Evolutionary Optimization of Least-Squares Support Vector Machines

Arjan Gijsberts, Giorgio Metta and Léon Rothkrantz

Abstract The performance of Kernel Machines depends to a large extent on its kernel function and hyperparameters. Selecting these is traditionally done using intuition or a costly "trial-and-error" approach, which typically prevents these methods from being used to their fullest extent. Therefore, two automated approaches are presented for the selection of a suitable kernel function and optimal hyperparameters for the Least-Squares Support Vector Machine. The first approach uses Evolution Strategies, Genetic Algorithms, and Genetic Algorithms with floating point representation to find optimal hyperparameters in a timely manner. On benchmark data sets the standard Genetic Algorithms approach outperforms the two other evolutionary algorithms and is shown to be more efficient than grid search. The second approach aims to improve the generalization capacity of the machine by evolving combined kernel functions using Genetic Programming. Empirical studies show that this model indeed increases the generalization performance of the machine, although this improvement comes at a high computational cost. This suggests that the approach may be justified primarily in applications where prediction errors can have severe consequences, such as in medical settings.

Arjan Gijsberts
Italian Institute of Technology, Via Morego, 30 – Genoa 16163, Italy
Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
e-mail: arjan.gijsberts@iit.it

Giorgio Metta
Italian Institute of Technology, Via Morego, 30 – Genoa 16163, Italy
University of Genoa, Viale F. Causa, 13 – Genoa 16145, Italy
e-mail: giorgio.metta@iit.it

Léon Rothkrantz
Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
Netherlands Defence Academy, Nieuwe Diep 8, 1781 AT Den Helder, The Netherlands
e-mail: L.J.M.Rothkrantz@ewi.tudelft.nl

1 Introduction

Kernel Machines allow the construction of powerful, non-linear classifiers using relatively simple mathematical and computational techniques [35]. As such, they have successfully been applied in fields as diverse as data mining, economics, biology, medicine, and robotics. Much of the success of the Kernel Machines is due to the *kernel trick*, which can best be described as an implicit mapping of the input data into a high dimensional feature space. In this manner, the algorithms can be applied in a high dimensional space, without the need to explicitly map the data points. This implicit mapping is done by means of a *kernel function*, which represents the inner product for the specific hypothetical feature space.

The performance of Kernel Machines is highly dependent on the chosen kernel function and parameter settings. Unfortunately, there are no analytical methods or strong heuristics that can guide the user in selecting an appropriate kernel function and good parameter values. The common way of finding optimal hyperparameters is to use a costly grid search, which scales exponentially with the number of parameters. Additionally, it is usually necessary to manually determine the region and resolution of the search to ensure computational feasibility. Selection of the kernel function is done similarly, i.e. either trial-and-error or only considering the default Gaussian kernel function. Consequently, tuning the techniques may be arduous, such that less than optimal performance is achieved. For a successful integration in real-life information systems, Kernel Machines should be combined with an automated, efficient optimization strategy for both hyperparameters and kernel function.

Two distinct approaches are proposed for the automated selection of the parameters and the kernel function itself. These models are based on techniques that fall in the class of Evolutionary Computation, which are techniques inspired by neo-Darwinian evolution. The first approach uses evolutionary algorithms to optimize the hyperparameters of a Kernel Machine in a time-efficient manner. The second aims to increase the generalization performance by constructing combined, problem-specific kernel functions using Genetic Programming. Implementations of both approaches have been evaluated on seven benchmark data sets, for which traditional grid search was used as a reference.

Kernel Machines and the kernel trick are presented in Sect. 2. We emphasize on one particular type of Kernel Machine, namely the Least-Squares Support Vector Machine. In Sect. 3, an introduction is given into the evolutionary algorithms that are used in the models. A review of related work on hyperparameter optimization and kernel construction is given in Sect. 4. The two approaches are presented in Sect. 5, after which the experimental results are presented in Sect. 6. The paper is finalized in Sect. 7 with the conclusions and suggestions for future work.

2 Kernel Machines

All Kernel Machines rely on a kernel function to transform a non-linear problem into a linear one by mapping the input data into a hypothetical, high dimensional feature space. This mapping – the *kernel trick* – is not done explicitly, as the kernel function calculates the inner product in the corresponding feature space. The kernel trick is explained together with the *Least-Squares Support Vector Machine* (LS-SVM), which is a particular type of Kernel Machine.

2.1 Least-Squares Support Vector Machines

Assume a set of ℓ labeled training samples, i.e. $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$, where $\mathbf{x} \in \mathscr{X} \subseteq \mathbb{R}^n$ is an input vector of n features and $y \in \mathscr{Y}$ is the corresponding label. In the case $\mathscr{Y}$ denotes a set of discrete classes, e.g. $\mathscr{Y} \subseteq \{-1, 1\}$, then the problem is considered a *classification* problem. Conversely, if $\mathscr{Y} \subseteq \mathbb{R}$, then we are dealing with a *regression* problem. The LS-SVM aims to construct a linear function [37]

$$f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b \ , \tag{1}$$

which is able to predict an output value y given an input sample $\mathbf{x}$. Note that for binary classification purposes it is necessary to apply the sign function on the predicted output value. The error in the prediction for each sample i is defined as

$$y_i - (\langle \mathbf{x_i}, \mathbf{w} \rangle + b) = \varepsilon_i \qquad\qquad \text{for } 1 \leq i \leq \ell \ . \tag{2}$$

The optimization problem in LS-SVM is analogous to that of traditional Support Vector Machines (SVM) [38]. The goal is to minimize both the norm of the weight vector $\mathbf{w}$ (i.e. maximize the margin) and the sum of the squared errors. In contrast to SVM, LS-SVM uses *equality constraints* for the errors instead of *inequality constraints*. Combining the optimization problem with the equality constraints for the errors (2), one obtains

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2}C\sum_{i=1}^{\ell}\varepsilon_i^2 && && (3)\\ &\text{subject to} && y_i = \langle \mathbf{x}_i, \mathbf{w} \rangle + b + \varepsilon_i && \text{for } 1 \leq i \leq \ell \ , \end{aligned}$$

where C is the regularization parameter. Reformulating this optimization problem as a Lagrangian gives the unconstrained minimization problem

$$\frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2}C\sum_{i=1}^{\ell}\varepsilon_i^2 - \sum_{i=1}^{\ell}\alpha_i\left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b + \varepsilon_i - y_i\right) \ , \tag{4}$$

where $\alpha_i \in \mathbb{R}$ for $1 \leq i \leq \ell$. Note that the Lagrange multipliers α_i can be either positive or negative, due to the equality constraints in the LS-SVM algorithm. The optimality conditions for this problem can be obtained by setting all derivatives equal to zero. This yields a set of linear equations

$$\sum_{j=1}^{\ell} \alpha_j \left\langle \mathbf{x}_j, \mathbf{x}_i \right\rangle + b + C^{-1}\alpha_i = y_i \qquad \text{for } 1 \leq i \leq \ell \ . \tag{5}$$

2.2 Kernel Functions

We observe that the training samples are only present within the inner products in (5). The kernel function used to compute an inner product is defined as

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle \ , \tag{6}$$

where $\phi(\mathbf{x})$ is the mapping of the input samples into a feature space. If we substitute the standard inner product with a kernel function in (5), we obtain the "kernelized" variant

$$\sum_{j=1}^{\ell} \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) + b + C^{-1}\alpha_i = y_i \qquad \text{for } 1 \leq i \leq \ell \ . \tag{7}$$

Usually it is convenient to define a symmetric kernel matrix as $\mathbf{K} = \left(k(\mathbf{x}_i, \mathbf{x}_j)\right)_{i,j=1}^{\ell}$, so that the system of linear equations can be rewritten as

$$\begin{bmatrix} \mathbf{K} + C^{-1}\mathbf{I} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ 0 \end{bmatrix} \ . \tag{8}$$

Note that the bottom row and rightmost column have been added to integrate the bias b in the system of linear equations. Other than the sign function, the algorithm is identical for both regression and classification. After the optimal Lagrange multipliers and bias have been obtained using (8), unseen samples can be predicted using

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b \ . \tag{9}$$

2.2.1 Conditions for Kernels

It is important to obtain functions that correspond to an inner product in some feature space. *Mercer's theorem* states that valid kernel functions must be symmetric, continuous, and positive semi-definite [38], formalized as the following condition:

$$\int_{\mathscr{X} \times \mathscr{X}} k(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) \, d\mathbf{x} d\mathbf{z} \geq 0 \qquad \text{for all } f \in L_2(\mathscr{X}) \ . \tag{10}$$

Kernel functions that satisfy these conditions are referred to as *admissible kernel functions*. If this condition is satisfied, then the kernel matrix is accordingly positive semi-definite [4]. Unfortunately, it is not trivial to verify that a kernel function satisfies Mercer's condition, nor whether the kernel matrix is positive semi-definite. There are, however, certain functions that have analytically been proven to be admissible. Common kernel functions – for classification and regression purposes – include the *polynomial* (11), the *RBF* (12), and the *sigmoid* function (13). Note that the sigmoid kernel function is only admissible for certain parameter values.

$$k(\mathbf{x},\mathbf{z}) = (\langle \mathbf{x},\mathbf{z}\rangle + c)^d \qquad \text{for } d \in \mathbb{N},\ c \geq 0 \tag{11}$$

$$k(\mathbf{x},\mathbf{z}) = \exp\left(-\gamma \|\mathbf{x}-\mathbf{z}\|^2\right) \qquad \text{for } \gamma > 0 \tag{12}$$

$$k(\mathbf{x},\mathbf{z}) = \tanh(\gamma \langle \mathbf{x},\mathbf{z}\rangle + c) \qquad \text{for some } \gamma > 0, c \geq 0 \tag{13}$$

All these function are parameterized, allowing for adjustments with respect to the training data. The kernel parameter(s) and the regularization parameter C are the *hyperparameters*. The performance of an LS-SVM (or an SVM, for that matter) is *critically dependent* on the selection of hyperparameters.

Mercer's condition can be used to infer simple operations for creating combined kernel functions, which are also admissible. For instance, assume that k_1 and k_2 are admissible kernel functions, then the following combined kernels are admissible [35]:

$$k(\mathbf{x},\mathbf{z}) = c_1 k_1(\mathbf{x},\mathbf{z}) + c_2 k_2(\mathbf{x},\mathbf{z}) \qquad \text{for } c_1, c_2 \geq 0 \tag{14}$$

$$k(\mathbf{x},\mathbf{z}) = k_1(\mathbf{x},\mathbf{z})\, k_2(\mathbf{x},\mathbf{z}) \tag{15}$$

$$k(\mathbf{x},\mathbf{z}) = a k_2(\mathbf{x},\mathbf{z}) \qquad \text{for } a \geq 0 \tag{16}$$

Moreover, these operations allow modular construction of kernel functions. Increasingly complex kernel functions can be constructed by recursively applying these operations.

3 Evolutionary Computation

Several biologically inspired techniques have been developed over the years for search, optimization, and machine learning under the collective term *Evolutionary Computation* (EC) [40]. The key principle in EC is that potential solutions are generated, evaluated, and reproduced iteratively. Between iterations, individuals are subject to certain forms of mutation and can reproduce with a probability that is proportional to their *fitness*. A selection procedure removes individuals with low fitness from the population, so that the more fit ones are more likely to "survive". Three of the main branches within EC are *Genetic Algorithms*, *Evolution Strategies*, and *Genetic Programming*.

3.1 Genetic Algorithms

Probably the most recognized form of EC is the class of *Genetic Algorithms* (GA), popularized by Holland [15]. Genetic Algorithms mainly operate in the realm of the genotype, which is commonly represented as a bitstring. This means that all parameters need to be converted to a binary representation and are then concatenated to form the chromosome. Various types of bit encoding may be used, such as *Gray codes* or even floating point representations.

Reproduction of individuals is usually emphasized in preference to mutation in GA. Two or more parents exchange part of their chromosome, resulting in offspring that contains genetic information from each of the parents. The common implementation is *crossover recombination*, in which two parents exchange a fragment of their chromosome. The size of the fragment is determined by a randomly selected crossover point. Mutation, on the other hand, is implemented by flipping the bits in the chromosome with a certain probability. Note that the implementation of both reproduction and mutation operators may depend on the specific representation that is used. For instance, reproduction of floating point chromosomes is done by blending the parents [10].

In addition to the mutation and recombination operators, the other key element in GA is the selection mechanism. The selection procedure selects the individuals that will be subject to mutation and reproduction with a probability proportional to their fitness. Further, offspring can be created on a *generational* interval or, alternatively, individuals can be replaced one by one (i.e. *steady state* GA).

3.2 Evolution Strategies

Evolution Strategies (ES) operate in the realm of the phenotype and use real-valued representations for the individuals [2]. An optimization problem with three parameters is represented as a vector $\mathbf{c} = (x_1, x_2, x_3)$, where the parameters $x_i \in \mathbb{R}$ are the *object parameters*. There are two main types of ES, namely $(\mu + \lambda)$-ES and (μ, λ)-ES. In these notations, μ is the size of the parent population and λ is the size of the offspring population. In $(\mu + \lambda)$-ES, the new parent population is chosen from *both* the current parent population and the offspring. In contrast, in (μ, λ)-ES the new parent population is chosen only from the offspring population, which requires that $\lambda \geq \mu$.

The canonical ES relies solely on the mutation operation for diversifying the genetic material. The mutation operation is typically implemented as a random perturbation of the parameters according to a probability distribution. More formally,

$$x'_i = x_i + \mathscr{N}_i(0, \sigma_i) \quad , \tag{17}$$

where $\mathscr{N}$ denotes a logarithmic normal distribution. Note that this mutation mechanism requires the user to specify a standard deviation σ_i (i.e. the *strategy parame-*

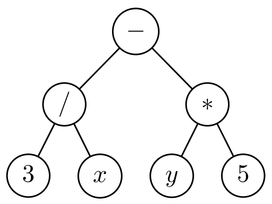


Fig. 1 An example tree representation for the mathematical function $(3/x) - (y*5)$.

ters) for each object parameter in the chromosome. The common approach is to not define these standard deviations explicitly, but to integrate them in the chromosome. This is known as *self adaptation*, as certain parameters of the algorithm are subject to the algorithm itself. An example of a chromosome with three object parameters and the additional *endogenous strategy parameters* is $\mathbf{c} = (x_1, x_2, x_3, \sigma_1, \sigma_2, \sigma_3)$ [3].

3.3 Genetic Programming

A vastly different paradigm within EC is that of *Genetic Programming* (GP) [22]. GP should rather be considered a form of automated programming than a parameter optimization technique. It aims to solve a problem by breeding a population of computer programs, which – when executed – are direct solutions to the problem. Obviously, this gives much more freedom in the structure of the solutions and it can therefore be applied to wide variety of problems. The common way to represent programs in GP is by means of *syntax trees*, as shown in Fig. 1. Other types of genotype representations, e.g. graphs or linear structures, may be preferred for certain problem domains.

GP includes recombination and mutation operators that are similar to their GA counterparts. In crossover recombination, two parents swap a sub-tree rooted at a random crossover point. Traditional mutation in GP involves randomly selecting a mutation point in the tree and replacing the sub-tree rooted at this point with a new, randomly generated tree.

For some problems it may be desirable to impose restrictions on the structure of the syntax tree, as to ensure that non-terminals operate only on appropriate data types. Consider, for instance, a binary equality function, which takes two integers as its children and returns a boolean. *Strongly Typed Genetic Programming* has been proposed as an enhanced version of GP that enforces this type of constraint [28]. This influences both the representation of the individuals and the chromosome altering operators. Firstly, while defining the terminals and non-terminals, the user also has to specify the types of the terminals, and the parameter and return types of non-terminals. Secondly, the recombination and mutation operators must be altered in such a way that they respect the type constraints.

4 Related Work

Hyperparameters and the kernel function are usually selected using a *trial-and-error* approach. Trial runs are performed using various configurations, the best of which is selected. This approach is generally considered time consuming and does not scale well with the number of parameters. Furthermore, the process often yields less than optimal performance in situations where time is a limited. More elaborate approaches have been suggested for both selection problems, which will be summarized below.

4.1 Hyperparameter Optimization

An analytical technique that has been proposed for hyperparameter optimization is that of *gradient descent* [5, 20], which finds a local minimum by taking steps in the negative gradient direction. This approach has been used for hyperparameter selection with a non-spherical RBF function, which means that each feature has a distinct scaling factor. Accordingly, there are more hyperparameters than there are features, demonstrating the scalability of the approach. The gradient descent method is shown to be able to find reasonable hyperparameters more efficiently than grid search. However, the method requires a continuous differentiable kernel and objective function, which may not be satisfiable for specific types of problems (e.g. non-vectorial kernel functions). Approaches based on *pattern search* have been proposed to overcome this problem [27]. In this method the neighborhood of a parameter vector is investigated in order to *approximate* the gradient empirically. However, the whole class of gradient descent methods has the inherent disadvantage that they may find local minima.

One of the first mentions of the use of EC for hyperparameter optimization can be found in the work of Fröhlich et al. [12], in which GA is primarily used for feature selection. However, the optimization of the regularization parameter C is done in parallel. Other GA-based approaches focus mainly on the optimization of the hyperparameters. The objective function in these type of approaches is either the error on a validation set [18, 26, 29], the radius-margin bound [7], or k-fold cross validation [6,33]. Some studies make use of a real-valued variant of GA [17,42], although it is not clear whether the real-valued representation performs significantly better than a binary representation. All these studies suggest that GA can successfully be applied for hyperparameter optimization. However, there are some caveats, such as heterogeneity of the solutions and the selection of a reliable *and* efficient objective function.

ES have only scarcely been used for hyperparameter optimization [11]. In this approach, ES optimizes not only the scaling, but also the orientation of the RBF kernel. An improvement on the generalization performance is achieved over the kernel parameters that were found using grid search. This result should be interpreted with care, as the optimal grid search parameters are used as the initial solutions for the evolutionary algorithm. The classification error on separate test sets is used as the empirical objective function.

The main advantage of evolutionary algorithms in comparison to grid search is that they usually find good parameter settings efficiently and that the technique scales well with the number of hyperparameters. An advantage compared to gradient descent methods is that they cope better with local minima. Furthermore, they do not impose requirements on the kernel and objective functions, such as differentiability.

4.2 Combined Kernel Functions

It is intuitive that combined kernel functions are capable of improving the generalization performance, as the implicit feature mapping can be tuned for a specific problem. Several methods have been proposed for the composition of kernel functions. One of the first manifestations of combined kernel optimization was investigated by Lanckriet et al. [23]. This work considers linear combinations of kernels, i.e. $\mathbf{K} = \sum_{i=0}^{m} a_i \mathbf{K}_i$ for $a > 0$ and $\mathbf{K}_i$ chosen from a predefined set of kernel functions. The optimization of weight factors $\mathbf{a}$ is done using *semi-definite programming*, which is an optimization method that deals with convex functions over the convex cone of positive semi-definite matrices. This method can be applied to kernel matrices, since these need to be semi-definite to satisfy Mercer's condition. However, other methods may be used for the optimization of the weights, such as so-called *hyperkernels* [30], the *Lagrange multiplier* method [19], or using a *generalized eigenvalue* approach [36].

Lee et al. argue that during the combination of kernels some potentially useful information is lost [24]. They propose a method for combining kernels that aims to prevent this loss of information. Instead of combining various kernel matrices into one, their method creates a large kernel matrix that contains all original kernel matrices and all possible mixtures of kernel functions, e.g. $k_{i,j}(\mathbf{x},\mathbf{z}) = \langle \phi_i(\mathbf{x}), \phi_j(\mathbf{z}) \rangle$, where ϕ_i is the mapping that belongs to kernel function k_i and ϕ_j the mapping that belongs to kernel k_j. This eliminates the requirement to optimize the weight factor for each kernel, as this is done implicitly by the SVM algorithm. However, special mixture functions need to be provided for the combination of two kernel functions. Furthermore, the spatial and temporal requirements of the algorithm increases drastically, as the kernel matrix is enlarged in both dimensions in proportion to the number of kernels in the combination.

Other EC inspired approaches have been proposed to combine kernel functions. Most of these optimize a linear combination of weighted kernels using either GA or ES. The distinguishing elements are the set of kernel functions that is considered and the type of combination operators. Some only consider linear combinations (i.e. the addition operator) [9, 31], whilst others may allow both addition and multiplication [25]. These studies suggest that combining kernel functions can improve the generalization performance of the machine. However, the combinations are restricted to a predefined size and structure.

Howley and Madden propose a method to construct complete kernel functions using GP [16]. In this method, a kernel function is evolved for use with an SVM classifier. They use a tree structured genotype, with the operators $+$, $-$, and $\times$ in both scalar and vector variants as the non-terminals. The terminals in their approach are the two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. Since the kernels are constructed using simple arithmetic, they are not guaranteed to satisfy Mercer's condition. Nonetheless, the technique still keeps up with (or outperforms) traditional kernels for most data sets. It is emphasized that techniques such as GP require a sufficiently large data set. Dioşan et al. have proposed some enhancements; their method differs from the original approach by an enriched operator set (e.g. various norms are included) and small changes to certain operators [8]. Similar modifications are presented for Kernel Nearest-Neighbor classification by Gagné et al., who also use co-evolution to keep the approach computationally tractable [14]. Besides a species that evolves kernel functions, there are two other species for the training and validation sets. The training set species cooperates with the kernel function on minimizing the error and thus maximizing the fitness, whereas the species for the validation set is competitive and tries to maximize the error of the kernel functions.

5 Evolutionary Optimization of Kernel Machines

Two methods for the evolutionary optimization of hyperparameters and the kernel function are proposed. The first approach uses ES, GA, and GA with floating point representation to optimize the hyperparameters for a given kernel function (EvoKM$^{\text{ES}}$, EvoKM$^{\text{GA}}$, and EvoKM$^{\text{GAflt}}$, respectively). The aim is to find optimal hyperparameters more efficiently than using traditional grid search. Our second model uses GP to evolve combined kernel functions (EvoKM$^{\text{GP}}$), with the aim to increase the generalization performance.

5.1 Hyperparameter Optimization

In the hyperparameter optimization models, ES and GA are used to optimize the hyperparameters θ. Two variants of the GA model have been implemented; one that uses the traditional bitstring representation with Gray coding and another that uses a floating point representation. Evolutionary algorithms are highly generalized and their application on this specific problem is straightforward.

In EvoKM$^{\text{ES}}$, the chromosomes contain the real-valued hyperparameters and the corresponding endogenous strategy parameters σ, which yields for the RBF kernel the chromosome $\mathbf{c} = \left[\gamma, C, \sigma_\gamma, \sigma_C\right]$. Note that all the models use the hyperparameters on a logarithmic scale with base-2. Each hyperparameter is initialized to the center of its range and mutated according to the initial standard deviation $\sigma_i = 1.0$. An interesting issue is whether to use $(\mu + \lambda)$-ES or (μ, λ)-ES in the model. Both types have their own specific advantages and disadvantages. Typical application areas of $(\mu+\lambda)$-ES are discrete finite size search spaces, such as combinatorial optimization problems [3]. When the problem is an unbounded, typically real-valued search spaces, then (μ,λ)-ES is preferred [34]. Furthermore, Whitley presents empirical evidence that indicates that (μ,λ)-ES generally performs better than $(\mu+\lambda)$-ES [41]. We prefer to follow both the heuristic and the empirical indications and adopted (μ,λ)-ES for our model. Unfortunately, there is no guarantee that the search process will converge, as would have been the case with $(\mu+\lambda)$-ES. For our model, we have empirically selected $\mu = 3$ and $\lambda = 12$ based on preliminary experimentations.

EvoKM$^{\mathrm{GA}}$ and EvoKM$^{\mathrm{GAflt}}$ differ from EvoKM$^{\mathrm{ES}}$ in terms of the the operators and the genotype representation. EvoKM$^{\mathrm{GA}}$ uses a Gray code of 18 bits for each parameter, and *one-point crossover recombination* and *bit-flip mutation* operators. One disadvantage of GA, as compared to ES, is that there are many more parameters parameters that need to be set. The population size of 10 is relatively low for GA standards. However, one must take into account that the maximum number of evaluations is limited to several hundreds up to a few thousand and, moreover, the goal is to see convergence to good solutions within the first hundred evaluations. Large population sizes, e.g. larger than 50, would have a disadvantage in this context, as the algorithm can only perform one or two generations within this range. Further, preliminary experiments have shown that a population size of 10 shows similar convergence to larger population sizes. Other parameters of EvoKM$^{\mathrm{GA}}$ have been tuned using a coarse grid search as well. One-point crossover recombination occurs with a probability of $p_c = 0.2$. During mutation, each bit in the chromosome is inverted with a probability of $p_m = 0.1$. The number of participants in tournament selection is 5. Further, the *steady state* variant of GA has been used.

EvoKM$^{\mathrm{GAflt}}$, on the other hand, uses a floating point representation. Crossover recombination in this model is performed by blending two individuals using the BLX-α method [10]. This recombination operator is applied with a probability of $p_c = 0.3$ and with $\alpha = 0.5$. Additionally, each parameter has a probability of $p_m = 0.4$ of being mutated using a random perturbation according to the normal distribution $\mathcal{N}(\mu,\sigma)$, where $\mu = 0$ and $\sigma = 0.5$. All other settings are equal to those for EvoKM$^{\mathrm{GA}}$.

5.2 Kernel Construction

The second optimization method constructs complete kernel functions using GP. In this model, the functions are represented using syntax trees. The syntactic structure of the trees is based on the combination operations that guarantee admissible kernel functions, cf. (14), (15), and (16). These operations form the set of non-terminals, whereas the polynomial and RBF kernels form the set of terminals. This is formalized in the context-free grammar shown in Fig. 2. The model makes use of Strongly Typed GP, as it needs to ensure that the syntactic structure is enforced for all indi-

$\langle$*kernel*$\rangle$	$\rightarrow$	$\langle$*add_kernels*$\rangle$ | $\langle$*multiply_kernels*$\rangle$ | $\langle$*weighted_kernel*$\rangle$ | $\langle$*polynomial*$\rangle$ | $\langle$*rbf*$\rangle$	
$\langle$*add_kernels*$\rangle$	$\rightarrow$	$\langle$*kernel*$\rangle$ **‘+’** $\langle$*kernel*$\rangle$	
$\langle$*multiply_kernels*$\rangle$	$\rightarrow$	$\langle$*kernel*$\rangle$ ‘$\times$’ $\langle$*kernel*$\rangle$	
$\langle$*weighted_kernel*$\rangle$	$\rightarrow$	a ‘$\times$’ $\langle$*kernel*$\rangle$	for $a \in \mathbb{R}^+$
$\langle$*polynomial*$\rangle$	$\rightarrow$	‘$(\langle \mathbf{x}, \mathbf{z} \rangle + c)^d$’	for $d \in \mathbb{N},\ c \in \mathbb{R}^+$
$\langle$*rbf*$\rangle$	$\rightarrow$	‘$\exp\left(-\gamma \|\|\mathbf{x} - \mathbf{z}\|\|^2\right)$’	for $\gamma \in \mathbb{R}^+$

Fig. 2 Context-free grammar – in Backus-Naur form – that constrains the generated expressions for the GP model.

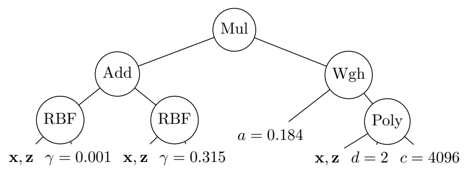

Mul
Add
Wgh
RBF
RBF
Poly
$a = 0.184$
$\mathbf{x}, \mathbf{z}$ $\gamma = 0.001$
$\mathbf{x}, \mathbf{z}$ $\gamma = 0.315$
$\mathbf{x}, \mathbf{z}$ $d = 2$ $c = 4096$

Fig. 3 An example of a tree generated by the GP model.

viduals. An example chromosome of a kernel function using the tree representation is shown in Fig. 3. Note that the regularization parameter C is omitted in this figure; it is included in an separate real-valued chromosome.

A common heuristic with regard to population size in GP is that difficult problems require a large population. As time efficiency is of primary concern for this model, the population size is set to 2000 and each run spans 13 generations[1]. Further, the following operators and settings are used within the EvoKM$^{\text{GP}}$ model:

1. *Reproduction* occurs with probability $p_r = 0.05$, i.e. an individual is directly copied into the offspring population, without any kind of mutation.
2. *Crossover recombination* occurs with probability $p_c = 0.2$, i.e. two parents exchange a subtree at a random crossover point; the two new individuals are *both* inserted in the offspring population.
3. *Random mutation* occurs with probability $p_m = 0.15$, i.e. substituting a subtree of the individual with a new random subtree.
4. *Shrink mutation* occurs with probability $p_s = 0.05$, i.e. replacing a subtree with one of the branches of this subtree in order to reduce the size of the tree.
5. *Swap mutation* occurs with probability $p_w = 0.05$, i.e. replacing a subtree in the individual with another subtree, effectively swapping two branches of the same tree.
6. *PDF parameter mutation* occurs with probability $p_p = 0.5$, i.e. mutating a hyperparameter according to a probability density function.

The PDF mutation operator is specially crafted for our model. This operator ensures that the optimization includes the hyperparameters, as well as evolving the

[1] The total number of evaluations will thus be less than 26000, as unmodified individuals are not reevaluated.

structure of the kernel functions. The common GP operators would only be able to mutate these parameters by substituting them for another randomly selected parameter.

5.3 Objective Function

When applying EC techniques it is important to decide which objective function to use, as this is the actual measure that is being optimized. It should, therefore, measure the “quality” of a solution for the given domain. In the context of this study quality is best described as the generalization performance of the machine. A very important aspect is that the fitness function must prevent overfitting of the machine to the training data. This is especially true for EvoKM$^{\mathrm{GP}}$, as this model tunes both the hyperparameters and the kernel function for the specific data set. There are several methods to estimate this generalization performance, of which cross validation can be applied to practically any learning method. Both k-fold and leave-one-out cross validation have been shown to be approximately unbiased in terms of estimating the true expected error [21]. However, k-fold cross validation usually exhibits a lower variance on the error than the leave-one-out measure. For this reason, k-fold cross validation is used as fitness function for both approaches.

6 Results

All models have been validated experimentally on a standard set of benchmark problems. An LS-SVM has been implemented in C++ using the efficient Atlas library for Linear Algebra [39]. This implementation uses an approximate variant of the LS-SVM kernel machine [32], so as to reduce the computational demands of the experiments. The size of the subset that is used to describe the model is set to 10% of the total data set. Although LS-SVM is only one specific type of kernel machine, all relevant aspects of the models have been kept generalized, so that extension to other types of Kernel Machines (e.g. SVM) is straightforward. Two kernel functions have been considered in these experiments. The first is the RBF kernel function, cf. (12), which is commonly regarded the “default” choice for kernel machines. The second is the polynomial function, cf. (11).

The evolutionary algorithms in the models have been implemented using the *OpenBeagle* framework for EC [13]. The objective function is, as explained, k-fold cross validation with $k = 5$. This value gives a adequate tradeoff between accuracy and computational expenses. For classification problems, the error measure is the normalized classification error; in case of regression problems the *mean-squared-error* (MSE) is used.

Table 1 Basic characteristics of the data sets used in the experiments.

Name	Type	#Samples	#Features	%Positive
Concrete	regression	1005	8	n/a
Diabetes	classification	768	8	65.1%
Housing	regression	506	13	n/a
Reaching 1	regression	1126	4	n/a
Reaching 2	regression	2534	4	n/a
Reaching 3	regression	2737	4	n/a
Wisconsin	classification	449	9	52.6%

6.1 Data Sets

Seven different benchmark data sets have been selected for the empirical validation. Five of these data sets are regression problems, whereas the remaining two are binary classification problems. The data sets *Concrete*, *Diabetes*, *Housing*, and *Wisconsin* are well-known benchmark data sets obtained from the UCI Machine Learning repository [1]. The data sets *Reaching* 1, 2, and 3 are obtained internally from the LiraLab of the University of Genoa[2]. These data sets concern orienting the head of a humanoid robot in the direction of its reaching arm. The features are the traces of 4 arm encoders, whereas the outputs are the corresponding actuator values for 3 head joints. Table 1 shows standard characteristics of the data sets after preprocessing. The exact preprocessing steps that have been performed on the data sets are as follows:

1. All features have been (independently) standardized, i.e. rescaling to zero mean and unit standard deviation.
2. For regression problems, output values have been standardized in the same manner as the features. For classification problems, labels have been set to $+1$ for positive labels and -1 for negative labels.
3. Duplicate entries have been removed from the data sets.
4. The order of the samples in the data set has been randomized.

6.2 Results for Hyperparameter Optimization

The models for hyperparameter optimization have been verified using the following scenario: a very coarse grid search has been performed to identify an interesting region for the parameter ranges for each data set and kernel function. Subsequently,

[2] These data sets can be obtained from `http://eris.liralab.it/wiki/Reaching_Data_Sets`.

a very dense grid search is performed on this region to establish a reference for our models. For the polynomial kernel function, which has two parameters, the degree has been kept fixed at $d = 3$ in order to keep the search computationally tractable. This reference contains the number of evaluations used for grid search[3] and the corresponding minimum error, which serves as the *target* for our models.

The evolutionary models have been used on the same parameter ranges as the grid search. The only exception is that for the polynomial kernel we have *not* kept the degree fixed at $d = 3$; instead it is set within a range of $d = \{1,..,8\}$. This exception is made to investigate the scaling properties of the ES-based approach, i.e. to see whether evolutionary optimization can yield better solutions by optimizing more parameters. The evolutionary search is terminated after the same number of evaluations as used for the grid search.

A comparison of the generalization performance of the grid search and the evolutionary models is shown in Table 2. The overall impression is that all the evolutionary algorithms are able to find competitive solutions. In particular EvoKM$^{\mathrm{GA}}$ shows stable performance, as it finds equal or better solutions for all of the data sets. The only minor exception is the Diabetes data set, for which it finds solutions that are only marginally worse than those found using grid search. Another observation is that for the majority of the data sets the inclusion of the degree of the polynomial kernel indeed decreases the generalization error. This suggests that the methods scale well with the number of parameters and, moreover, that the extra degree of freedom is used to decrease the error. Furthermore, EvoKM$^{\mathrm{ES}}$ and EvoKM$^{\mathrm{GAflt}}$ perform worse than that of the GA-based model on this real-valued optimization problem, suggesting that real-valued chromosomes are not necessarily beneficial for hyperparameter optimization.

More interesting than the optimal solutions is the rate of convergence of the various methods. This has been analyzed by considering the number of evaluations that were needed to reach an error that is close to the target, cf. Table 3. These results confirm the previous observation that EvoKM$^{\mathrm{GA}}$ outperforms the two other models in most situations. The GA method converges to the target error in only a fraction of the number of evaluations used for grid search, with the exception of the Diabetes data set. Furthermore, in almost all situations, it is able to find solutions within a range of 5% of the target within the first 100 evaluations.

The ES and GAflt methods converge slower than EvoKM$^{\mathrm{GA}}$, although EvoKM$^{\mathrm{ES}}$ outperforms the others on a number of regression data sets. Conversely, it performs much worse on the Wisconsin classification data set. One of the reasons for this behavior is that ES uses the mutated offspring to sample the proximity of the parent individuals. This information is then used to find a direction in which the error is decreasing, in a manner similar to gradient descent or pattern search. The difficulty with classification problems is that the error surface incorporates plateaus. Offspring individuals in the proximity of a parent are thus likely to have an identical fitness score and the algorithm will perform a random search on the plateau. Smoothness of the fitness landscape may be regarded as a prerequisite to efficient optimization

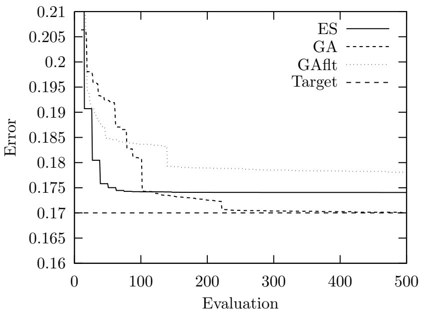
[3] Note that the number of evaluations directly translates into time, as solving the LS-SVM problem is independent of the chosen parameters.

Table 2 Comparison of the minimum errors of grid search and the evolutionary optimization methods. Note that the results of the latter are averages over 25 runs.

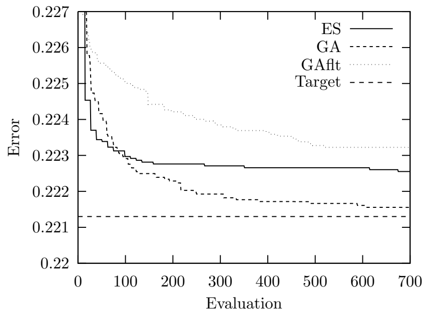
		Grid Search		*EvoKM*^{ES}	*EvoKM*^{GA}	*EvoKM*^{GAflt}
Name	**Kernel**	ε_{min}	**Eval.**	$\bar{\varepsilon}_{min}$	$\bar{\varepsilon}_{min}$	$\bar{\varepsilon}_{min}$
Concrete	RBF	0.1607	1221	0.1591 ± 0.0000	0.1590 ± 0.0000	0.1590 ± 0.0000
	Poly.	0.1700	899	0.1741 ± 0.0000	0.1698 ± 0.0001	0.1778 ± 0.0235
Diabetes	RBF	0.2200	621	0.2201 ± 0.0004	0.2202 ± 0.0006	0.2207 ± 0.0015
	Poly.	0.2213	777	0.2226 ± 0.0003	0.2215 ± 0.0012	0.2231 ± 0.0016
Housing	RBF	0.1676	2793	0.1674 ± 0.0000	0.1674 ± 0.0000	0.1674 ± 0.0000
	Poly.	0.1675	1739	0.1641 ± 0.0016	0.1661 ± 0.0015	0.1646 ± 0.0022
Reaching 1	RBF	0.0683	3185	0.0683 ± 0.0000	0.0683 ± 0.0000	0.0683 ± 0.0000
	Poly.	0.0720	1517	0.0670 ± 0.0002	0.0670 ± 0.0001	0.0677 ± 0.0029
Reaching 2	RBF	0.0042	561	0.0042 ± 0.0000	0.0042 ± 0.0000	0.0042 ± 0.0000
	Poly.	0.0063	399	0.0045 ± 0.0004	0.0043 ± 0.0001	0.0045 ± 0.0003
Reaching 3	RBF	0.0019	561	0.0019 ± 0.0000	0.0019 ± 0.0000	0.0019 ± 0.0000
	Poly.	0.0032	399	0.0022 ± 0.0001	0.0021 ± 0.0001	0.0023 ± 0.0003
Wisconsin	RBF	0.0423	3185	0.0467 ± 0.0025	0.0423 ± 0.0000	0.0432 ± 0.0017
	Poly.	0.0401	2337	0.0433 ± 0.0031	0.0400 ± 0.0017	0.0424 ± 0.0017

Table 3 Comparison of the convergence of the evolutionary models. The column *ETT* (*Evaluations To Target*) denotes the number of evaluations that the average run needs to reach the target error. Analogously, the column $ETT_{5\%}$ denotes the number of evaluations needed to reach an error that is at most 5% higher than the target error.

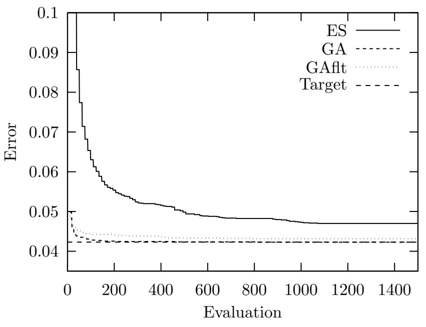
		Grid Search		*EvoKM*^{ES}		*EvoKM*^{GA}		*EvoKM*^{GAflt}	
Name	**Kernel**	ε_{min}	**Eval.**	**ETT**	**ETT**$_{5\%}$	**ETT**	**ETT**$_{5\%}$	**ETT**	**ETT**$_{5\%}$
Concrete	RBF	0.1607	1221	243	135	98	79	274	134
	Poly.	0.1700	899	>899	39	513	102	> 899	285
Diabetes	RBF	0.2200	621	>621	3	>621	10	>621	10
	Poly.	0.2213	777	>777	3	>777	10	>777	10
Housing	RBF	0.1676	2793	267	123	306	64	333	135
	Poly.	0.1675	1739	291	27	550	19	558	39
Reaching 1	RBF	0.0683	3185	435	207	165	35	446	84
	Poly.	0.0720	1517	39	15	28	19	18	10
Reaching 2	RBF	0.0042	561	63	51	128	71	237	136
	Poly.	0.0063	399	39	39	44	44	82	44
Reaching 3	RBF	0.0019	561	75	51	183	88	278	130
	Poly.	0.0032	399	39	39	28	28	65	65
Wisconsin	RBF	0.0423	3185	>3185	>3185	802	28	>3185	80
	Poly.	0.0401	2337	>2337	>2337	2158	270	>2337	>2337



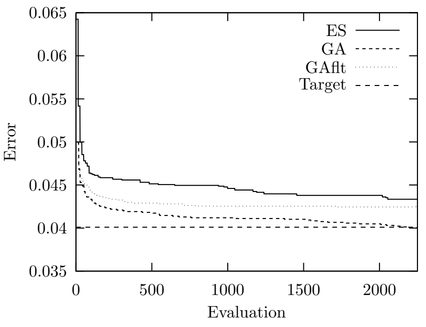
(a) Concrete + Poly.



(b) Diabetes + Poly.



(c) Wisconsin + RBF



(d) Wisconsin + Poly.

Fig. 4 Convergence of the various optimization methods in several problematic combinations of data sets and kernels.

using ES [3]. The situation is somewhat similar for EvoKM$^{\mathrm{GAflt}}$, as this model also incorporates a random perturbation operator for mutation. However, this model has a larger population size and a recombination operator, which can "diversify" the population when progress is ceased on a plateau.

The problematic behavior of EvoKM$^{\mathrm{ES}}$ can be verified in the error convergences depicted in Fig. 4. Albeit the ES method shows a steep initial convergence, the search in these situations stagnates, indicating a random search. Further, in Figs. 4(c) and (d) it can be seen that EvoKM$^{\mathrm{ES}}$ has a considerably higher initial position. This can be attributed to the smaller initial population size, as these individuals are used as the starting points for the search. Additionally, the individuals in EvoKM$^{\mathrm{ES}}$ are initialized near the center of the range, in contrast to the two other methods. We have verified that, for this data set only, the results of EvoKM$^{\mathrm{ES}}$ can be improved by initializing the individuals uniformly over the search space, as is done in the other two models.

Inspection of the solutions confirms the observation that all the evolutionary models produce heterogeneous "optimal" solutions. This is not necessarily problematic, given that variance in the quality of the solutions is limited. Further, although the presented results give some insight regarding the performance of various evolutionary algorithms, it must be taken into account that there is a variety of parameters

Table 4 The minimum errors as obtained with EvoKM$^{\mathrm{GP}}$. Note that $\bar{\varepsilon}_{min}$ indicates the average minimum error over 10 runs, whereas ε_{min} indicates the absolute minimum error.

	Grid Search	*EvoKM^{GP}*	
Name	ε_{min}	$\bar{\varepsilon}_{min}$	ε_{min}
Concrete	0.1607	0.1513 $\pm$ 0.0010	0.1490
Diabetes	0.2200	0.2176 $\pm$ 0.0032	0.2096
Housing	0.1675	0.1633 $\pm$ 0.0006	0.1620
Reaching 1	0.0683	0.0592 $\pm$ 0.0004	0.0587
Reaching 2	0.0042	0.0038 $\pm$ 0.0000	0.0037
Reaching 3	0.0019	0.0018 $\pm$ 0.0000	0.0018
Wisconsin	0.0401	0.0358 $\pm$ 0.0012	0.0333

and operators – in particular for EvoKM$^{\mathrm{GA}}$ and EvoKM$^{\mathrm{GAflt}}$ – that influence the speed of convergence. It is likely that additional fine-tuning of these parameters can improve the performance of these models.

6.3 Results for EvoKM^{GP}

The results from grid search have also been used as a performance benchmark for EvoKM$^{\mathrm{GP}}$. However, for this model we consider only the quality of the solution and ignore the temporal aspects (i.e. number of evaluations). The minimum errors of both grid search and EvoKM$^{\mathrm{GP}}$are shown in Table 4. It can be observed that EvoKM$^{\mathrm{GP}}$ increases the generalization performance for all data sets. However, the minimum errors are only marginally lower than those obtained by grid search. This indicates that the combined kernel functions perform only slightly better than singular kernel functions.

It is difficult to provide strict interpretations of this result, since not finding any combined kernel functions that drastically improves the generalization performance does not necessarily mean that they will not exist at all. This relates directly to the difficulty of finding good configurations for the GP method, as seen with the GA models as well. There are many parameters that need to be set and one has to find a suitable evolver model (i.e. the set of individual altering operators and their order). Unfortunately, there is no structured approach for optimizing the configuration. Therefore, it remains mostly a task that has to be solved using loose heuristics or even intuition. This problem is particularly evident in this GP context, as the computational demand does not allow for an empirical verification of multiple possible configurations, as was done for the ES and GA models[4].

[4] The experiments that we presented for EvoKM$^{\mathrm{GP}}$ need more than half a year of CPU time on a Pentium 4 class computer running at 3 GHz.

7 Conclusions and Future Work

Two approaches for the evolutionary optimization of LS-SVM have been presented. The distinction is that the first aims to find optimal hyperparameters more efficiently than traditional methods (i.e. grid search) and the second aims to increase the generalization performance by means of combined kernel functions. The models for the first approach are based on ES, GA, and GA with a floating point representation. In particular the standard GA model has shown to be an efficient and generalized method for performing hyperparameter optimization for LS-SVM. It was able to find solutions comparable to optimal grid search solutions in only a fraction of the computational demands. Furthermore, the method scales well with the number of parameters. The ES and floating point GA models performed worse than GA, although they are still preferable to grid search for regression problems. Classification problems, on the other hand, are more challenging particularly for the ES model, as the error surface is discontinuous. ES uses the offspring individuals to sample the neighborhood in order to find a direction that minimizes the error. The plateaus found in the error surface of classification problem interfere with this strategy, as offspring are likely to have a fitness that is identical to that of the parent. This problem may be avoided by using the squared error loss function also for classification problems, such that the error surface becomes continuous. Further, the performance of all models may be improved upon by fine-tuning the variety of parameters. In future work, it would be interesting to compare the evolutionary algorithms with various gradient descent methods in terms of solution quality and convergence rate.

The Genetic Programming approach for the generation and selection of kernel functions increases the generalization performance of the Kernel Machine only marginally. This suggests that combined kernel functions may not improve the performance as much as one may expect. In most circumstances, this slight improvement will not justify the high computational demands of this model. The fact that we have not found kernel functions that considerably improve on the generalization performance does not necessarily mean that such kernel functions will not exist at all. The configuration of GP, in terms of the evolver model and parameters, influences to a great extent the results. However, the numerous options and the high computational demand make it very difficult to find an optimal configuration for our model. It is worth investigating whether more advanced variants of GP and further tuning of the configuration can improve the presented results.

Acknowledgment

This study has partially been funded by EU projects *RobotCub* (IST-004370) and *CONTACT* (NEST-5010). The authors gratefully acknowledge Francesco Orabona for his constructive comments, and Francesco Nori and Lorenzo Natale for supplying the Reaching data sets.

References

1. Arthur Asuncion and David J. Newman. UCI machine learning repository, 2007.
2. Hans-Georg Beyer. *The theory of evolution strategies*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.
3. Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies a comprehensive introduction. *Natural Computing: an international journal*, 1(1):3–52, 2002.
4. Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
5. Olivier Chapelle, Vladimir N. Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1–3):131–159, 2002.
6. Peng-Wei Chen, Jung-Ying Wang, and Hahn-Ming Lee. Model selection of svms using ga approach. *Proceedings of the 2004 IEEE International Joint Conference on Neural Networks*, 3(2):2035–2040, July 2004.
7. Zheng Chunhong and Jiao Licheng. Automatic parameters selection for svm based on ga. In *WCICA 2004: Fifth World Congress on Intelligent Control and Automation*, volume 2, pages 1869–1872, June 2004.
8. Laura Dioşan and Mihai Oltean. Evolving kernel function for support vector machines. In Cagnoni C., editor, *The 17th European Conference on Artificial Intelligence, Evolutionary Computation Workshop*, pages 11–16, 2006.
9. Laura Dioşan, Mihai Oltean, Alexandrina Rogozan, and Jean Pierre Pecuchet. Improving svm performance using a linear combination of kernels. In *ICANNGA '07: International Conference on Adaptive and Natural Computing Algorithms*, number 4432 in LNCS, pages 218–227. Springer, 2007.
10. Larry J. Eshelman and J. David Schaffer. Real–coded genetic algorithms and interval-schemata. In L. Darrell Whitley, editor, *Proceedings of the Second Workshop on Foundations of Genetic Algorithms*, pages 187–202, San Mateo, 1993. Morgan Kaufmann.
11. Frauke Friedrichs and Christian Igel. Evolutionary tuning of multiple svm parameters. In *ESANN 2004: Proceedings of the 12th European Symposium on Artificial Neural Networks*, pages 519–524, April 2004.
12. Holger Fröhlich, Olivier Chapelle, and Bernhard Schölkopf. Feature selection for support vector machines by means of genetic algorithms. In *ICTAI '03: Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, page 142, Washington, DC, USA, 2003. IEEE Computer Society.
13. Christian Gagné and Marc Parizeau. Genericity in evolutionary computation software tools: Principles and case study. *International Journal on Artificial Intelligence Tools*, 15(2):173–194, April 2006. 22 pages.
14. Christian Gagné, Marc Schoenauer, Michele Sebag, and Marco Tomassini. Genetic programming for kernel-based learning with co-evolving subsets selection. In *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *LNCS*, pages 1008–1017, Reykjavik, Iceland, September 2006. Springer-Verlag.
15. John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
16. Tom Howley and Michael G. Madden. The genetic kernel support vector machine: Description and evaluation. *Artificial Intelligence Review*, 24(3-4):379–395, 2005.
17. Chin-Chia Hsu, Chih-Hung Wu, Shih-Chien Chen, and Kang-Lin Peng. Dynamically optimizing parameters in support vector regression: An application of electricity load forecasting. In *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 30.3, Washington, DC, USA, 2006. IEEE Computer Society.
18. Cheng-Lung Huang and Chieh-Jen Wang. A ga-based feature selection and parameters optimization for support vector machines. *Expert Systems with Applications*, 31(2):231–240, 2006.
19. Jaz Kandola, John Shawe-Taylor, and Nello Cristianini. Optimizing kernel alignment over combinations of kernels. Technical Report 121, Department of Computer Science, Royal Holloway, University of London, UK, 2002.

20. S. Sathiya Keerthi, Vikas Sindhwani, and Olivier Chapelle. An efficient method for gradient-based adaptation of hyperparameters in svm models. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 673–680. MIT Press, Cambridge, MA, USA, 2007.
21. Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*, pages 1137–1145, 1995.
22. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
23. Gert R. G. Lanckriet, Nello Cristianini, Peter L. Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.
24. Wan-Jui Lee, Sergey Verzakov, and Robert P. W. Duin. Kernel combination versus classifier combination. In *MCS 2007: Proceedings of the 7th International Workshop on Multiple Classifier Systems*, pages 22–31, May 2007.
25. Stefan Lessmann, Robert Stahlbock, and Sven F. Crone. Genetic algorithms for support vector machine model selection. In *IJCNN '06: International Joint Conference on Neural Networks*, pages 3063–3069. IEEE Press, July 2006.
26. Sung-Hwan Min, Jumin Lee, and Ingoo Han. Hybrid genetic algorithms and support vector machines for bankruptcy prediction. *Expert Systems with Applications*, 31(3):652–660, 2006.
27. Michinari Momma and Kristin P. Bennett. A pattern search method for model selection of support vector regression. In *Proceedings of the Second SIAM International Conference on Data Mining*. SIAM, April 2002.
28. David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.
29. Syng-Yup Ohn, Ha-Nam Nguyen, Dong Seong Kim, and Jong Sou Park. Determining optimal decision model for support vector machine by genetic algorithm. In *CIS 2004: First International Symposium on Computational and Information Science*, pages 895–902, Shanghai, China, December 2004.
30. Cheng S. Ong, Alexander J. Smola, and Robert C. Williamson. Learning the kernel with hyperkernels. *Journal of Machine Learning Research*, 6:1043–1071, 2005.
31. Tanasanee Phienthrakul and Boonserm Kijsirikul. Evolutionary strategies for multi-scale radial basis function kernels in support vector machines. In *GECCO '05: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 905–911, New York, NY, USA, 2005. ACM Press.
32. Ryan Rifkin, Gene Yeo, and Tomaso Poggio. Regularized least squares classification. In *Advances in Learning Theory: Methods, Model and Applications*, volume 190, pages 131–154, Amsterdam, 2003. VIOS Press.
33. Sergio A. Rojas and Delmiro Fernandez-Reyes. Adapting multiple kernel parameters for support vector machines using genetic algorithms. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 626–631, Edinburgh, Scotland, UK, September 2005. IEEE Press.
34. Hans-Paul Schwefel. Collective phenomena in evolutionary systems. In *Problems of Constancy and Change – The Complementarity of Systems Approaches to Complexity*, volume 2, pages 1025–1033. International Society for General System Research, 1987.
35. John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, June 2004.
36. Jian-Tao Sun, Ben-Yu Zhang, Zheng Chen, Yu-Chang Lu, Chun-Yi Shi, and Wei-Ying Ma. Ge-cko: A method to optimize composite kernels for web page classification. In *WI '04: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, pages 299–305, Washington, DC, USA, 2004. IEEE Computer Society.
37. Johan A. K. Suykens, Tony Van Gestel, Jos De Brabanter, Bart De Moor, and Joost Vandewalle. *Least Squares Support Vector Machines*. World Scientific Publishing Co., Pte, Ltd., Singapore, 2002.
38. Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

39. R. Clint Whaley and Antoine Petitet. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience*, 35(2):101–121, February 2005.
40. Darrell Whitley. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, 43(14):817–831, 2001.
41. Darrell Whitley, Marc Richards, Ross Beveridge, and Andre' da Motta Salles Barreto. Alternative evolutionary algorithms for evolving programs: evolution strategies and steady state gp. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 919–926, New York, NY, USA, 2006. ACM Press.
42. Chih-Hung Wu, Gwo-Hshiung Tzeng, Yeong-Jia Goo, and Wen-Chang Fang. A real-valued genetic algorithm to optimize the parameters of support vector machine for predicting bankruptcy. *Expert Systems with Applications*, 32(2):397–408, 2007.