Evol. Intel. (2012) 5:153–170
DOI 10.1007/s12065-012-0073-8

SPECIAL ISSUE

Tuning and evolution of support vector kernels

Patrick Koch · Bernd Bischl · Oliver Flasch · Thomas Bartz-Beielstein · Claus Weihs · Wolfgang Konen

Received: 10 March 2011 / Accepted: 17 April 2012 / Published online: 4 May 2012

Abstract Kernel-based methods like Support Vector Machines (SVM) have been established as powerful techniques in machine learning. The idea of SVM is to perform a mapping from the input space to a higher-dimensional feature space using a kernel function, so that a linear learning algorithm can be employed. However, the burden of choosing the appropriate kernel function is usually left to the user. It can easily be shown that the accuracy of the learned model highly depends on the chosen kernel function and its parameters, especially for complex tasks. In order to obtain a good classification or regression model, an appropriate kernel function in combination with optimized pre- and post-processed data must be used. To circumvent these obstacles, we present two solutions for optimizing kernel functions: (a) automated hyperparameter tuning of kernel functions combined with an optimization of pre- and post-processing options by Sequential Parameter Optimization (SPO) and (b) evolving new kernel functions by Genetic Programming (GP). We review modern techniques for both approaches, comparing their different strengths and weaknesses. We apply tuning to SVM kernels for both regression and classification. Automatic hyperparameter tuning of standard kernels and pre- and post-processing options always yielded to systems with excellent prediction accuracy on the considered problems. Especially SPO-tuned kernels lead to much better results than all other tested tuning approaches. Regarding GP-based kernel evolution, our method rediscovered multiple standard kernels, but no significant improvements over standard kernels were obtained.

Keywords Machine learning · Parameter tuning · Support vector machines · Genetic programming · Kriging

First, second and third author contributed equally

P. Koch (✉) · O. Flasch · T. Bartz-Beielstein · W. Konen
Institute of Computer Science, Cologne University of Applied Sciences, Cologne, Germany
e-mail: patrick.koch@fh-koeln.de

O. Flasch
e-mail: oliver.flasch@fh-koeln.de

T. Bartz-Beielstein
e-mail: thomas.bartz-beielstein@fh-koeln.de

W. Konen
e-mail: wolfgang.konen@fh-koeln.de

B. Bischl · C. Weihs
Faculty of Statistics, University of Dortmund, Dortmund, Germany
e-mail: bischl@statistik.tu-dortmund.de

C. Weihs
e-mail: weihs@statistik.tu-dortmund.de

1 Introduction

Kernel-based learning methods are state-of-the-art techniques in supervised machine learning. The kernel trick makes it possible to perform a transformation from the input data space to a higher-dimensional feature space, where the transformed data can be described by linear models and the problem becomes tractable. However, the result highly depends on the considered transformation. If the kernel function is not appropriate for the problem, or kernel parameters are badly set, the fitted model can be of poor quality. Due to this, special care must be taken in selecting both the kernel function and kernel parameters to obtain good results.

We propose a solution to this by using a statistical optimization tool and a current genetic programming framework to tune and evolve kernels of Support Vector Machines.

We define the following hypotheses:

H1 In complex classification and regression tasks good parameter settings for standard kernels and pre- and post-processing options can be found almost automatically with the right statistical optimization tool (SPO).

H2 Genetic Programming (GP) can be used to find data-optimized kernel functions for machine learning problems. These kernel functions may show better performance (for some problems) than tuned standard kernels.

This paper is organized as follows: Support Vector Machines, Sequential Parameter Optimization and Genetic Programming are introduced in Sects. 2.1.1–2.1.3 respectively. Section 2.2 gives a short review of recent kernel tuning and evolution approaches. In Sect. 2.3 our general concept of tuning and evolution of SVM kernels are presented, and we point out possible potential for further kernel tuning. In Sect. 2.4 we mention our own software for these purposes. We give a detailed experimental study on tuned SVM kernels in Sect. 3. Here, we use GP-evolved and SPO-tuned kernels to build optimized models for data mining problems.

2 Approach

This section explains the different approaches to SVM tuning studied in this work. After a short introduction to basic methods, including SVM, SPO, and GP, a review of existing tuning strategies is given. We then continue to describe our approaches to tuning standard kernels and evolving custom SVM kernels for specific applications. This section closes with a definition of the performance measures used in this work.

2.1 Methods

2.1.1 Support vector machines

Support Vector Machines have been proposed as a supervised learning algorithm for both classification and regression. Since the early nineties the field of kernel-based learning algorithms has developed very quickly and many extensions and comparable methods emerged.

In supervised machine learning data can be represented as a number of observations

$$(\mathbf{X}_1, Y_1), (\mathbf{X}_2, Y_2), \ldots, (\mathbf{X}_m, Y_m) \in \mathcal{X} \times \mathcal{Y} \tag{1}$$

where the set $\mathcal{X}$ defines the inputs and $\mathcal{Y}$ are the targets, e.g., real values in regression or class labels for classification. For classification we will consider only the binary case in this section where $\mathcal{Y} = \{-1, +1\}$. One basic assumption in machine learning is that two observations "being near in input space" should have a similar target value. Out of this reason we can define a function

$$k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \tag{2}$$

denoting the similarity of two observations. k is a symmetric, positive semi-definite kernel function, which can be interpreted as a dot product in a high-dimensional space [37]. It implicitly transforms the data into this space and by enlarging the dimension enables us to tackle nonlinear problems with essentially linear techniques.

Making this more rigorous we can define H as the associated reproducing kernel Hilbert space for k and define the optimal model as the solution to the following optimization problem

$$\hat{f} = \arg \inf_{f \in H, b \in \mathbb{R}} ||f||_H^2 + C \sum_{i=1}^{m} L(Y_i, f(\mathbf{X}_i) + b). \tag{3}$$

(Here, f maps into $\mathbb{R}$ even in the case of binary classification and in order to get discrete predictions we would calculate its sign.) The second term measures the closeness of our predictions to the true targets by means of a loss function, while the first term $||f||_H^2$ is called a penalty and in case of the 2-norm penalizes non-smooth functions. The balance between the loss and the smoothness penalty is controlled by the hyperparameter C.

For classification we usually select the hinge loss $L(Y, t) = L_h(Y, t) = \max(0, 1 - Yt)$, while 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 $\mathbf{X}$. Both losses lead to quadratic programming problems for (3), which can be solved efficiently, and the non-differentiability of these two loss functions further provides for sparse solutions [9].

In time series regression for a time series $x(t)$, where $t \in \mathbb{R}$ is time, we define a state vector $\mathbf{X}_t = (x(t), x(t - \tau), \ldots, x(t - (d - 1)\tau))$ with time delay τ and embedding dimension d. We are interested in predicting a point in the future with time horizon p by using the past values encoded in the state vector: $Y_t = x(t + p) = f(\mathbf{X}_t)$. Support vector regression (or any other regression technique) can now be used to model and estimate f [39]. Note, that it is straightforward to extend the approach above for multivariate time series. See Drucker et al. [15] for examples of applications.

2.1.2 Sequential parameter optimization

Sequential Parameter Optimization (SPO) is a framework for improving and understanding the behavior of general algorithms by experimentation. Often these algorithms are evolutionary search methods but SPO can also be applied to machine learning methods. Note, that in the main part of this section we will introduce the general concept of SPO which encompasses all these scenarios.

First experiments using SPO were devoted to the analysis of stochastic search algorithms, namely Evolution Strategies (ES) and Simulated Annealing (SA) [5]. SPO combines methods from classical Design of Experiments (DoE) and modern Design and Analysis of Computer Experiments (DACE).

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.

Kriging was developed in geostatistics as a tool for curve fitting and response surface approximation, while DACE was introduced in the 1980s for deterministic computer generated data [44, 45]. The latter is based on the Kriging approach and [6] demonstrated that Kriging can also be applied to tuning stochastic problems.

Both classical and modern methods from statistics are considered for optimizing algorithm parameters to improve algorithm performance. SPO sequentially performs a pre-defined number of algorithm runs, and uses the information during exploration of the search space to build and refine a meta model of the true objective function. This model is used as a cheap surrogate for the true objective function and optimized to produce new design points, which in turn are evaluated by the algorithm and used to update the model. Such a meta model is usually an interpolating or regression model. We will focus on kriging here, as it often performs best in modeling the nonlinear fitness landscapes in computer experiments, but SPO is general enough that it does not require a specific model.

As the algorithm we want to analyze and tune might be stochastic (or the problem instances it is applied to), we need to incorporate a mechanism that deals with noise. In SPO two strategies are used for this purpose: (a) The algorithm is evaluated multiple times at the design points. The number of replications is slowly increased during the sequential optimization and promising design points are evaluated more often than worse ones. (b) Some changes might be necessary for the meta model, if we do not want to interpolate the design points. They include estimating a nugget effect for the Kriging model and the re-interpolation technique by Forrester. These are explained in the following Kriging part.

In Algorithm 1 the pseudo code of SPO is presented. Note, that in this section we will use the notation $x^{(i)}$, $y^{(i)}$ for the data passed to the surrogate model, instead of X_i, Y_i as for the learning data for the SVM. This was done to emphasize the fact that in our applications of tuning SVM-based models the data passed to regression model in SPO are the parameters of the learning machine and not the observations in the training set itself.

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

Different strategies for choosing the number of replications are possible. In order to keep things simple, we present a strategy where in each iteration the currently best design point is evaluated one additional time (until a maximum number of replications is reached) and the newly proposed points are evaluated the same number of times as the current best.

During the first stage of experimentation SPO explores the performance space of the algorithm A, which is treated as a black box. A set of input design points $\mathbf{x}$ is passed to A, usually these are created by a space filling design, e.g. latin hypercube sampling. Each run of the algorithm produces some output y regarding its performance. SPO now tries to determine a functional relationship between $\mathbf{x}$ and y and to sequentially improve it. In the sequential improvement loop SPO optimizes the current model $Y(\mathbf{x})$ over the considered space of input variables by means of a cost function. In the simplest case this cost function is the estimated output itself as this should reflect the performance of A. But more sophisticated criteria are possible to select the next sampling points, and in case of kriging they are called 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.

2.1.2.1 Kriging in SPO For now we make the usual assumption for a general regression setting that the output data are subject to simulation-model errors, i.e., we have noisy measurements y_i at the i-th data point $\mathbf{x}^{(i)}$, where ϵ_i is the measurement noise.

The SPO approach consists of two steps: (1) Model construction and (2) optimizing the model. We will only present these for kriging here, where both steps are based on the maximum likelihood estimation (MLE) approach presented in [18, 27].

(1) Model construction is implemented as follows: Consider a dataset $\{(\mathbf{x}^{(i)}, y^{(i)})\}, (i = 1, \ldots, n)$. The observed responses $\mathbf{y} = \{y^{(i)}\}$ are considered as if they were from a Gaussian process, i.e., we will use a set of random vectors $\mathbf{Y} = \left(Y(\mathbf{x}^{(1)}), \ldots, Y(\mathbf{x}^{(n)})\right)^T$ with associated $(n \times n)$ correlation matrix

$$\boldsymbol{\Psi} = \left(\mathrm{cor}(Y(\mathbf{x}^{(i)}), Y(\mathbf{x}^{(j)}))\right)_{i=1,j=1}^{n} \tag{4}$$

and correlation function

$$\mathrm{cor}(Y(\mathbf{x}^{(i)}, Y(\mathbf{x}^{(l)}) = \exp\left(-\sum_{j=1}^{k} \theta_j (x_j^{(i)} - x_j^{(l)})^2\right). \tag{5}$$

Under standard assumptions, cf. [18], the likelihood is

$$L(\mathbf{Y}^{(1)}, \ldots, \mathbf{Y}^{(n)} | \mu, \sigma) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{\sum(\mathbf{Y}^{(i)} - \mu)^2}{2\sigma^2}\right). \tag{6}$$

In order to filter noise, a regression constant λ was added to the leading diagonal of $\boldsymbol{\Psi}$, also sometimes called a nugget effect. Expressing Eq. 6 in terms of the sample data, taking derivatives, and setting to zero, we obtain the estimates

$$\hat{\mu} = \frac{\mathbf{1}^T (\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1} \mathbf{y}}{\mathbf{1}^T (\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1} \mathbf{1}}, \tag{7}$$

$$\hat{\sigma}^2 = \frac{(\mathbf{y} - \mathbf{1}\hat{\mu})^T (\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1} (\mathbf{y} - \mathbf{1}\hat{\mu})}{n}, \tag{8}$$

and the *concentrated ln-likelihood function*

$$\ln(L) \approx -\frac{n}{2} \ln(\hat{\sigma}^2) - \frac{1}{2} \ln \det(\boldsymbol{\Psi} + \lambda \mathbf{I}). \tag{9}$$

We can now determine the unknown parameters, i.e., the vector $\boldsymbol{\theta}$ introduced in Eq. 5 and the regression constant λ by maximizing the concentrated ln-likelihood function.

(2) Now that we have constructed the model, we can perform the second step, i.e., optimizing it.

A prediction at $\mathbf{x}$ is given by

$$\hat{y}(\mathbf{x}) = \hat{\mu} + \boldsymbol{\psi}(\mathbf{x})^T (\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1} (\mathbf{y} - \mathbf{1}\hat{\mu}), \tag{10}$$

with $\hat{\mu}$ as defined in Eq. 7 and $\boldsymbol{\psi}(\mathbf{x})$ the vector of correlations between the observed data and a new prediction, i.e.,

$$\boldsymbol{\psi}((\mathbf{x})) = \left(\mathrm{cor}(Y(\mathbf{x}^{(1)}), Y(\mathbf{x})), \ldots, \mathrm{cor}(Y(\mathbf{x}^{(n)}), Y(\mathbf{x}))\right)^T.$$

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

$$\hat{s}^2(\mathbf{x}) = \hat{\sigma}^2 \Bigg(1 - \boldsymbol{\psi}(\mathbf{x})^T (\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1} \boldsymbol{\psi}(\mathbf{x}) + \frac{1 - \boldsymbol{\psi}(\mathbf{x})^T (\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1} \boldsymbol{\psi}(\mathbf{x})}{\mathbf{1}^T (\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1} \mathbf{1}}\Bigg), \tag{11}$$

with $\hat{\sigma}$ as defined in Eq. 8. Because of the inclusion of the nugget effect λ Eq. 11 does not reduce to zero when we calculate it at an already evaluated sample point.

When determining the next point $\mathbf{x}$ to evaluate in the sequential loop of SPO, the kriging model is optimized with respect to a so-called infill criterion. This measure defines the expected cost of $\mathbf{x}$ for the black-box optimization problem under consideration. The expected improvement is the one most widely used. Its mean idea is to visit points which should either exhibit a low objective function value or a high model uncertainty. This ensures a balanced exploration and exploitation of the search space. Other infill criteria are discussed in [46].

First, we define the random variable $I(x)$, which denotes the improvement compared to the so far visited point $\bar{\mathbf{x}}$ with objective value $\bar{y}$ when we evaluate the process at a new point $\mathbf{x}$:

$$I(\mathbf{x}) = \max(\bar{y} - Y(\mathbf{x}), 0) \tag{12}$$

Finally, Eqs. 10 and 11 are combined to calculate the expected improvement

$$E(I(\mathbf{x})) = (\bar{y} - \hat{y}(\mathbf{x}))\Phi\left(\frac{\bar{y} - \hat{y}(\mathbf{x})}{\hat{s}(\mathbf{x})}\right) + \hat{s}(\mathbf{x})\phi\left(\frac{\bar{y} - \hat{y}(\mathbf{x})}{\hat{s}(\mathbf{x})}\right), \tag{13}$$

where $\bar{y}$ is the best observed value so far and $\Phi(\cdot)$ and $\phi(\cdot)$ 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 $\lambda = 0$, but see also the comments in the following paragraph about re-interpolation. For the expected improvement the gradient can be calculated analytically, but as it is a multi-modal function it is usually maximized with an evolutionary algorithm or by employing restart strategies.

2.1.2.2 Modifications for non-deterministic problems

The original version of the presented algorithm required deterministic data, i.e., evaluating one design point multiple time would generate the same response. Thus, it could not be applied directly to non-deterministic data.

In principle, there are two approaches to cope with non-deterministic output data:

1. *Repeated measurements*. This approach, estimates y at $\mathbf{x}$ by the mean of repeated measurements and this value is fed to the Kriging model. This was already covered in algorithm 1.
2. *Reinterpolation by Forrester*. In this approach, the kriging model is able to handle the raw, noisy data. The noise can simply be handled in the Kriging model by fitting it with a so-called nugget effect (already explained above as well), but the expected improvement calculation has to be modified. A simple solution through reinterpolation is offered by [18], and will be referred to as *Forrester* later on. Here the Kriging model is fitted twice. First with nugget-effect due to the noisy observations, then the response values $y^{(i)}$ of the data are substituted by the predictions of the regression model. Now the data is interpolated in a second step by a Kriging model without nugget effect, for which the expected improvement can be calculated without modification. 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.

2.1.2.3 Comparing kriging and SVM Note, that there is a close connection between Kriging (Gaussian processes) and SVMs 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 fully stochastic model or interpretation for the SVM is currently known. Some work has been done by Sollich in that regard [48], but his interpretation does not seem to be universally accepted by the community (Rasmussen and Williams call his construction „rather contrived” in [42]). A full and proper discussion of this connection is out of scope for the current paper and the reader is referred to [17, 40, 42].

2.1.3 Genetic programming

Genetic programming (GP) is a collection of techniques from evolutionary computing (EC) for the automatic generation of symbolic expressions for solving a user-defined task [3, 35, 41]. 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.

An important advantage of GP is that no prior knowledge concerning the solution structure is needed. Tasks are defined by fitness functions associating candidate solutions with numerical fitness values encoding solution quality. Another 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.

Before 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.

Building blocks A set of *building blocks* consisting 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. Because we want to bias our search to simple individuals, instead of employing the classical ramped half and half heuristic, we employ a strategy that grows random individuals to a random tree depth less or equal than a maximum tree depth given as a parameter [35].

Variation operators A set of *variation operators* 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 here in a type-safe manner. [35, 41]

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.

The data and control flow of the GP algorithm used in this work is shown in Fig. 1.

2.2 Review of tuning approaches

Many different optimization techniques have already been tested to set kernel hyperparameters. For derivative-free search these include among others:

(a) simple grid search, where parameter settings are tested by performing geometric steps inside the boundary of each parameter range. Grid search should only be applied with very few hyperparameters, e.g., two or maximally three, because the exponentially increasing search space quickly makes this method inapplicable.
(b) pattern search, where one locally improves the current point by considering a finite, deterministic neighborhood [38],
(c) the well-known Nelder-Mead simplex strategy [9],
(d) evolutionary strategies like CMA-ES [20],
(e) simulated annealing [1],
(f) expected improvement maximization by Kriging [21]
(g) design-of-experiments sampling techniques [49].

To our knowledge Koch et al. [31] were the first who used SPO as a tuner for optimizing the radial basis kernel function for Support Vector Machines (SVM) combined with preprocessing parameters on a complex real-world data mining task.

Previous work in optimizing kernel functions for SVMs by means of GP has been proposed by Howley and Madden [26]. Compared with standard kernel functions a tree-based GP produces 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 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. [13] also use GP for evolving kernels and respect the PSD property of kernel functions. In their work a larger GP function set compared to [26] is used, but results are only evaluated on few benchmark datasets. Sullivan and Luke [50] 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. [22] use a co-evolutionary approach for the kernel evaluation. Kernels obtained by their GP approach are ranked by a nearest neighbor classification algorithm instead of SVM, since the PSD property can not 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

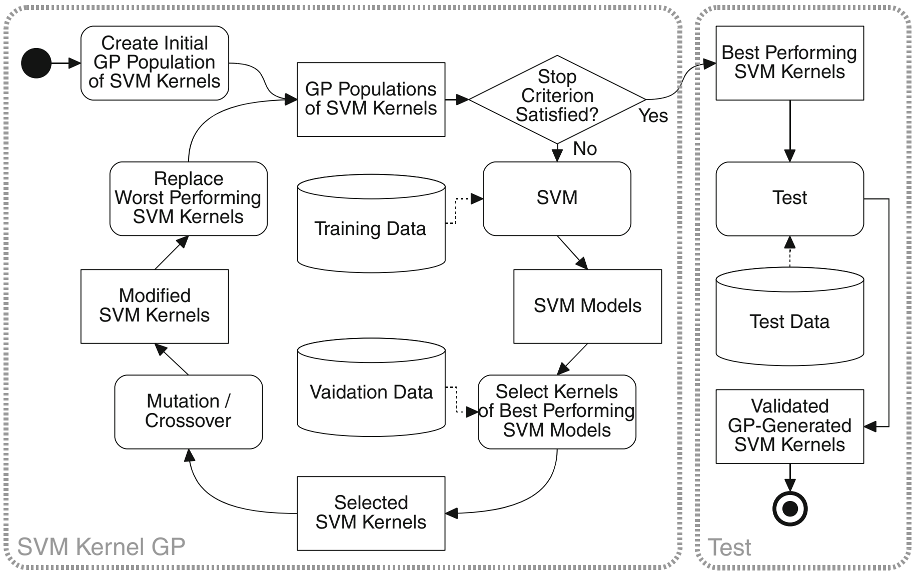
Create Initial GP Population of SVM Kernels
GP Populations of SVM Kernels
Stop Criterion Satisfied?
Yes
No
Best Performing SVM Kernels
Replace Worst Performing SVM Kernels
Training Data
SVM
Test
Modified SVM Kernels
SVM Models
Test Data
Mutation / Crossover
Vaidation Data
Select Kernels of Best Performing SVM Models
Validated GP-Generated SVM Kernels
Selected SVM Kernels
SVM Kernel GP
Test

Fig. 1 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

- the PSD property of the kernel functions,
- the fitness function, e.g. the evaluation of intermediate kernels,
- the runtime of the modeling process. Model training and prediction can last very long, which means that the evaluation of the objective function can be very expensive. The internal runtimes of tuning algorithms like CMA-ES or Kriging are not discussed here, since in our experiments they only constituted a very small part in the optimization process. However, with increasing number of parameters, runtimes of tuning algorithms should be discussed critically.

Also, by either considering a smoothed version of the cross-validation error [29] or a likelihood function of the hyperparameters [23] 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 are currently not aware of a really comprehensive comparison study.

2.3 Kernel tuning and evolution

The performance of Support Vector Machines highly depends on the chosen parameter values for regularization and the kernel parameters. First, we describe how SPO and other search heuristics can be used for parameter tuning of standard kernel functions, pre-processing and post-processing operations. Note, that although the fitting of SVMs is deterministic, their performance prediction is not, see Sect. 2.3.3, when stochastic resampling technique is used. Second, we evaluate whether new, non-standard kernel functions can be found by by means of Genetic Programming and refer to the requirements of the evolved functions, e.g., positive semi-definiteness. We also point out the possibility to tune constants during evolution and after having kernel functions evolved by GP.

2.3.1 Parameter tuning for SVM kernels

In this approach we use the TDM framework [32, 34] to tune parameters of the SVM kernel function $k(\mathbf{X}_i, \mathbf{X}_j)$. Sometimes these parameters are just set by hand, although it can give high improvements when performing an optimization step for them. Kernel parameters usually have a certain range, e.g., real values between 0 and ∞ or 0 and 1. Hence, a finite region of interest (ROI) for the tuning algorithms (SPO, CMA-ES, L-BFGS-B) is either defined or can be obtained (or improved) by preliminary runs. We consider as kernel functions for tuning the popular RBF kernel:

$$k(\mathbf{X}_i, \mathbf{X}_j) = exp(-\gamma||\mathbf{X}_i - \mathbf{X}_j||^2) \quad (14)$$

where γ is a kernel parameter. Other kernels (e.g., linear, polynomial) were initially tested on some real-world datasets but showed worse performance.

In a previous work Koch et al. [30, 31] used SPOT for tuning both SVM parameters and pre- and post-processing parameters. We investigate how parameter tuning influences the quality of a trained SVM model using different approaches for the tuning. In this work the following methods are compared to each other:

- Quasi-Newton search, more specifically the BFGS [8] algorithm was originally designed for unconstrained optimization, although it can be used for constrained optimization problems supporting box-constraints, which is sufficient in our case. Since no analytical gradient is available for our objective functions, we use numerical approximations instead. In our experiments we use an extended version of BFGS called L-BFGS-B. For more details on this method we refer the reader to [8, 55].
- Evolution Strategies, e.g. CMA-ES
- SPOT

For simplicity we define an input-output function $f_{[\mathcal{T},\mathcal{E}]}(p, q) = y$ where p is usually the set of parameters for the SVM. E.g., all parameters affecting the SVM and its kernel function. Other parameters for task-specific pre-processing can be defined in q, and $y \in \mathbb{R}$ is the corresponding objective function value. A SVM is trained on some training data $\mathcal{T}$ using its parameter set $\{p, q\}$, and the model is evaluated on data $\mathcal{E}$.

2.3.2 SVM kernel evolution by genetic programming

Tuning of kernel functions has shown to be useful for obtaining better results for data mining tasks. Up to now, kernel tuning by SPOT is restricted to existing kernel functions. We believe, that this is not the end of the road and that by replacing standard kernel functions by more appropriate ones better results can be obtained. E.g., Cortes et al. [12] have proposed a gradient-descent algorithm for learning polynomial combinations of kernels in regression. We want to further investigate the possibility to use GP for evolving kernel functions and improve upon this technique. GP has become more and more popular with the increase in computational power in the last years, and is now applicable even to complex tasks.

2.3.2.1 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 [37] 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: $k_1(\mathbf{X}_i, \mathbf{X}_j) + k_2(\mathbf{X}_i, \mathbf{X}_j)$
- Closure under product: $k_1(\mathbf{X}_i, \mathbf{X}_j) \cdot k_2(\mathbf{X}_i, \mathbf{X}_j)$
- Closure under positive scalar multiplication: $a \cdot k_1(\mathbf{X}_i, \mathbf{X}_j)$, with $a > 0$
- Closure under exponentiation: $\exp(k_1(\mathbf{X}_i, \mathbf{X}_j))$

For a more detailed description of closure properties we refer to Cortes et al. [10, 11]. However, Schölkopf [47] showed that even when the PSD condition is violated, some non-PSD kernel functions have been used successfully in practice for learning tasks. Probably the most well-known kernel function, which is not strictly PSD, is the sigmoid kernel:

$$k(\mathbf{X}_i, \mathbf{X}_j) = \tanh(\alpha \mathbf{X}_i^T \mathbf{X}_j + C) \tag{15}$$

Vapnik [51] 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.

2.3.2.2 GP search space Table 1 defines the strongly typed GP search space for SVM kernel evolution. The used building blocks (constants, input variables, and operators) allow the definition of all standard support vector kernels presented in Sect. 2.3.1, 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 Table 1 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 $\mathbf{X}_i$ and $\mathbf{X}_j$.
- k must contain at most 4 symbolic constants.
- The kernel matrix M_k of k must not contain numerical problems such as `NaN`s or infinite values.

Table 1 Strongly typed GP search space for SVM kernel evolution

Building block class	Type	Building blocks
Constant	$\mathbb{R}$	$\{c_1, c_2, c_3, c_4\}$
Input variable	$\mathbb{R}^D$	$\{\mathbf{X}_i, \mathbf{X}_j\}$
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	$\mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}^D$	$\{\cdot + \cdot, \cdot - \cdot\}$
	$\mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$	$\{\cdot \times \cdot, \lVert \cdot - \cdot \rVert^2\}$
	$\mathbb{R} \times \mathbb{R}^D \to \mathbb{R}^D$	$\{\cdot \times \cdot\}$
	$\mathbb{R}^D \times \mathbb{R} \to \mathbb{R}^D$	$\{\cdot \times \cdot\}$

All possible building blocks for each building block class are shown. The variables x, x_i and x_j denote real vectors of the SVM input data dimension D, the constants $c_1, \ldots, c_4$ 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. Note that some operators, such as the $\times$ operator, are "overloaded" to work with multiple operand types

- M_k's eigenvalues must be non-negative (up to a small numerical margin), i.e. M_k must be numerically PSD.

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.

Figure 2 introduces the typed GP variation operators used in this study by means of examples. Formal definitions of these standard operators are given in [41]. The parameter settings of the GP system are listed in Table 3.

2.3.2.3 GP fitness function The quality of an individual k, i.e. the quality of the symbolic representation of a support vector kernel, 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, the worst possible 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

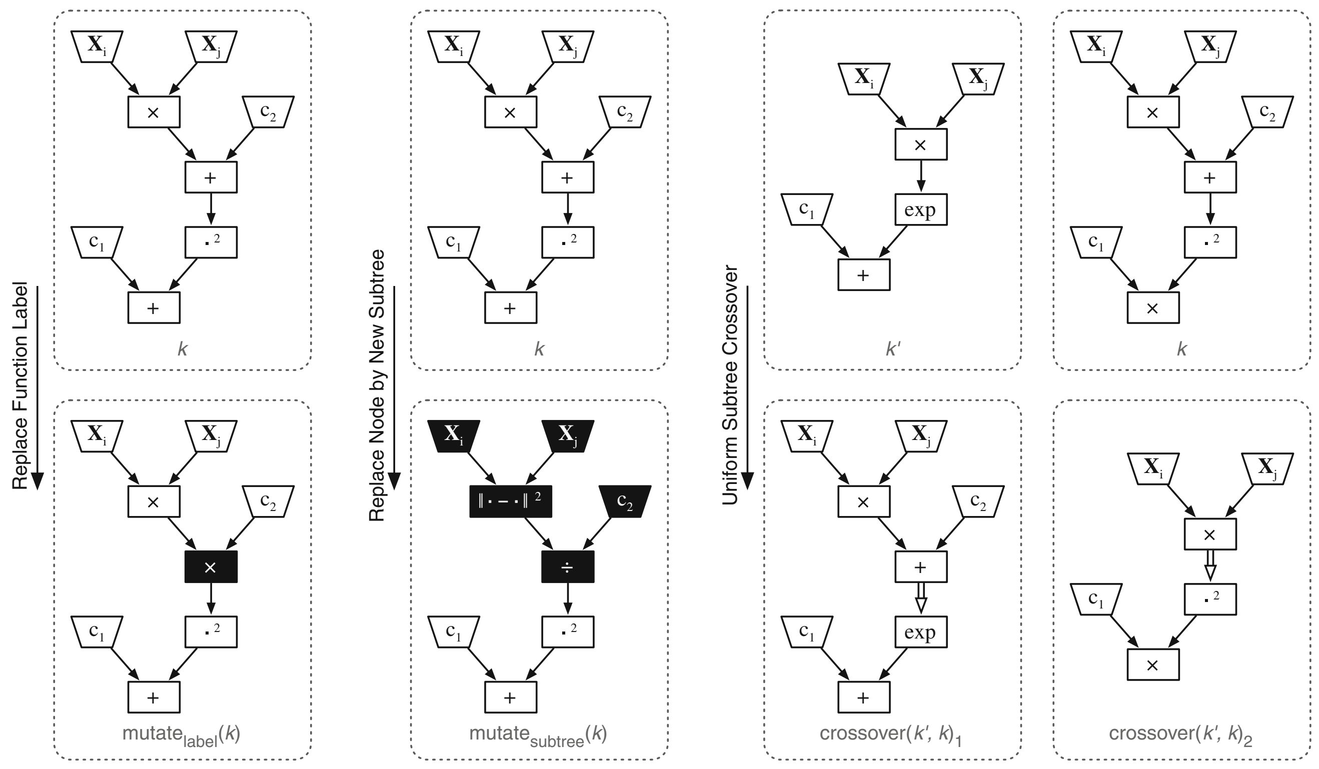

Fig. 2 Standard typed GP variation operators explained by means of example: Two different mutation operators and a single crossover operator are employed. The first mutation operator "Replace Function Label" chooses a node by uniform random selection to be replaced by a node of matching type. Nodes affected 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 in inverse

size $d \cdot (\mathrm{nconst}(k) + 1)$ is created by latin hypercube sampling, where 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 $\mathbf{Y}$ of performance values. Performance is measured by mean misclassification error (MMCE).
4. The pointwise minimum of $\mathbf{Y}$, 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 Sect. 3.2.1 for a definition of ϵ.

The general outline of GP evolution of SVM kernels is depicted in Fig. 1.

2.3.3 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 looking only at a subset of our available data, we evaluate on the remaining observations to avoid optimistic bias. This process is repeated a couple of times in order to maximally utilize the limited amount of data, and the generalization of this concept is called resampling. 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. 3. 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 and should look for less expensive alternatives if possible. 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. [16] for a comparison of possibilities.

2.4 Software

All the experiments presented in this work are based on software solely written with the help of available open source toolboxes. We describe in this section some of these toolboxes, which are central to the ideas presented here.

2.4.1 SPOT

The *Sequential Parameter Optimization Toolbox* SPOT is a software implementation of the Sequential Parameter Optimization framework SPO as described in Sect. 2.1.2. SPOT is open source software and available for both Matlab and R. [4]

2.4.2 TDM

The TDM framework [32, 34] is written in R with the aim to facilitate the setup, training and evaluation of data mining models. It puts special emphasis on tuning these data mining models as well as simultaneously tuning certain pre-processing options. TDM is especially designed to work with SPOT as the preferred tuner, but it offers also the possibility to use other tuners (CMA-ES [24], LHD [36] and local optimizers) for comparison. The goal of TDM can be formulated as follows: Provide a recipe/template for a generic data mining process (classification or regression) which works well on many different data mining tasks. In its current version the TDM framework contains:

1. Sampling, i.e., the division of the data in training and test set (random, k-fold cross validation (CV), …)
2. Generic feature generation and generic feature selection (currently Random-Forest-based variable ranking and GA)
3. Modeling: currently SVM, Random Forest (RF), MC.RF [34], but other models, especially all those available in `R` can easily be integrated
4. Model application: predict class and (optional, depending on model) class probabilities
5. User-defined post-processing (optional)
6. Evaluation of model: confusion matrix, gain matrix, score, generic visualization, …

TDM can optimize the pre-processing and modeling parameters contained in step 2. and 3. by a generic tuning process with one of the above-mentioned tuning algorithms.

2.4.3 MLR

The mlr package [7] provides a generic, object-oriented interface to about 50 machine learning methods in R for classification and regression and can easily be extended with further ones. It enables the researcher to rapidly conduct complex experiments or implement his own meta-methods using building blocks of the package. Resampling like cross-validation, bootstrapping and subsampling are used to assess the generalization performance, measured by e.g. MMCE, MSE, cost-sensitive measures, ROC measures, etc. Custom measures can easily be defined as well.

Learner functionality can be extended by various building blocks like multiclass-to-binary reduction, pre-processing and post-processing steps and optimizers, resulting in complex, tunable data mining systems. Hyperparameters of complex learning systems can be tuned by grid search or more sophisticated deterministic or stochastic search methods like e.g. Nelder-Mead, CMA-ES or SPO. The same holds true for variable selection. Here, various feature selection wrapper approaches (forward search, backward search or genetic algorithms) and fast filter methods are available.

Benchmark experiments with two levels of resampling, e.g. nested cross-validation, can be specified with few lines of code to compare different learning systems. Parallel high-performance computing is supported and experiments can be converted to parallelized versions with a simple configuration command, without touching any further code.

2.4.4 RGP

RGP is an open source genetic programming system based on the R environment. The system implements classical untyped tree-based genetic programming as well as more advanced variants including, for example, strongly typed genetic programming and Pareto genetic programming. It strives for high modularity through a consistent architecture that allows the customization and replacement of every algorithm component, while maintaining accessibility for new users by adhering to the "convention over configuration" principle. RGP's support for strongly typed genetic programming, breeding, and easy customization makes it a good fit for SVM kernel evolution.

3 Experimental study

We compare different variants for kernel tuning when applied to complex real-world problems and also simple benchmark problems to make our approach comparable to other work. More specifically we use the TDM framework for tuning parameters of two real-world problems—the acid concentration problem and the stormwater problem. The benchmark problems were chosen to show that a systematic tuning is especially beneficial, when the problem at hand is rather difficult. We want to point out that with such hard problems it is often difficult to get good results without tuning in contrast to simple benchmark problems. Within TDM the tuning algorithms SPO, CMA-ES and L-BFGS-B were employed.

MLR was used as a framework for combining the SVM algorithm with RGP and a comparison study with tuned RBF kernels. For the kernel evolution we evaluated on simpler benchmark problems from the UCI repository, since GP is computationally expensive and in order to compare our results to already published similar approaches [13, 22, 50]. Nevertheless GP can be also used to find kernels for real-world problems. It should be noted that the difficulty in finding good solutions for real-world problems is considerably higher than the complexity of most standard benchmark problems. Most complex real-world problems as time series regression or classification problems contain more than 30,000 records, which is much more data than usually used in benchmarks and their predictability can be poor (e.g., it can be difficult to obtain a good accuracy with standard methods from machine learning without incorporating expert knowledge).

3.1 Experimental setup

For parameter tuning we use the tuning algorithms to optimize both SVM specific parameters (e.g., kernel parameters and regularization parameters) and problem specific (pre-processing) parameters (e.g., class weighting in classification and embedding in time series regression). In all experiments we set a limit of a certain number of model evaluations. Since tuning algorithms like CMA-ES or SPOT incorporate pseudo-random numbers, and random resampling leads to different data on which the parameter configurations are evaluated, all runs are repeated multiple times. If no repeats are considered, search methods may get trapped in false optima caused by too optimistic models. The detailed settings for parameter tuning and kernel evolution are shown in Tables 2 and 3 respectively.

In kernel evolution, we use a standard strongly typed GP algorithm with a total budget of 2,500 fitness evaluations. The parameter "design factor" determines the size of the LHD design for regularization and kernel parameters. The parameter "breeding tries" defines the number of times a genetic operator is retried until returning its unmodified input when it fails to generate a kernel that satisfies the SVM kernel search space constraints (see Sect. 2.3.2). To prevent loss of diversity during GP search, we use an extinction prevention strategy that prevents the insertion of duplicated individuals to the GP population.

Table 2 Experimental settings for parameter tuning

Setting	AppAcid	AppStorm
Total budget	{50, 100, 200}	250
SPOT initial design size	10	50
SPOT predictor	Kriging	Kriging
Number of runs	5	5

Table 3 General settings for all kernel evolution by GP experiments

Setting	Value
Algorithm	Strongly typed GP
Total budget (Evaluations)	2,500
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
Number of runs (n)	20

3.2 Parameter tuning

3.2.1 Stormwater prediction

In water resource management, efficient controllers of storm-water tanks prevent flooding of sewage 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 [25]. Recently, first work has been proposed using SVM for predicting fill levels of stormwater tanks, which require special pre-processing for time series, e.g., embedding of integrated rainfall [30, 31]. Here, we compare SPOT with other optimization techniques: the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and L-BFGS-B.

Good-working region of interest (ROI) settings were obtained for the stormwater problem by Koch et al. [30, 31]. As pre-processing, a special operator termed *leaky rain* was incorporated for the stormwater problem. It is defined by Eq. 16, where $r(t)$ is the rainfall at time t and *leaky* and W are tunable parameters. The idea of this function is to use convolved rainfall as additional input of the feature set used for training. Here we use two *leaky rain* functions with different parameter settings for W and *leaky*.

$$\sum_{i=0}^{W_1} leaky_1^i \cdot r(t-i) \quad \text{and} \quad \sum_{i=0}^{W_2} leaky_2^i \cdot r(t-i). \qquad (16)$$

For more details on the leaky rain operator see Koch et al. [30, 31]. 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 region of interest (ROI) for the variables is shown in Table 4.

The time series data was split into four consecutive parts of 5,000 records each, named set 1 to set 4. As training set we chose set 2. The hyperparameters were optimized by the search strategies SPOT, CMA-ES and L-BFGS-B. For a comparison between the automatically tuned parameter settings and user-defined settings we give the *hand-tuning* result. Here, all parameters were set to the best values by changing them manually in several preliminary runs.

As objective function we used the RMSE on set 4. The results of this tuning are shown in Table 5. It can be seen that SPOT clearly 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. Nevertheless we included it as a baseline comparison. The CMA-ES produced a good best result but likewise has to cope with a very high standard deviation.

For the last predictive models 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 another test using a different objective function for tuning the model parameters. Otherwise the test set error might be too optimistic since the model has been tuned and tested on the same set. In Tables 6, 7 and 8, we present the results of training a SVM

Table 4 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

Table 5 Parameter tuning for the stormwater problem when trained on training set 2 and evaluated on test set 4 giving a maximum budget of 200 function evaluations for different tuning algorithms

Method	Best	Mean	Worst	SD
SPOT (Kriging)	6.83	7.68	8.12	0.52
L-BFGS-B	12.98	14.12	15.98	1.17
CMA-ES	8.37	10.84	14.42	3.10
Hand-tuning	9.73	—	—	—

Table 6 Results of SPOT tuning on the stormwater problem

	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 %

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

Table 7 Same structure as in Table 6, 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 %

Best values are indicated in bold

Table 8 Same structure as in Table 6, 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 %

Best values are indicated in bold

model on set 2, and tuning the system-relevant parameters on sets 1, 3, 4 (rows) and mutually evaluating the tuned models on all these sets (columns) with SPOT, CMA-ES and L-BFGS-B respectively.

For all tuning algorithms the RMSE is considerably lower when validation set (for tuning) and test set are the same. This becomes clear when best values are always present in the diagonal of the tables, indicating too optimistic tuning results. We quantify this effect by evaluating the following formula: let R_{vt} denote the RMSE for row v and column t of Tables 6, 7 and 8. We define

$$V_t = \frac{S_t - R_{tt}}{R_{tt}} \quad \text{with} \quad S_t = \frac{1}{2}\left(\sum_{v=1}^{3} R_{vt} - R_{tt}\right) \tag{17}$$

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 Tables 6, 7 and 8. In case of no oversearching, a value of V_t close to zero would be expected, whereas values larger than zero indicate oversearching.

The SPOT tuned model chains do not always perform best when validation set and test set are the same (diagonals). However, the best results obtained by CMA-ES and L-BFGS-B seem to be too optimistic, because although any 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 good robustness of SPOT in tuning model chains, although a small amount of oversearching is measurable in any optimization method.

3.2.2 *Acid concentration prediction*

The goal of this benchmark is to classify acid concentrations solely from spectroscopy information of a fluid. In the acid concentration problem the user has defined five classes, each denoting a certain range of acid concentration. The record numbers $N_c = (228, 1528, 1880, 731, 70)$ for each class $c = 1..5$ are highly unbalanced.

The user-defined goal is to maximize the mean class accuracy

$$MCA = \sum_{c=1}^{5} \frac{1}{N_c} \sum_{i=1}^{N_c} U(\mathbf{X}_i) \tag{18}$$

where $U(\mathbf{X}_i)$ is 1 for each correctly predicted record $\mathbf{X}_i$ and 0 otherwise. This means that each of the 70 records of class 5 (they define a critical plant state) has a much higher importance than one of the 1,880 records of class 3. The research question is here whether a tuning of SVM based on TDM can achieve a similar or even better performance than GerDA [53], the so far best approach. GerDA, as described in the work of Wolf et al. [54], learns interesting feature combinations in an unsupervised fashion with an approach based on Boltzmann machines.

The dataset contains 4,437 records with 212 attributes, each attribute denoting a sample point from the spectral curve. Training set (3,326 records) and test set (1,109 records) were defined as in the work by Wolf et al. [54]. The results in Wolf et al. [54] showed that good classification can be obtained if class weights are taken into consideration.

We performed experiments using the Tuned Data Mining (TDM) framework with SVM as classifier.[1] 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 UV/vis spectral curve and thus adjacent attributes have similar values. A principal component analysis (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 so that no further implementation is necessary.
2. To aid the SVM classifier in finding nonlinear 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 Table 9 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 can be reviewed in [34].

[1] As an alternative to SVM we tested also Random Forest (RF) which gave similar results.

Table 9 ROI for the acid concentration problem

Parameter	ROI	Type	Description
XPERC	[0.05, 1.00]	float	Fraction of features taken as input
CUTOFF[1..5]	[0.01 0.40]	float	Voting scheme
CLASSWT[1..5]	[0.05 1.0]	float	Class weight vector
γ	[0.005, 0.3]	float	RBF kernel width
C	[1,10]	float	Regularization term

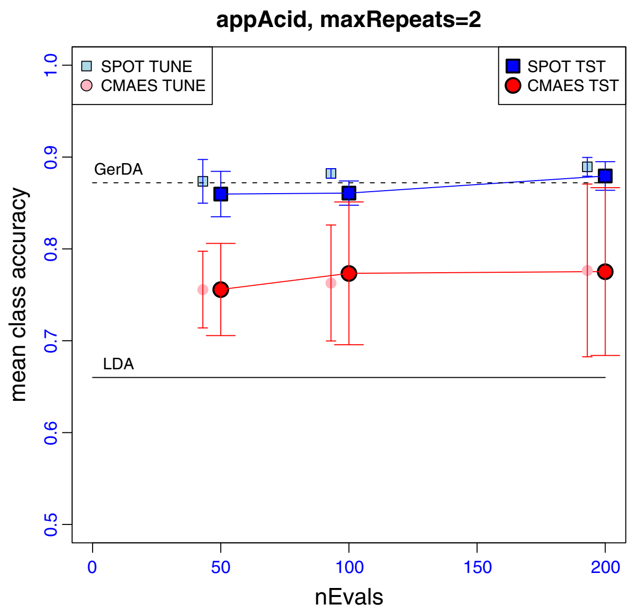

Fig. 3 Results of SVM tuning in the acid concentration problem. The number of model trainings (nEval = 50,100,200) is deliberately set to quite low values, since this training is the time-consuming part of the tuning process. *Error bars* denote standard deviations from 5 repeated experiments with different 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 experiments in Fig. 3 were created by repeatedly executing the following procedure five times:

- In a sequential process each tuner generates design points, that is certain settings for the hyperparameters within their predefined ROIs.
- During tuning only the training set (3,326 records) was used. This set was further split randomly in 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 tuner. This random data split as well as the design point generation of the tuner contain non-deterministic variations, the training process of the SVM itself is fully deterministic. Each design point is evaluated twice (maxRepeats = 2) and the tuner takes the average of the two results reported back.
- A second source of randomness lies in the pre-processing parameter XPERC: Given the pre-processed inputs a variable ranking is performed (see Konen et al. [33] for details) and only those variables from the top of the ranking list are taken which contain together the fraction XPERC of the overall importance. Since the variable ranking is based on Random Forest (RF), it is also subject to slight random variations.
- At the end of the tuning process (which is stopped after nEval = 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 (3,326 records) and the model strength (mean class accuracy) is evaluated on the 1,109 unseen test data records.

We see from Fig. 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.

3.3 Kernel evolution

In this subsection first results of our support vector kernel evolution scheme described in Sect. 2.3.2 based on the general experimental setup described in Sect. 3.1 are reported. To allow for a comparison of our results to earlier work in the field of SVM kernel evolution, we used well known datasets from the UCI machine learning repository [19] for evaluation. These results are linked to the parameter tuning approach discussed earlier 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 tuned by a grid search with discrete settings $2^{-8}, \ldots, 2^{-7}, \ldots, 1, \ldots, 2^{8}$ for both C and γ. In contrast to the GP experiments that used a single holdout set (33 % length) for validation during kernel evolution, in the tRBF experiments we used fivefold cross validation during tuning. In both experiments, the reported test performance is obtained by 20-fold subsampling (80 % training, 20 % test).

Table 11 shows an overview of the test performances 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. The test performances of GP-generated and tRBF kernels is generally quite similar on all considered data sets. 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 10 Tunable parameters, their region of interest (ROI) and best results obtained for the acid concentration problem ($N_{eval} = 200$)

	ROI		Best value					
Name	Low	High	i:	1	2	3	4	5
XPERC	0.05	1.00				0.237		
GAMMA	0.001	0.80				0.195		
COST	0.00	100				54.6		
CLASSWT[i]	0.05	1.00		0.849	0.431	0.246	0.468	0.814
CUTOFF[i]	0.01	0.40		0.216	0.023	0.335	0.389	0.232

There were 12 parameters to tune, since CUTOFF[5] was not tuned but set by a sum constraint

Table 11 Results obtained on test datasets with GP evolved custom kernels (GP) ($N_{eval} = 2,500$, $N_{runs} = 20$) compared with results obtained with grid search tuned RBF kernels (tRBF) on UCI test data sets

	Dataset	Best	Mean	Worst	SD
GP	Ionosphere	0.0282	0.0845	0.1831	0.04293839
	Glass2	0.1212	0.2545	0.3939	0.07513299
	Heart	0.0556	0.1516	0.2593	0.04939896
	Liver	0.1884	0.2929	0.3623	0.04495063
	Pima	0.1883	0.2300	0.2727	0.02374211
tRBF	Ionosphere	0.0282	0.0599	0.1127	0.02279090
	Glass2	0.0606	0.2015	0.3333	0.07375339
	Heart	0.0926	0.1593	0.2593	0.05076841
	Liver	0.2029	0.3043	0.4493	0.05340525
	Pima	0.1688	0.2403	0.3117	0.03304342

All test problems use mean misclassification error (MMCE) as performance measure

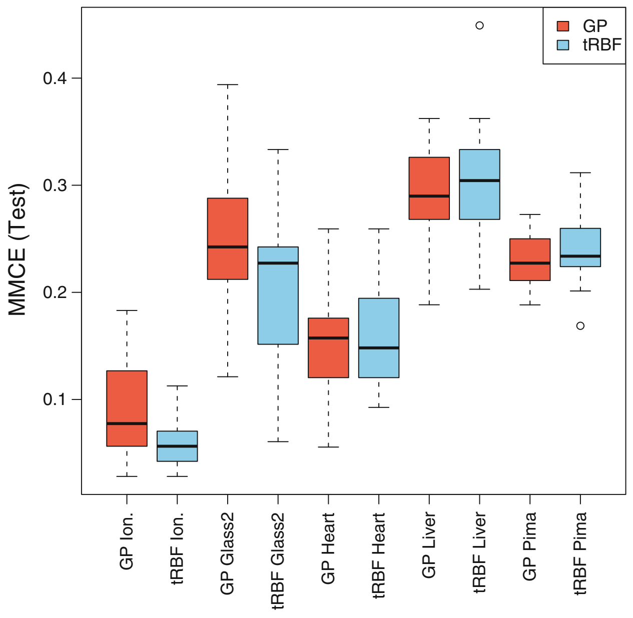

Fig. 4 Boxplots for comparing the test MMCE of GP evolved custom kernels (GP) ($N_{eval} = 2,500$, $N_{runs} = 20$, no tuning) with the test MMCE of grid search tuned RBF kernels (tRBF) on UCI test data sets

Table 12 Best GP evolved kernels for UCI test datasets found in 20 runs ($N_{eval} = 2{,}500$)

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

These kernels where selected based on their performance on validation data

Table 12 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.

4 Discussion

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 [2] 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 with a relatively small number of function evaluations to good results. This has the added benefit of relatively modest compute time requirements, making this approach feasible for use in real-world problems like acid concentration problem and stormwater problem. Compute time requirements 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 a topic of further research.

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 [28], making it necessary to include a mechanism to stop these processes in our evaluation framework.

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. 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 data sets. 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 [13, 22], but we arrive at a more cautious interpretation when comparing to default kernel results. Although we find the contributions of the before 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. [13] only mention that their 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. [22] 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 data sets they evaluate. They achieve significantly worse test set performances for all three data sets 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 might have been the technical difficulties we mentioned above.
3. [50] 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 a 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.

Nonetheless, 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 dataset, 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. Casual observation of the GP search process and of the genealogy 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 [14, 43, 52].

5 Conclusion

In this paper we analyzed the tuning and evolution of Support Vector Machines. The objectives of this paper were twofold: First we demonstrated that good settings for a parameter sensitive method like SVM combined with pre- and post-processing operations can be found almost automatically and efficiently by combining SVM with the right optimization algorithm for parameter tuning (hypothesis **H1**). Among all considered parameter tuning algorithms, SPO showed best results on the tested benchmark problems.

Secondly, a GP system was implemented to discover new SVM kernel functions in an evolutionary process. Our GP system was consistently able to rediscover standard kernels and to discover custom kernels with performances comparable to tuned standard kernels. However, the performance of GP-generated custom kernels does not offer a significant improvement compared to tuned standard kernels, at least on the relatively simple UCI test datasets. We therefore are unable to accept hypothesis **H2** at this point in time. However, our results seem promising enough to warrant further work in improving GP for support vector kernel evolution.

Acknowledgments This work was partly supported by the Research Training Group “Statistical Modelling” of the German Research Foundation, and the Bundesministerium für Bildung und Forschung (BMBF) under the grant SOMA (AiF FKZ 17N1009) and by the Cologne University of Applied Sciences under the research focus grant COSA. Some experimental calculations were performed on the LiDO HPC cluster at the TU Dortmund. We would like to thank the LiDO team at the TU Dortmund for their support.

References

1. Acevedo J, Maldonado-Bascón S, Siegmann P, Lafuente-Arroyo S, Gil P (2007) Tuning L1-SVM hyperparameters with modified radius margin bounds and simulated annealing. In: IWANN, pp 284–291
2. Auger A, Hansen N (2005) A restart CMA evolution strategy with increasing population size. In: Proceedings of the IEEE congress on evolutionary computation, CEC 2005, pp 1769–1776
3. Banzhaf W, Francone FD, Keller RE, Nordin P (1998) Genetic programming: an introduction: on the automatic evolution of computer programs and its applications. Morgan Kaufmann Publishers Inc., San Francisco
4. Bartz-Beielstein T, Flasch O, Koch P, Konen W (2010) SPOT: a toolbox for interactive and automatic tuning in the R environment. In: Hoffmann F, Hüllermeier E (eds) Proceedings 20. Workshop computational intelligence. Universitätsverlag Karlsruhe, pp 264–273
5. Bartz-Beielstein T. (November 2003) Experimental analysis of evolution strategies—overview and comprehensive introduction. Interner Bericht des Sonderforschungsbereichs 531 computational intelligence CI–157/03, Universität Dortmund, Germany
6. Bartz-Beielstein T, Parsopoulos KE, Vrahatis MN (2004) Design and analysis of optimization algorithms using computational statistics. Appl Numer Anal Comput Math (ANACM) 1(2):413–433
7. Bischl B (2011) mlr: Machine learning in R, http://mlr.r-forge.r-project.org
8. Byrd R, Lu P, Nocedal J, Zhu C (1995) A limited memory algorithm for bound constrained optimization. SIAM J Sci Comput 16(5):1190–1208
9. Christmann A, Luebke K, Marin-Galiano M, Rüping S (2005) Determination of hyper-parameters for kernel based classification and regression. Tech. rep., University of Dortmund, Germany
10. Cortes C, Haffner, P, Mohri M (2003) Positive definite rational kernels. Proceedings of the 16th annual conference on computational learning theory (COLT 2003). vol 1, pp 41–56
11. Cortes C, Haffner P, Mohri M (2004) Rational kernels: theory and algorithms. J Mach Learn Res 5:1035–1062
12. Cortes C, Mohri M, Rostamizadeh A (2009) Learning non-linear combinations of kernels. Adv Neural Inf Process Syst 22:396–404
13. Diosan L, Rogozan A, Pecuchet J (2007) Evolving kernel functions for SVMs by genetic programming. In: icmla, pp 19–24. IEEE Comput Soc
14. Droste S, Wiesmann D (2000) Metric based evolutionary algorithms. In: Proceedings of the european conference on genetic programming. Springer-Verlag, London, pp 29–43 http://portal.acm.org/citation.cfm?id=646808.703953
15. Drucker H, Burges C, Kaufman L, Smola A, Vapnik V (1997) Support vector regression machines. Adv Neural Inform Process Syst: 155–161
16. Duan K, Keerthi S, Poo A (2003) Evaluation of simple performance measures for tuning SVM hyperparameters. Neurocomputing 51:41–59
17. Evgeniou T, Pontil M, Poggio T (2000) Regularization networks and support vector machines. Adv Comput Math 13(1): 1–50, http://dx.doi.org/10.1023/A:1018946025316
18. Forrester A, Sobester A, Keane A (2008) Engineering design via surrogate modelling. Wiley, London
19. Frank A, Asuncion A (2010) UCI machine learning repository, http://archive.ics.uci.edu/ml
20. Friedrichs F, Igel C (2005) Evolutionary tuning of multiple SVM parameters. Neurocomputing 64:107–117
21. Fröhlich H, Zell A (2005) Efficient parameter selection for support vector machines in classification and regression via model-based global optimization. In: Neural Networks, 2005. IJCNN ’05. Proceedings. 2005 IEEE international joint conference on. vol 3, pp 1431–1436
22. Gagné C, Schoenauer M, Sebag M, Tomassini M (2006) Genetic programming for kernel-based learning with co-evolving subsets selection. Parallel problem solving from nature-PPSN IX, pp 1008–1017

23. Glasmachers T, Igel C (2010) Maximum Likelihood model selection for 1-norm soft margin svms with multiple parameters. IEEE Transaction pattern analysis and machine intelligence
24. Hansen N (2006) The CMA evolution strategy: a comparing review. In: Lozano J, Larranaga P, Inza I, Bengoetxea E (eds) Towards a new evolutionary computation, Springer, Berlin, pp 75–102
25. Hilmer T (2008) Water in society—integrated optimisation of sewerage systems and wastewater treatment plants with computational intelligence tools. Ph.D. thesis, Open Universiteit Nederland, Heerlen
26. Howley T, Madden M (2005) The genetic kernel support vector machine: description and evaluation. Artif Intell Rev 24(3):379–395
27. Jones DR (December 2001) A taxonomy of global optimization methods based on response surfaces. J Global Optim 21: 345–383, http://dx.doi.org/10.1023/A:1012771025575
28. Karatzoglou A, Smola A, Hornik K, Zeileis A (2004) kernlab – an S4 package for kernel methods in R. J Stat Softw 11(9): 1–20, http://www.jstatsoft.org/v11/i09/
29. Keerthi S, Sindhwani V, Chapelle O (2007) An efficient method for gradient-based adaptation of hyperparameters in SVM models. Adv Neural Inform Process Syst 19:673–680
30. Koch P, Bartz-Beielstein T, Konen W (2010) Optimization of support vector regression models for stormwater prediction. In: Hoffmann F, Hüllermeier E (eds) Proceedings 20. workshop computational intelligence. Universitätsverlag Karlsruhe, http://www.gm.fh-koeln.de/konen/Publikationen/GMACI10_optimSVR.pdf
31. Koch P, Konen W, Flasch O, Bartz-Beielstein T (2010) Optimizing support vector machines for stormwater prediction. In: Bartz-Beielstein T, Chiarandini M, Paquete L, Preuss M (eds) Proceedings of workshop on experimental methods for the assessment of computational systems joint to PPSN2010. No. TR10-2-007, TU Dortmund, pp 47–59. ls11-http://www.cs.tu-dortmund.de/_media/techreports/tr10-07.pdf
32. Konen W (2011) The TDM framework: tuned data mining in R. CIOP Technical Report 01-11, Cologne University of Applied Sciences
33. Konen W, Koch P, Flasch O, Bartz-Beielstein T (2010) Parameter-tuned data mining: a general framework. In: Hoffmann F, Hüllermeier E (eds) Proceedings 20. Workshop computational intelligence. Universitätsverlag Karlsruhe, http://www.gm.fh-koeln.de/konen/Publikationen/GMACI10_tunedDM.pdf
34. Konen W, Koch P, Flasch O, Bartz-Beielstein T, Friese M, Naujoks B (2011) Tuned data mining: a benchmark study on different tuners. CIOP Technical Report 02-11, Cologne University of Applied Sciences
35. Koza J (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
36. McKay MD, Beckman RJ, Conover WJ (1979) A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 21(2):239–245
37. Mercer J (1909) Functions of positive and negative type, and their connection with the theory of integral equations. Philos Trans R Soc Lond 209:415–446
38. Momma M, Bennett K (2002) A pattern search method for model selection of support vector regression. In: SDM, pp 1345–1350
39. Müller KR, Smola AJ, Rätsch G, Schölkopf B, Kohlmorgen J, Vapnik V (1999) Using support vector machines for time series prediction, pp 243–253. MIT Press, Cambridge http://portal.acm.org/citation.cfm?id=299094.299107
40. Ojeda F, Suykens JAK, Moor BD (2008) Low rank updated ls-svm classifiers for fast variable selection. Neural Networks 21(2-3):437–449
41. Poli R, Langdon WB, McPhee NF (2008) A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, (With contributions by J. R. Koza)
42. Rasmussen CE, Williams CKI (2005) Gaussian processes for machine learning. The MIT Press, Cambridge
43. Rosca JP, Rosca, JP, Ballard DH, Ballard DH (1995) Causality in genetic programming. In: Genetic algorithms: proceedings of the sixth international conference (ICGA95. pp 256–263. Morgan Kaufmann
44. Sacks J, Welch WJ, Mitchell TJ, Wynn HP (1989) Design and analysis of computer experiments. Stat Sci 4(4):409–435
45. Santner TJ, Williams BJ, Notz WI (2003) The design and analysis of computer experiments. Springer, Berlin, Heidelberg, New York
46. Sasena MJ, Papalambros P, Goovaerts P (2002) Exploration of metamodeling sampling criteria for constrained global optimization. Eng Optim 34:263–278
47. Schölkopf B, Burges C, Smola A (1999) Advances in kernel methods: support vector learning. The MIT press, Cambridge
48. Sollich P (2002) Bayesian methods for support vector machines: evidence and predictive class probabilities. Mach Learn 46(1-3): 21–52
49. Staelin C (2003) Parameter selection for support vector machines. Hewlett-Packard Company, Tech. Rep. HPL-2002-354R1
50. Sullivan K, Luke S (2007) Evolving kernels for support vector machine classification. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation. p. 1707. ACM
51. Vapnik V (1995) The nature of statistical learning theory. Springer, NY
52. Wiesmann D (2002) From syntactical to semantical mutation operators for structure optimization. In: Proceedings of the 7th international conference on parallel problem solving from nature. PPSN VII, Springer, London, pp 234–246 http://portal.acm.org/citation.cfm?id=645826.669440
53. Wolf C, Gaida D, Stuhlsatz A, Ludwig T, McLoone S, Bongards M (2011) Predicting organic acid concentration from UV/vis spectro measurements - a comparison of machine learning techniques. Trans Inst Meas Control
54. Wolf C, Gaida D, Stuhlsatz A, McLoone S, Bongards M (2010) Organic acid prediction in biogas plants using UV/vis spectroscopic online-measurements. Life system modeling and intelligent computing 97: 200–206, http://dx.doi.org/10.1007/978-3-642-15853-7_25
55. Zhu C, Byrd R, Lu P, Nocedal J (1997) Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. ACM Trans Mathematical Software (TOMS) 23(4):550–560