C.J. Lin. A practical guide to Support Vector Classification. Technical Report 15/2010, Department of Computer Science, National Taiwan University, 2003.

[119] 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.

[120] E. Hughes. Evolutionary multi-objective ranking with uncertainty and noise. In *Proceedings of the Evolutionary Multi-Criterion Optimization Conference (EMO)*, pages 329–343. Springer, 2001.

[121] F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. *Learning and Intelligent Optimization*, pages 507–523, 2011.

[122] F. Hutter, H. Hoos, K. Leyton-Brown, and K.P. Murphy. An experimental investigation of model-based parameter optimisation: SPO and beyond. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 271–278. ACM, 2009.

[123] F. Hutter, H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, volume 2, pages 1152–1157. MIT Press, 2007.

[124] F. Imbault and K. Lebart. A stochastic optimization approach for parameter tuning of Support Vector Machines. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, volume 4, pages 597–600. IEEE, 2004.

[125] A. Jain and D. Zongker. Feature selection: evaluation, application, and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):153–158, 1997.

[126] R. Jin, W. Chen, and T.W. Simpson. Comparative studies of metamodeling techniques under multiple modelling criteria. *Structural and Multidisciplinary Optimization*, 23(1):1–13, 2001.

[127] R. Jin, X. Du, and W. Chen. The use of metamodeling techniques for optimization under uncertainty. *Structural and Multidisciplinary Optimization*, 25(2):99–116, 2003.

[128] Y. Jin. Pareto-based multi-objective machine learning. In *Proceedings of the International Conference on Hybrid Intelligent Systems (HIS)*, pages 397–415. IEEE, 2007.

[129] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments — a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[130] Y. Jin and B. Sendhoff. Pareto-based multiobjective machine learning: an overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(3):397–415, 2008.

[131] T. Joachims. Estimating the generalization performance of a SVM efficiently. Technical Report 25, TU Dortmund, 2000.

[132] D.R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.

[133] D.R. Jones, M. Schonlau, and W.J. Welch. Efficient Global Optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[134] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for Genetic Algorithms. In *Proceedings of the International Conference on Genetic Algorithms (ICGA)*, pages 184–192, 1995.

[135] H. Kantz and T. Schreiber. *Nonlinear Time Series Analysis*. Cambridge University Press, 2004.

[136] A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab — An S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.

[137] A. Keane. Surrogate-based optimization. *Encyclopedia of Aerospace Engineering*, 2010.

[138] S. Keerthi. Efficient tuning of SVM hyperparameters using radius/margin bound and iterative algorithms. *IEEE Transactions on Neural Networks*, 13(5):1225–1229, 2002.

[139] S. Keerthi, V. Sindhwani, and O. Chapelle. An efficient method for gradient-based adaptation of hyperparameters in SVM models. *Advances in Neural Information Processing Systems*, 19, 2007.

[140] J. Kennedy and R. Eberhart. Particle Swarm Optimization. In *Proceedings of the International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[141] B.S. Kim, Y.B. Lee, and D.H. Choi. Comparison study on the accuracy of metamodeling technique for non-convex functions. *Journal of Mechanical Science and Technology*, 23(4):1175–1181, 2009.

[142] K. Kim. Financial time series forecasting using Support Vector Machines. *Neurocomputing*, 55(1):307–319, 2003.

[143] S. Kirkpatrick, C.D. Gelatt Jr, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[144] 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.

[145] J. Knowles, D. Corne, and A. Reynolds. Noisy multiobjective optimization on a budget of 250 evaluations. In *Proceedings of the Conference on Evolutionary Multi-Criterion Optimization (EMO)*, pages 36–50. Springer, 2009.

[146] J. Knowles and H. Nakayama. Meta-modeling in multiobjective optimization. *Multiobjective Optimization*, pages 245–284, 2008.

[147] P. Koch, T. Bartz-Beielstein, and Konen W. Optimization of Support Vector Regression models for stormwater prediction. In F. Hoffmann and E. Hüllermeier, editors, *Proceedings of the 20th Workshop on Computational Intelligence*. Universitätsverlag Karlsruhe, 2010.

[148] P. Koch, B. Bischl, O. Flasch, T. Bartz-Beielstein, C. Weihs, and W. Konen. Tuning and evolution of support vector kernels. *Evolutionary Intelligence*, pages 1–18, 2011.

[149] P. Koch and W. Konen. Efficient sampling and handling of variance in tuning Data Mining models. In V. Cutello and M. Pavone, editors, *Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN)*. Springer, 2012.

[150] P. Koch, W. Konen, O. Flasch, and T. Bartz-Beielstein. Optimizing Support Vector Machines for Stormwater Prediction. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Proceedings of the Workshop on Experimental Methods for the Assessment of Computational Systems (WEMACS)*, number TR10-2 in 007, pages 47–59, TU Dortmund, 2010.

[151] P. Koch, W. Konen, and K. Hein. Gesture recognition on few training data using Slow Feature Analysis and parametric bootstrap. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2010.

[152] R. Kohavi and G.H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324, 1997.

[153] W. Konen. On the numeric stability of the SFA implementation `sfa-tk`. Technical Report 05-10, Cologne University of Applied Sciences, 2009.

[154] W. Konen. The TDM Framework: Tuned Data Mining in R. Technical Report 01-11, Cologne University of Applied Sciences, 2011.

[155] W. Konen, P. Koch, O. Flasch, T. Bartz-Beielstein, M. Friese, and B. Naujoks. Tuned Data Mining: A benchmark study on different tuners. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1995–2002. ACM, 2011.

[156] W. Konen, T. Zimmer, and T. Bartz-Beielstein. Optimierte Modellierung von Füllständen in Regenüberlaufbecken mittels CI-basierter Parameterselektion. *at — Automatisierungstechnik*, 57(3):155–166, 2009.

[157] M. Kotanchek, G. Smits, and E. Vladislavleva. Pursuing the Pareto paradigm: tournaments, algorithm variations and ordinal optimization. In R. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice IV*, pages 167–185. Springer, 2007.

[158] J.R. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, 1992.

[159] Krzysztof Krawiec. Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genetic Programming and Evolvable Machines*, 3(4):329–343, 2002.

[160] D.G. Krige. *A statistical approach to some mine valuation and allied Ppoblems on the Witwatersrand*. PhD thesis, University of the Witwatersrand, 1951.

[161] M. Last. Improving Data Mining utility with projective sampling. In *Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining (SIGKDD 2009)*, pages 487–496. ACM, 2009.

[162] J. C. Lee. Hacking the Nintendo Wii Remote. *Pervasive Computing*, 7(3):39–45, 2008.

[163] L.H. Lee, E.P. Chew, S. Teng, and D. Goldsman. Optimal computing budget allocation for multi-objective simulation models. In *Proceedings of the Winter Simulation Conference*, volume 1. IEEE, 2004.

[164] R. V. Lenth. Some practical guidelines for effective sample size determination. *The American Statistician*, 55(3):187–193, 2001.

[165] R. Li, M. Emmerich, J. Eggermont, and E.G.P. Bovenkamp. Mixed-integer optimization of coronary vessel image analysis using Evolution Strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1645–1652. ACM, 2006.

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

[167] G.P. Liu and V. Kadirkamanathan. Learning with multi-objective criteria. In *Proceedings of the 4th International Conference on Artificial Neural Networks*, pages 53–58, 1995.

[168] J. Liu, Z. Wang, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uWave: Accelerometer-based personalized gesture recognition and its applications. In *Proceedings of the Conference on Pervasive Computing and Communications*, volume 1, pages 1–9. IEEE, 2009.

[169] W. Loh and Y. Shih. Split selection methods for classification trees. *Statistica Sinica*, 7:815–840, 1997.

[170] A.A.T. Macrobius and M.T. Cicero. *Commentarii in somnium Scipionis*. Liviana, 1981.

[171] P. Malmestig and S. Sundberg. SignWiiver — implementation of sign language technology, 2008.

[172] R.B. Marimont and M.B. Shapiro. Nearest Neighbour searches and the curse of dimensionality. *Journal of Applied Mathematics*, 24(1):59–70, 1979.

[173] O. Maron and A. Moore. Hoeffding races: accelerating model selection search for classification and function approximation. *Advances in Neural Information Processing Systems*, 6:59–66, 1994.

[174] G. Matheron. Principles of Geostatistics. *Economic Geology*, 58(8):1246–1266, 1963.

[175] M.D. McKay, R.J. Beckman, and W.J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, pages 239–245, 1979.

[176] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society*, 209:415–446, 1909.

[177] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph. Exploratory landscape analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 829–836. ACM, 2011.

[178] O. Mersmann, M. Preuss, and H. Trautmann. Benchmarking Evolutionary Algorithms: towards exploratory landscape analysis. In *Proceedings of the Conference on Parallel Problem Solving from Nature (PPSN)*, pages 73–82. Springer, 2011.

[179] P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4):337–352, 2000.

[180] H. Minh, P. Niyogi, and Y. Yao. Mercer's theorem, feature maps, and smoothing. *Learning Theory*, pages 154–168, 2006.

[181] A.W. Moore, J. Schneider, et al. Memory-based stochastic optimization. *Advances in Neural Information Processing Systems*, pages 1066–1072, 1996.

[182] S. Mukherjee, E. Osuna, and F. Girosi. Nonlinear prediction of chaotic time series using Support Vector Machines. In *Proceedings of the Workshop on Neural Networks for Signal Processing*, pages 511–520. IEEE, 1997.

[183] K. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V.N. Vapnik. Predicting time series with Support Vector Machines. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN 1997)*, pages 999–1004. Springer, 1997.

[184] V. Nannen and A.E. Eiben. Relevance estimation and value calibration of Evolutionary Algorithm parameters. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 975–980. Morgan Kaufmann Publishers Inc., 2007.

[185] J.A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[186] J. Neyman. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society*, 97(4):558–625, 1934.

[187] T. Oates and D. Jensen. The effects of training set size on decision tree complexity. In *Proceedings of the International Conference on Machine Learning (ICML 1997)*, pages 254–262. Morgan Kaufmann Publishers Inc., 1997.

[188] F. Ojeda, J.A.K. Suykens, and B. De Moor. Low rank updated LS-SVM classifiers for fast variable selection. *Neural Networks*, 21(2–3):437–449, 2008.

[189] K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[190] L.S. Penrose. The elementary statistics of majority voting. *Journal of the Royal Statistical Society*, 109(1):53–57, 1946.

[191] V. Picheny, D. Ginsbourger, and Y. Richet. Noisy expected improvement and on-line computation time allocation for the optimization of simulators with tunable fidelity. In *Proceedings of the 2nd International Conference on Engineering Optimization*, pages 1–10, 2010.

[192] V. Picheny, T. Wagner, and D. Ginsbourger. A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization*, 2012.

[193] R. Poli, W.B. Langdon, and N. Freitag McPhee. *A field guide to Genetic Programming*. Published via `http://lulu.com`, 2008.

[194] W. Ponweiser, T. Wagner, D. Biermann, and M. Vincze. Multiobjective optimization on a limited budget of evaluations using model-assisted $\mathcal{S}$-metric selection. In *Proceedings of the Conference on Parallel Problem Solving from Nature (PPSN)*, pages 784–794. Springer, 2008.

[195] B. Poppinga. Wiigee, a Java-based gesture recognition library for the Wii remote. Technical Report 09/07, University of Oldenburg, 2007.

[196] F. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (SIGKDD 1999)*, pages 23–32. ACM, 1999.

[197] J.R. Quinlan. *C4.5: programs for machine learning*, volume 1. Morgan Kaufmann, 1993.

[198] J.R. Quinlan and R. Cameron-Jones. Oversearching and layered search in empirical learning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1019–1024. Morgan Kaufmann, 1995.

[199] R Core Team. *R: a language and environment for statistical computing*. R Foundation for Statistical Computing, 2012.

[200] L. Rabiner and B. Juang. An introduction to Hidden Markov Models. *ASSP Magazine*, 3(1):4–16, 1986.

[201] C.E. Rasmussen and C.K.I. Williams. *Gaussian processes for machine learning*, volume 1. MIT Press, 2006.

[202] S. J. Raudys and A. K. Jain. Small sample size effects in statistical pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):252–264, 1991.

[203] I. Rechenberg. Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment Library Translation*, 1122, 1965.

[204] M. Rehm, N. Bee, and E. André. Wave like an egyptian — accelerometer based gesture recognition for culture specific interactions. In *Procedings of the Conference on Human Computer Interaction (HCI 2008)*, 2008.

[205] J.R. Rice. The algorithm selection problem. Technical Report CSD-TR 152, Purdue University, 1975.

[206] J.P. Rosca and D.H. Ballard. Causality in Genetic Programming. In *Proceedings of the International Conference on Genetic Algorithms (ICGA)*, pages 256–263. Morgan Kaufmann, 1995.

[207] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Reviews*, 65(6):386–408, 1958.

[208] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*, volume 74. Prentice Hall International, 1995. 3rd revised edition.

[209] J. Sacks, W.J. Welch, T.J. Mitchell, and H.P. Wynn. Design and analysis of computer experiments. *Statistical science*, 4(4):409–423, 1989.

[210] N. Sapankevych and R. Sankar. Time series prediction using Support Vector Machines: a survey. *IEEE Computational Intelligence Magazine*, 4(2):24–38, 2009.

[211] R.E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[212] R.E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[213] T. Schlömer, B. Poppinga, N. Henze, and S. Boll. Gesture recognition with a Wii controller. In *Proceedings of the Tangible and Embedded Interaction Conference (TEI)*, pages 11–14, 2008.

[214] B. Schölkopf, C.J.C. Burges, and A.J. Smola. *Advances in kernel methods: Support Vector learning*. The MIT press, 1999.

[215] B. Schölkopf and A.J. Smola. *Learning with kernels: Support Vector Machines, regularization, optimization, and beyond*. MIT Press, 2002.

[216] O. Schütze, X. Esquivel, A. Lara, and C. Coello. Using the averaged Hausdorff distance as a performance measure in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 16(4):504–522, 2012.

[217] H.P. Schwefel. Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik. Master's thesis, TU Berlin, 1965.

[218] M.R. Segal. Machine learning benchmarks and Random Forest regression. Technical Report CA 94143-0560, Center for Bioinformatics and Molecular Biostatistics, UC San Francisco, 2004.

[219] D.F. Shanno and P.C. Kettler. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[220] H. Shin and S. Cho. Fast pattern selection for Support Vector classifiers. *Advances in Knowledge Discovery and Data Mining*, pages 567–567, 2003.

[221] M. Signoretto and J. Suykens. Kernel methods. *Handbook on Computational Intelligence*, 2013.

[222] S. Smit and A.E. Eiben. Beating the world champion Evolutionary Algorithm via REVAC tuning. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2010.

[223] S. Smit and A.E. Eiben. Parameter tuning of Evolutionary Algorithms: generalist vs. specialist. *Applications of Evolutionary Computation*, pages 542–551, 2010.

[224] M.G. Smith and L. Bull. Genetic Programming with a Genetic Algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines*, 6(3):265–281, 2005.

[225] G. Smits, A. Kordon, K. Vladislavleva, E. Jordaan, and M. Kotanchek. Variable selection in industrial datasets using Pareto Genetic Programming. *Genetic Programming Theory and Practice III*, pages 79–92, 2006.

[226] G. Smits and M. Kotanchek. Pareto-front exploitation in symbolic regression. *Genetic Programming Theory and Practice II*, pages 283–299, 2005.

[227] G. Smits and E. Vladislavleva. Ordinal Pareto Genetic Programming. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, pages 3114–3120. IEEE, 2006.

[228] A.J. Smola and B. Schölkopf. A tutorial on Support Vector Regression. *Statistics and Computing*, 14(3):199–222, 2004.

[229] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, 1:194–281, 1986.

[230] P. Sollich. Bayesian methods for Support Vector Machines: evidence and predictive class probabilities. *Machine Learning*, 46(1-3):21–52, 2002.

[231] R. Storn and K. Price. Differential Evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[232] C. Strobl, A.L. Boulesteix, T. Kneib, T. Augustin, and A. Zeileis. Conditional variable importance for Random Forests. *BMC Bioinformatics*, 9(1):307, 2008.

[233] A. Stuhlsatz, C. Meyer, F. Eyben, T. Zielke, G. Meier, and B. Schuller. Deep Neural Networks for acoustic emotion recognition: raising the benchmarks. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP 2011)*, pages 5688–5691. IEEE, 2011.

[234] K.M. Sullivan and S. Luke. Evolving kernels for Support Vector Machine classification. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, page 1707. ACM, 2007.

[235] H. Sun. Mercer theorem for RKHS on noncompact sets. *Journal of Complexity*, 21(3):337–349, 2005.

[236] J. Sun, C. Zheng, X. Li, and Y. Zhou. Analysis of the distance between two classes for tuning SVM hyperparameters. *IEEE Transactions on Neural Networks*, 21(2):305–318, 2010.

[237] A. Syberfeldt, A. Ng, R.I. John, and P. Moore. Evolutionary optimisation of noisy multi-objective problems using confidence-based dynamic resampling. *European Journal of Operational Research*, 204(3):533–544, 2010.

[238] J. Teich. Pareto-front exploration with uncertain objectives. In *Proceedings of the Conference on Evolutionary Multi-Criterion Optimization (EMO)*, pages 314–328. Springer, 2001.

[239] H. Theil. *Economics and information theory*. North-Holland Publishing Company, 1967.

[240] U. Thissen, R. Van Brakel, A.P. De Weijer, W.J. Melssen, and L.M.C. Buydens. Using Support Vector Machines for time series prediction. *Chemometrics and Intelligent Laboratory Systems*, 69(1):35–49, 2003.

[241] E.R. Tufte and P.R. Graves-Morris. *The visual display of quantitative information*, volume 31. Graphics Press, 1983.

[242] P. Turney. Types of cost in inductive concept learning. In *Proceedings of the Cost-Sensitive Learning Workshop*, pages 1–7, 2000.

[243] G. Valentini and T.G. Dietterich. Bias-variance analysis of Support Vector Machines for the development of SVM-based ensemble methods. *The Journal of Machine Learning Research*, 5:725–775, 2004.

[244] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[245] V.N. Vapnik. *The nature of statistical learning theory*. Springer, 2nd edition, 1995.

[246] V.N. Vapnik. *Statistical learning theory*. John Wiley and Sons Inc., 1st edition, 1998.

[247] E. Vazquez, J. Villemonteix, M. Sidorkiewicz, and E. Walter. Global optimization based on noisy evaluations: an empirical study of two statistical approaches. *Journal of Physics*, 35(1):1–8, 2008.

[248] T. Wagner. A subjective review of the state of the art in model-based parameter tuning. In *Proceedings of the Workshop on Experimental Methods for the Assessment of Computational Systems (WEMACS)*, pages 1–13, 2010.

[249] T. Wagner, M. Emmerich, A. Deutz, and W. Ponweiser. On expected-improvement criteria for model-based multi-objective optimization. In *Proceedings of the Conference on Parallel Problem Solving from Nature (PPSN)*, pages 718–727. Springer, 2010.

[250] T. Wagner and S. Wessing. On the effect of response transformations in Sequential Parameter Optimization. *Evolutionary Computation*, 20(2):229–248, 2011.

[251] D. Wiesmann. From syntactical to semantical mutation operators for structure optimization. In *Proceedings of the Conference on Parallel Problem Solving from Nature (PPSN)*, pages 234–246. Springer, 2002.

[252] L. Wiskott. Learning invariance manifolds. In *Proceedings of the Joint Symposium on Neural Computation*, volume 8, pages 196–203. Univ. of California, 1998.

[253] L. Wiskott. Estimating driving forces of nonstationary time series with Slow Feature Analysis. arXiv.org e-Print archive, `http://arxiv.org/abs/cond-mat/0312317/`, 2003.

[254] L. Wiskott and T. Sejnowski. Slow Feature Analysis: unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.

[255] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 3rd edition, 2005.

[256] C. Wolf, D. Gaida, A. Stuhlsatz, T. Ludwig, S. McLoone, and M. Bongards. Predicting organic acid concentration from UV/vis spectrometry measurements — A comparison of machine learning techniques. *Transactions of the Institute of Measurement and Control*, 19:1–11, 2011.

[257] C. Wolf, D. Gaida, A. Stuhlsatz, S. McLoone, and M. Bongards. Organic acid prediction in biogas plants using UV/vis spectroscopic online-measurements. *Life System Modeling and Intelligent Computing*, pages 200–206, 2010.

[258] M. Zaefferer, B. Naujoks, T. Bartz-Beielstein, M. Emmerich, and T. Wagner. A case study on multi-criteria optimization of an event detection software under limited budgets. In *Proceedings of the Conference on Evolutionary Multi-Criterion Optimization (EMO)*, 2013.

[259] Q. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[260] Q. Zhang, W. Liu, E. Tsang, and B. Virginas. Expensive multiobjective optimization by MOEA/D with Gaussian process model. *IEEE Transactions on Evolutionary Computation*, 14(3):456–474, 2010.

[261] D. Zhao and D. Xue. A comparative study of metamodeling methods considering sample quality merits. *Structural and Multidisciplinary Optimization*, 42(6):923–938, 2010.

[262] E. Zitzler, L. Thiele, and J. Bader. On set-based multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 14(1):58–79, 2010.

[263] W. Zucchini. An introduction to model selection. *Mathematical Psychology*, 44(1):41–61, 2000.

Appendix A

Analysis of noisy multi-criteria optimization

In this chapter we analyze the influence of noisy evaluations in multi-criteria optimization. Here, we focus on the approximation of the Pareto set, when the real Pareto front is known and noise is artificially added to the solutions.

A.1 Estimating noise in multi-criteria machine learning experiments

In the multi-criteria tuning in Chapter 7 we observed, that very high classification accuracies were obtained in some runs. While this seems to be a good result at first, it is obvious, that these solutions can be biased due to some lucky evaluations. Possible reasons for this can be beneficial training and test set splittings, while the real generalization error of such solutions might be worse. This means, that the parameter settings need not necessarily perform equally well on other sub-samples of the data. Unfortunately this drawback can not be prevented without making restrictions elsewhere, e.g., repeated evaluations leading to inferior exploration. In fact each solution is uncertain, but is always regarded as a real observation, which is a necessary requirement for the expected improvement criterion. It is important to know, that the solutions will not be discarded by the optimization procedure, unless they are dominated by other solutions. During the optimization run using the re-interpolation procedure by Forrester *et al.* [82], a solution is evaluated only once on a holdout set, which is randomly sampled from a larger test set. A simple solution for avoiding biases would be to perform repeated evaluations in order to stabilize the tuning error. But as we showed earlier, repeated evaluations also lead to less exploration of the search space, since the budget is limited. As a result of Jin and Branke [129] N repeated evaluations reduce the noise by a factor of $\sqrt{N}$, but at the same time increase the total budget by the factor N. For this reason we do not recommend this as a real alternative.

A frequently used option to alleviate biased solutions is to consider alternative

performance measures, like cross-validation or leave-one-out CV. An overview about resampling strategies has been given in [20]. These resampling strategies yield a good estimation of the real generalization performance. Unfortunately most measures are computationally expensive and quickly become infeasible for large-scale data. For this reason we propose to estimate the generalization performance with SVM in the following section, followed by an overview about strategies for noise handling in multi-criteria optimization.

A.1.1 Related work

In this section we give a short overview about existing approaches measuring the generalization performance of SVMs and multi-criteria optimization in the presence of noise.

A.1.1.1 Estimating the generalization performance with SVM

There exists a large body of work for analyzing the generalization performance of SVM according to bias and parameter settings: Cherkassky and Ma [41] proposed an analytical method to select hyperparameters C and ϵ without direct resampling, but they did not incorporate a systematic tuning of the kernel parameter γ. We think that for this reason the approach is very limited and rather suited for simpler learning tasks. The approach only aims at setting SVM specific parameters, but it can not be extended for additional learning parameters. In our experiments we always obtained a very high sensitivity with regard to parameter γ, and it was also shown that the performance of heuristical γ values is not competitive with tuned γ values. Joachims [131] recommends a bound for the leave-one-out error of SVM which is more efficient to calculate. This is promising, since the leave-one-out error is a good estimator for the unbiased generalization error. Although Joachims reports a very strong correlation with the leave-one-out error, the method was only tested on text classification problems. Therefore it remains unclear, if the method also works for other classification and regression tasks. Valentini and Dietterich [243] analyzed bias and variance of the error to gain insights into SVM learning. They discovered patterns of bias and variance related to different parameter settings and kernel functions of SVMs. However, they did not incorporate noisy data in their experimental study, which makes it difficult to transfer the effects to real situations.

A.1.1.2 Noisy multi-criteria optimization

Some research has been undertaken for coping with noise directly inside the optimization procedure. Comprehensive overviews about noise in multi-criteria optimization have been given by Goh and Tan [96] and Eskandari and Geiger [71]. In more detail, the following approaches have been proposed to handle noise in multi-criteria optimization problems.

A theoretical foundation for noisy multi-criteria optimization has been advocated by Hughes [120]. Hughes discovered, that the probability for finding a non-dominated solution

can be analytically quantified, when the noise is normally distributed and the variance of the noise is known. For two solutions a and b the probability that b dominates a can be calculated as follows:

$$p(b \prec a) = \frac{1}{\sqrt{2\pi(s^2+1)}} \int_0^\infty e^{-\frac{(x-m)^2}{2(s^2+1)}} dx \tag{A.1}$$

$$= \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{m}{\sqrt{2+2s^2}}\right) \tag{A.2}$$

where erf is the error function (without loss of generality the probability density function can be considered therefore). Variables m and s are defined as follows:

$$m = (\mu_a - \mu_b)/\sigma_b,$$
$$s = \sigma_a/\sigma_b$$

where μ_a and μ_b are the mean fitness and σ_a and σ_b the variance of solutions a and b respectively.

Hughes finally ranked the solutions according to the probability given by Eq. A.2. Independently, Teich [238] proposed a similar concept termed *probabilistic dominance*. For these works Fieldsend and Everson [75] advocated a Bayesian algorithm for learning the noise variance.

Bui *et al.* [34] compared the approaches of Hughes and Teich using the popular NSGA-II [57] algorithm with a strategy termed *fitness inheritance*. Here, the fitness of solutions is assigned by the mean fitness of the parents, and if the fitness of the offspring does not satisfy a certain confidence level, the fitness of the offspring is re-evaluated a number of times. In an empirical study Buy *et al.* [34] discovered, that the probabilistic dominance concept is inferior to the re-evaluation strategy. As its main weakness, the probabilistic dominance concepts suffers from a deteriorated diversity of the solution set. As another disadvantage, also high computational costs occur, caused by the evaluation of the integral formula.

For this reason Syberfeldt *et al.* [237] advocate an adaptive re-evaluation approach where the number of replicates are adapted according to the amount of noise. In an experimental study they compare the probability dominance concept of Teich [238] and Hughes [120] with their re-evaluation strategy. Similarly Bui *et al.* [34] they come to the conclusion, that the probabilistic dominance concept performs worse compared with re-evaluation. However, they also indicate that the re-evaluation of inferior solutions is unsatisfactory. As a solution to this problem they propose multi-criteria OCBA [163] for defining the number of repeats. Unfortunately multi-criteria OCBA was not incorporated in their experimental study, where a default setting of 2 evaluations for each design point was stated. In contrast to our approach their multi-criteria optimization algorithm is model-assisted by Artificial Neural Networks, but does not allow for internal noise estimation like Gaussian processes or Kriging.

A.1.2 Model-assisted noise handling

While the previously mentioned approaches all aim at estimating the noise by re-evaluations or additional considerations inside the learning algorithm, it is more convenient, to estimate the noise by the surrogate model itself. Kriging surrogate models have a built-in mechanism for estimating uncertainties, that can be ideally used for the selection of new infill points. We investigate in this section, if this prediction is sufficient, that is wrong decisions of the optimization procedure are avoided.

In the previous experiments the RI by Forrester *et al.* [82] revealed the best performance. In RI at first a non-interpolating Kriging model is built, which assumes a variance greater than zero for all predictions. Note, that all required values are given by the surrogate model: $y_{real}(\vec{x})$ is the observed value of a parameter setting $\vec{x}$, while $\hat{\sigma}(\vec{x})$ is the corresponding variance.

It is possible that Kriging fails in correctly predicting the variances $\hat{\sigma}$. It is also possible that this failure is not revealed, because the real variance always remains unknown. Only in some cases wrong variances can be detected by the users of ML. E.g., an estimated variance of $\hat{\sigma} = 0.1$ for a gain value of $y_{real} = 0.98$ prediction accuracy is infeasible for one side (yielding $0.98 + 0.10 = 1.08$). It is obvious, that the prediction accuracy can never be larger than 1, which equals 100% correctly predicted patterns. And even this value would be at least questionable as generalization performance in ML. An alternative to avoid such irregular variances is to incorporate other variance estimators inside Kriging, e.g., by using conditional distributions, like a truncated normal distribution. We think that this is a matter of further research, and rather concentrate on analyzing the quality of the variance estimation.

A.1.2.1 Research questions

The research questions in this section are defined as follows:

Q1 How many solutions are selected in the final non-dominated set because of *beneficial noise*, that is noise which improves the solutions artificially?

Q2 How large is the distance in the solution space between the noisy solutions in the non-dominated set and the same solutions without noise?

Q3 When re-evaluating the non-dominated points on the objective function without noise, are the solutions Pareto-optimal, or can they be dominated by other solutions?

A.1.2.2 Experimental analysis

In an experimental study we analyze the influence of noisy evaluations for SMS-EGO. Up to now we experimented on tuning ML processes. If reasonable training samples are selected in ML, the observed error should fluctuate around the true generalization error.

A good estimator of the true generalization error is the leave-one-out error. However, it is not possible to define a *true* Pareto front for ML processes, because of the prevailing uncertainties caused by the resampling of data. For this reason we analyze the behaviour of SMS-EGO on the deterministic test function ZDT2, instead of taking into account real ML data. For ZDT2, the Pareto front can be analytically calculated, which will help us in our analysis.

A.1.2.3 ZDT2 test function

The two-criteria function ZDT2 is defined as follows [45]:

$$f_1 = x_1 \tag{A.3}$$

$$f_2 = g(\vec{x}) \cdot [1.0 - (x_1/g(\vec{x}))^2] \tag{A.4}$$

$$g(\vec{x}) = 1 + \tfrac{9}{n-1} \left(\textstyle\sum_{i=2}^{n} x_i\right) \tag{A.5}$$

subject to:

$$0 \leq x_i \leq 1, i = 1, ..., n \tag{A.6}$$

The shape of the Pareto front is non-convex. The Pareto front itself can be analytically calculated by

$$x_1 = x_1 \tag{A.7}$$

$$x_2 = 1 - x_1^2 \tag{A.8}$$

This enables us to compare the attainment surface of SMS-EGO and the true Pareto front of ZDT2.

To make the experiment more comparable to ML experiments, we added normally distributed noise to the function values:

$$y = y + \sigma N[0, 1] \tag{A.9}$$

For ZDT2 we set a value of $\sigma = 0.1$ for both objectives f_1 and f_2. This adaptation of the standard deviation was incorporated, because in ML comparable noise levels have been observed in preliminary runs.

We ran SMS-EGO on the ZDT2 function with normally distributed additive noise, and saved the parameter settings, as well as the objective function values obtained during the runs. For SMS-EGO we defined the following settings: We limited the number of function evaluations by defining a maximal budget of 150 function evaluations, where 20 evaluations were spent for the initial LHS (initial design). The input space X was set to three input dimensions. With these settings it should be possible to attain the Pareto front. However, our main interest is not the solution quality of SMS-EGO, but whether the algorithm can model the noise correctly and does not retain.

A.1.2.4 Results

The approximation sets of the noisy ZDT2 and the ZDT2 function without noise are shown in Fig. A.1. The plot shows that the Pareto front of ZDT2 without noise could be well approximated by SMS-EGO (gray points). This was intended, to expose the effects of the approximation of the algorithm for the noisy ZDT2 function. The red dots picture the non-dominated solutions of ten independent runs, when Gaussian additive noise was added for ZDT2. It can be seen from the plot, that almost all red dots lie beyond the real ZDT2 Pareto front (in the case without noise). This is now the same effect as observed in some ML tuning runs, where parameter settings have been evaluated and the evaluation of the prediction model was too optimistic.

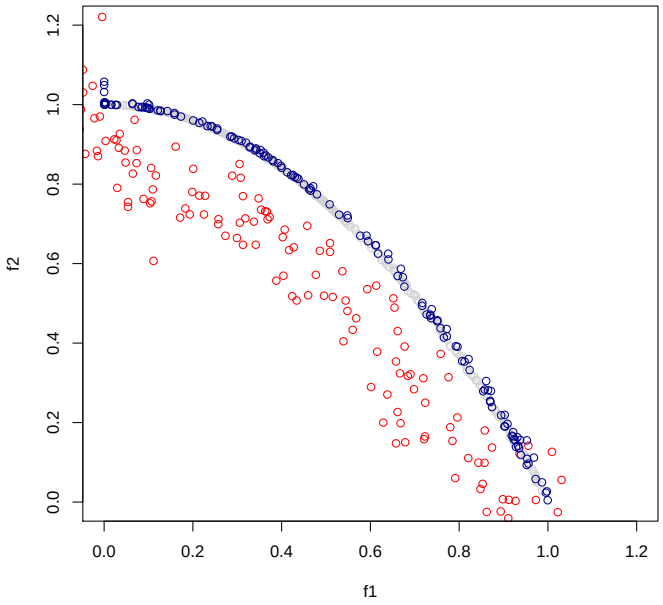


Figure A.1: Approximation of ZDT2. The plot shows the approximated points for the first and second objective on the x- and y-axis respectively. The gray points show the non-dominated solutions obtained after a non-dominated filtering for the solutions of ten runs on ZDT2 without noise. The budget for each run was 150 evaluations with an initial design size of 20. Instead the red points show the non-dominated solutions of ten runs on the noisy ZDT2. They seem to dominate the gray solution set, because (possibly negative) Gaussian noise was added to the solutions. The blue points then show the corrected solutions, when the noise is subtracted again.

Summarizing all ten runs, we obtain the non-dominated solutions as shown in Fig. A.2. Therefore we applied a non-dominated filter for all solutions of the ten runs with SMS-EGO. Again, the points shown in red depict the (now aggregated) non-dominated solution set for

ZDT2 with additive noise, while the blue points depict the corresponding function values without this noise. It can be seen in the plot that the blue solutions are again distributed over the whole Pareto front. Also "corner" solutions, that are solutions when optimizing one single objective of the multi-criteria problem, have been obtained and remain non-dominated.

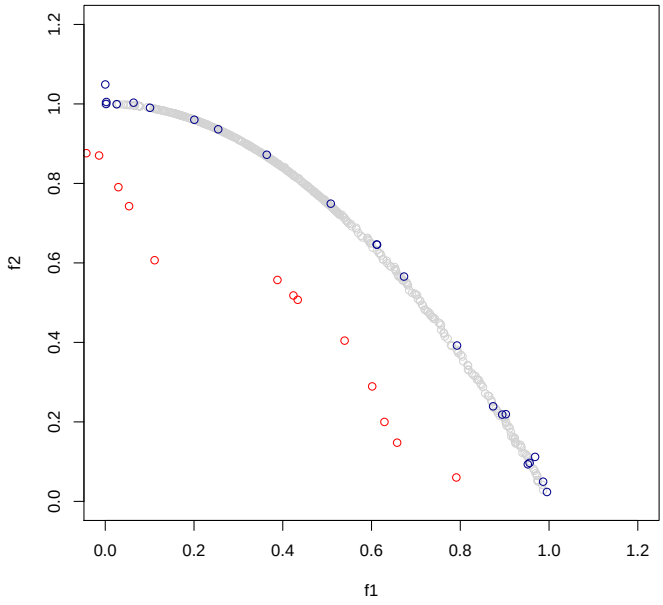


Figure A.2: Same plot for ZDT2 as in Fig. A.1 but now with a non-dominated filtering applied to the aggregated solutions of ten runs.

A.1.2.5 Conclusion

We can conclude that SMS-EGO can deal with misleading noise. Of course the algorithm selects noisy solutions, which appear to be better because of the additive noise term. The solutions with additive noise are improved by the preset standard deviation of the additive normal distribution $N(0, 1)$, which was $\sigma = 0.1$ in our experiments. This is a direct answer to research question **Q2**. Surprisingly no wrong decisions could be observed, e.g., it did not occur, that too many solutions were selected which are not close to the Pareto front of the non-noisy function. Instead, the obtained solution set is well distributed over the Pareto front. We can state, that the main objectives of multi-criteria optimization are respected with additional noise. Our experiment shows, that although noisy solutions are taken in the solution set of SMS-EGO, the solutions are nevertheless close to the real Pareto front after substracting the noise. As a consequence the question deployed in **Q1** can be answered

positively, stating that although the solutions are influenced by the additive noise term, most solutions are still Pareto optimal when the noise is neglected. For this reason we can also affirmate research question **Q3** which can be seen as a good result for the noise handling inside model-assisted optimization using Kriging surrogate models.

Samenvatting (Dutch)

Het afstemmen van parameters van algoritmes voor machinaal leren wordt de laatste jaren in toenemende mate belangrijk. Met de sterke groei van de beschikbare rekenkracht en opslagruimte zijn databases drastisch in omvang toegenomen. Dit vormt een grote uitdaging voor het afstemmen van de parameterinstellingen van leeralgoritmes, aangezien het trainen van machinaal leren erg intensief is voor grote datasets. Dit roept om efficiëntere methoden voor het afstemmen van de parameters, waarmee de leeralgoritmes in staat zijn betere voorspellingen te doen. In dit proefschrift gebruiken we surrogaat-modelgebaseerde optimalisatie technieken voor het afstemmen van de parameters, voor de efficiënte optimalisatie van datasets onderhevig aan ruis en gebruikmakend van een zeer beperkte hoeveelheid aan daadwerkelijke evaluaties van de gekozen parameter-instellingen.

Binnen dit raamwerk combineren we leeralgoritmes met methoden voor kenmerk-constructie en selectie. We onderzoeken de integratie van verschillende elementen in het leerproces, bijvoorbeeld:

- Levert het afstemmen van parameterinstellingen winst op voor leertaken als regressie van tijdsreeksen, gebruikmakend van de modernste leeralgoritmes?
- Zijn statistische methoden in staat tot het reduceren van de effecten van ruis?
- Kunnen surrogaatmodellen zoals Kriging worden gebruikt om een redelijke benadering van de afbeelding van het parameterlandschap naar kwaliteitsmaten te verkrijgen?

Samengevat, we analyseren of betere leeralgoritmes kunnen worden samengesteld, met de nadruk op een efficintere doorlooptijd.

We besteden, naast aan de voordelen van de systematische aanpak van het afstemmen van parameterwaarden, aandacht aan de moeilijkheden en mogelijke obstakels binnen het systematisch afstemmen. Verschillende methodes voor het afstemmen worden vergeleken en de invloed van hun specifieke eigenschappen worden zichtbaar gemaakt. Dit onderzoek heeft als doel inzicht te verschaffen in de wijze waarop de meest recente leeralgoritmes in de praktijk op bruikbare wijze toegepast kunnen worden.

Curriculum Vitae

Patrick Koch was born in Lippstadt, Germany on January 18th, 1982. After the high school degree he studied Computer Science in Paderborn, Germany. He graduated with the diploma in 2008, and started working as a research associate at the Cologne University of Applied Sciences in 2009. In the same year he joined the Natural Computing Group of Prof. Thomas Bäck at the Leiden Institute for Advanced Computer Science (LIACS). Patrick's research interests include but are not limited to Computational Intelligence, machine learning, multi-criteria optimization and statistics.