Appl Intell (2012) 36:280–294
DOI 10.1007/s10489-010-0260-1

Improving classification performance of Support Vector Machine by genetically optimising kernel shape and hyper-parameters

Laura Dioşan · Alexandrina Rogozan · Jean-Pierre Pecuchet

Published online: 27 October 2010

Abstract Support Vector Machines (*SVM*s) deliver state-of-the-art performance in real-world applications and are now established as one of the standard tools for machine learning and data mining. A key problem of these methods is how to choose an optimal kernel and how to optimise its parameters. The real-world applications have also emphasised the need to consider a combination of kernels—a multiple kernel—in order to boost the classification accuracy by adapting the kernel to the characteristics of heterogeneous data. This combination could be linear or non-linear, weighted or un-weighted. Several approaches have been already proposed to find a linear weighted kernel combination and to optimise its parameters together with the *SVM* parameters, but no approach has tried to optimise a non-linear weighted combination. Therefore, our goal is to automatically generate and adapt a kernel combination (linear or non-linear, weighted or un-weighted, according to the data) and to optimise both the kernel parameters and *SVM* parameters by evolutionary means in a unified framework. We will denote our combination as a kernel of kernels (*KoK*). Numerical experiments show that the *SVM* algorithm, involving the evolutionary kernel of kernels (*eKoK*) we propose, performs better than well-known classic kernels whose parameters were optimised and a state of the art convex linear and an evolutionary linear, respectively, kernel combinations. These results emphasise the fact that the *SVM* algorithm could require a non-linear weighted combination of kernels.

L. Dioşan · A. Rogozan · J.-P. Pecuchet
Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes, EA 4108, Institut National des Sciences Appliquées, Rouen, France

A. Rogozan
e-mail: arogozan@insa-rouen.fr

J.-P. Pecuchet
e-mail: pecuchet@insa-rouen.fr

L. Dioşan (✉)
Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
e-mail: lauras@cs.ubbcluj.ro

Keywords Classification problems · Kernel of kernels · Multiple kernel · *SVM* · Hyper-parameters optimisation · Hybrid model · Genetic programming

1 Introduction

As a broad subfield of artificial intelligence, machine learning is concerned with the design and development of algorithms and techniques that allow computers to "learn". Herbert Simon has provided a very simple, but eloquent definition: "Learning is any process by which a system improves performance from experience." [49]. The general problem of machine learning is to search a, usually very large, space of potential hypotheses to determine the one that will best fit the data and any prior knowledge. There are many learning algorithms today and their performances (estimated by different measures, e.g. classification accuracy, solution correctness, solution quality or speed of performance) are related not only to the problem to be solved, but also to their parameters. Therefore, the best results can be achieved only by identifying the optimal values of these parameters. Although this is a very complex task, different optimisation methods have been developed in order to optimise the parameters of Machine Learning algorithms.

In this context, evolutionary computations have been theoretically and empirically proven to be robust for searching solutions in complex spaces and have been widely used in optimisation, training neural networks, estimating parameters in system identification or adaptive control applications [11, 38]. Evolutionary algorithms form a subset of evolutionary computation in that they generally only involve techniques implementing mechanisms inspired by biological evolution such as reproduction, mutation, recombination, natural selection and survival of the fittest. Candidate solutions to the optimisation problem play the role of individuals in a population, and the cost function determines the environment within which the solutions "live". Evolution of the population then takes place after the repeated application of the above operators.

In 1995, Support Vector Machines (*SVM*s) marked the beginning of a new era in the paradigm of learning from examples. Rooted to the Statistical Learning Theory and the Structural Risk Minimisation principle developed by Vladimir Vapnik at AT&T in 1963 [56], *SVM*s quickly gained attention from the Machine Learning community due to a number of theoretical and computational merits. The main idea is to use a linear-separating hyper-plane to classify a set of items. In 1995 Cortes and Vapnik [13] have suggested a modified maximum margin idea that allows for mislabelled examples. Boser et al. (1992) [5] constructed the *SVM*s for the non-linear data by involving the kernel function. The kernel functions map the input vectors into a very high-dimensional space, possibly of infinite dimension, where the linear separation between items is more likely. This process amounts to a non-linear separation in the original input space. Hence, the complexity of the achieved boundaries depends on the nature and the properties of the used kernel.

Two key elements in the implementation of *SVM* are the techniques of mathematical programming and kernel functions. The parameters are found by solving a quadratic programming problem with linear equality and inequality constraints; rather than by solving a non-convex, unconstrained optimisation problem. The flexibility of kernel functions allows the *SVM* to search a wide variety of hypothesis spaces.

To date, various methods have been proposed to optimise the hyper-parameters of an *SVM* algorithm that uses a particular kernel. However, it was shown that any classical kernel achieves good enough performances for some classification problems [7, 10]. In real world problems, especially when the data is heterogeneous, engineering an appropriate kernel becomes the major part of the modelling process. In this context, an original idea has been recently proposed: to learn the expression of a new kernel function from the problem data by using an evolutionary approach that combines the basic elements of a kernel function (the mathematical operators and the input vectors) [18, 29].

Furthermore, rather than to design a kernel from scratch (by combining the input vectors through different mathematical operations), another novel idea was to generate a multiple kernel (*MK*) function as a combination of classic kernels [17, 39, 45, 50]. It was shown that for complex classification problems, an *MK* improves the performance of *SVM* classifier by adapting better to the characteristics of the data. In this context, several questions arise concerning an *MK*:

- Which is the most efficient combination of kernels: a linear or a non-linear one?
- Is it necessary to consider a weighted or un-weighted combination of kernels?
- Which are the kernels that have to be considered for the most efficient combination: different classic kernels and/or several instances of the same kernel, but with different parameters?
- How to optimise the hyper-parameters of an *MK*-based *SVM* algorithm?

Our paper, through the state of the art and the proposed solution, tries to answer these questions. Therefore, we choose to use the evolutionary framework in order to adapt the expression of a kernel combination and its parameters for several classification problems.

Evolutionary algorithms are a class of probabilistic search algorithms that emulate natural evolutionary process. In this paper we propose a framework to design an evolutionary kernel that could be a linear or a non-linear, a weighted or un-weighted combination of kernels (depending and adapted of/to the problem than must be solved). We will denote our kernel combination as a kernel of kernels (*KoK*). The best (adapted) *KoK* is learnt by the algorithm itself by using the data of a particular problem. Note that our model is the first one that deals with both non-linear *MK* expression and weighting coefficients optimisation for a *KoK*. For this aim the Genetic Programming (*GP*) technique [37] is combined with an *SVM* algorithm [48, 56] within a two-level hybrid model, since hybridisation seems to improve the performance of solving classification problems [58]. The prosed model simultaneously tackles two problems: to find the most efficient expression of the *KoK* function and to optimise the *SVM* hyper-parameters. These two objectives are achieved simultaneously because each *GP* chromosome encodes both the shape of a *KoK* and its parameters (the individual kernels and their parameters). After an iterative process, which runs more generations, an optimal evolutionary kernel of kernels (*eKoK*) is provided.

The *GP*-based *KoK* has the added advantage that its expression needs not to be chosen *a priori*. Furthermore, it is adapted to the problem to be solved, allowing for automatic discovering of a befitting functional form. The *eKoK*s proposed is this paper are compared not only with several well-known classic kernels, but also with a convex linear multiple kernel [39] and with an evolved linear multiple kernel [17]. The numerical experiments show that our hybrid model is able to discover *KoK*s that are more efficient and their optimal parameters on the considered data sets.

The paper is organised as follows: Sect. 2 outlines the theory behind *SVM* classifiers giving a particular emphasis to the kernel functions. An overview of the related work in the field of optimisation methods for the hyper-parameters and kernel functions of an *SVM* algorithm is presented in Sect. 3. Section 4 describes the new hybrid model proposed in order to evolve *KoK*s. This is followed by Sect. 5 where the results of the experiments are presented and discussed. Finally, Sect. 6 concludes the paper.

2 Support Vector Machine

2.1 Generalities

Initially, *SVM* algorithm has been proposed in order to solve binary classification problems [56]. Later, these algorithms have been generalised for multi-classes problems [14]. Consequently, we will explain the theory behind *SVM* only on binary-labelled data.

Suppose the training data has the following form: $D = (x_i, y_i)_{i=\overline{1,m}}$, where $x_i \in \Re^d$ represents an input vector and each y_i, $y_i \in \{-1, +1\}$, the output label associated to the item x_i. *SVM* algorithm maps the input vectors to a higher dimensional space where a maximal separating hyper-plane is constructed [56]. The main idea of *SVM* implies to minimise the norm of the weight vector (w in (1)) under the constraint that the training items of different classes belong to opposite sides of the separating hyper-plane. Since $y_i \in \{-1, +1\}$ we can formulate this constraint as:

$$y_i(w^T x + b) \geq 1, \quad \forall i \in \{1, 2, \dots, m\}, \tag{1}$$

where v^T represent the transpose of v, the primal decision variables w and b define the separating hyper-plane.

The items that satisfy (1) with equality are called support vectors since they define the resulting maximum-margin hyper-planes. To account for misclassification, e.g. items that do not satisfy (1), the soft margin formulation of *SVM* has introduced some slack variables $\xi_i \in \Re$ [13].

Moreover, the separation surface has to be nonlinear in many classification problems. *SVM* was extended to handle nonlinear separation surfaces by using a feature function $\phi(x)$. The *SVM* extension to nonlinear datasets is based on mapping the input variables into a feature space $\mathcal{F}$ of a higher dimension and then performing a linear classification in that higher dimensional space. The important property of this new space is that the data set mapped by ϕ might become linearly separable if an appropriate feature function is used, even when that data set is not linearly separable in the original space.

Hence, to construct a maximal margin classifier one has to solve the convex quadratic programming problem encoded by (2), which is the primal formulation of it:

$$\begin{aligned} &\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i \\ &\text{subject to:} \quad y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \\ &\qquad\qquad\quad \xi_i \geq 0, \quad \forall i \in \{1, 2, \dots, m\}. \end{aligned} \tag{2}$$

The coefficient C (usually called *penalty error* or *regularisation parameter*) is a tuning parameter that controls the trade off between maximising the margin and classifying without error. Larger values of C might lead to linear functions with smaller margin, allowing to classify more examples correctly with strong confidence. A proper choice of this parameter is crucial for *SVM* to achieve good classification performance. We will see in Sect. 3 how it is possible to optimise its value.

Instead of solving (2) directly, it is a common practice to solve its dual problem, which is described by (3):

$$\begin{aligned} &\max_{a \in \Re^m} \sum_{i=1}^{m} a_i - \frac{1}{2} \sum_{i,j=1}^{m} a_i a_j y_i y_j \phi(x_i)^T \phi(x_j) \\ &\text{subject to} \quad \sum_{i=1}^{m} a_i y_i = 0, \\ &\qquad\qquad\quad 0 \leq a_i \leq C, \quad \forall i \in \{1, 2, \dots, m\}. \end{aligned} \tag{3}$$

In (3), a_i denotes the Lagrange variable for the ith constraint of (2).

The optimal separating hyper-plane $f(x) = w \cdot \phi(x) + b$, where w and b are determined by (2) or (3) is used to classify the un-labelled input data x_k:

$$y_k = \text{sign}\left(\sum_{x_i \in S} a_i \phi(x_i)^T \phi(x_k) + b \right), \tag{4}$$

where S represents the set of support vector items x_i.

We will see in the next section that is more convenient to use a kernel function $K(x, z)$ instead of the dot product $\phi(x)^T \phi(z)$.

2.2 Kernel formalism

The original optimal hyper-plane algorithm proposed by Vapnik in 1963 was a linear classifier [56]. However, in 1992, Boser, Guyon and Vapnik [5] suggested a way to create non-linear classifiers by applying the *kernel trick*. Kernel methods work by mapping the data items into a high-dimensional vector space $\mathcal{F}$, called feature space, where the separating hyper-plane has to be found [5]. This mapping is implicitly defined by specifying an inner product for the feature space via a positive semi-definite kernel function:

$K(x,z)=\phi(x)^T\ \phi(z)$, where $\phi(x)$ and $\phi(z)$ are the transformed data items x and z [47].

The kernels that correspond to a space embedded with a dot product belong to the class of positive definite kernels. This has far-reaching consequences. The positive definite and symmetric kernels verify the Mercer's theorem [41]—a condition that guarantees the convergence of training for discriminant classification algorithms such as *SVM*s. The kernels of this kind can be evaluated efficiently even though they correspond to dot products in infinite dimensional dot product spaces. In such cases, the substitution of the dot product with the kernel function is called the *kernel trick* [5].

In order to obtain an *SVM* classifier with kernels one has to solve the following optimisation problem:

$$\begin{aligned}&\max_{a\in\Re^m}\sum_{i=1}^m a_i-\frac{1}{2}\sum_{i,j=1}^m a_i a_j y_i y_j K(x_i,x_j)\\ &\text{subject to}\quad \sum_{i=1}^m a_i y_i=0,\\ &\qquad\qquad\quad 0\le a_i\le C,\quad \forall i\in\{1,2,\dots,m\}.\end{aligned}\tag{5}$$

In this case, (4) becomes:

$$y_k=\operatorname{sign}\left(\sum_{x_i\in S}a_i K(x_i,x_k)+b\right),\tag{6}$$

where S represents the set of support vector items x_i.

There are a wide choice for a positive definite and symmetric kernel K from (6). The selection of a kernel has to be guided by the problem that must be solved. In what follows, some details will be given about the kernels for vectors and their parameters because they have gained considerable attention in the *SVM* community.

3 Related work

While one of the first feelings about *SVM* algorithm is that it can solve a learning task automatically, it actually remains challenging to apply *SVM*s in a fully automatic manner. Questions regarding the choice of the kernel function and the hyper-parameters values remain largely empirical in the real-world applications. While default setting and parameters are generally useful as a starting point, major improvements can result from careful choosing of an optimal kernel. In this context, three directions of optimisation could be identified in order to improve the performances of an *SVM* algorithm: kernel function optimisation, hyper-parameters optimisation and kernel function together with the hyper-parameters optimisation.

While *SVM* classifiers intrinsically account for a trade off between model complexity and classification accuracy, the performance is still highly dependent on appropriate selection of the penalty error C and kernel parameters. Thus, several methods could be used to optimise the hyper-parameters of an *SVM* classifier. Ideally, we would like to choose the value of the kernel parameters that minimise the true risk of the *SVM* classifier. Unfortunately, since this quantity is not accessible, one has to build estimates or bounds for it.

Table 1 The expression of several classic kernels

Name	Expression	Type
Sigmoid	$K_{Sig}\ (x,z)=\tanh(\sigma x^T\cdot z+r)$	Projective
RBF	$K_{RBF}(x,z)=\exp(-\sigma\lvert x-z\rvert^2)$	Radial
Polynomial	$K_{Pol}\ (x,z)=(x^T\cdot z+coef)^d$	Projective

Cross-validation is a popular technique for estimating the generalisation error and there are several interpretations [59]. Leave-one-out (LOO) method could be viewed as a particular form of k-fold cross-validation in which k is equal to the number of examples (m). In LOO, one example is left out for testing each time, and so the training and testing are repeated m times. In the case of *SVM*, it is not necessary to run the LOO procedure on all m examples. In spite of several strategies to speed up LOO procedure [55] and to optimise *SVM* hyper-parameters, LOO is still too expensive.

For efficiency, it is useful to have simpler estimates of the error that, though crude, are not expensive to compute. Once the *SVM* is learnt with a given set of hyper-parameters, the estimates of the error bounds can be obtained with very little additional work. During the past few years, several such simple bound estimates have been proposed, some of them being the Xi-Alpha bound [33], the generalised approximate cross-validation [59], the approximate span bound [57], the VC bound [57], the radius-margin bound [57] or the quality functional of the kernel [44].

Choosing a suitable kernel function for *SVM*s is a very important step for the learning process. There are few if any systematic techniques to assist in this choice. Until now, different kernels for vectors have been proposed [54]; the most utilised of them by an *SVM* algorithm are listed in Table 1.

One of the first kernels involved in the *SVM* algorithm was the Sigmoid kernel. It was quite popular due to its origin from neural networks [47]. In fact, an *SVM* model using a Sigmoid kernel function was proven to be equivalent to a two-layer, feed-forward neural network [30]. The Sigmoid kernel with a positive gain $(\sigma>0)$ and a negative threshold $(r<0)$ is always positive definite and it is successfully used in practice.

Most popular today are the RBF kernels and the Polynomial kernels. The RBF kernel is one of the most frequently used kernels, thanks to its capacity to generate nonparametric classification functions.

Furthermore, it is traditional to distinguish the kernels of projective type from those of radial type. The projective kernels are based on the scalar product between variables, while the radial ones utilise the norm of the difference between the inputs. The radial kernels are stationary or invariant by translation: $K(x,z) = K_s(x-z)$, whereas the projective kernels utilise a scalar product between the variables: $K(x,z) = K_p(x^T \cdot z)$. The projective kernels belong to the most general class of non-stationary kernels.

It should be noted that the majority of these kernels depend on several parameters. Observe that the smaller the parameter σ (called bandwidth), the more peaked the Gaussians are around the support vectors, and therefore the more complex the decision boundary could be. Larger σ corresponds to a smoother decision boundary. The parameters $coef$ and r could be viewed as offset (or shifting) parameters that control the threshold of the mapping.

To obtain good generalisation, it is also necessary to optimise the hyper-parameters. Their values could dramatically affect the quality of the *SVM* solution. Initially, Vapnik [56] recommended direct setting of the kernel parameters and cost function by experts, based on *a priori* knowledge of the particular data set to be evaluated.

Extensive explorations, such as performing line search for one hyper-parameter or grid search for two hyper-parameters, are frequently applied when such knowledge is unavailable [51]. More elaborated techniques for optimising the *SVM* hyper-parameters are the gradient-based approaches [10, 12, 24]. Keerthi et al. [34] have developed a hyper-parameter tuning approach based on minimising a smooth performance validation function, actually the smoothed k-fold cross validation error, by using non-linear optimisation techniques.

The previous gradient-based optimisation methods are highly efficient. They have, however, some drawbacks and limitations: the objective function has to be differentiable. The score function, which is used to assess the performance of the hyper-parameters (or at least an accurate approximation of this function), has also to be differentiable with respect to all hyper-parameters, which excludes reasonable measures such as the number of support vectors.

The previous approaches required to train the model several times with different hyper-parameter values. Therefore, new methods have been proposed to overcome these problems. Several promising recent approaches [1, 60] are based on solution path algorithms, which can trace the entire solution path as a function of the hyper-parameters (the penalty error C and the kernel parameters) without having to train the model multiple times. Furthermore, the paper [27] argues that it is quite tractable to compute the *SVM* solution for all possible values of the regularisation parameter C.

A new study [3] has proposed to directly tackle the model selection by using out-of-sample testing as an optimisation problem. It seeks a set of hyper-parameters, such that when the optimal training problem is solved for each training set, the loss function over the test sets is minimised. The resulting optimisation problem is thus a bi-level programming problem.

The gradient-based optimisation methods have been deeply investigated and discussed, but several derivative-free optimisation methods have been developed as an alternative to the convex methods when the last ones are not applicable. In [42], a pattern search methodology [28] for hyper-parameters, optimisation has been developed as an alternative to the gradient descent. The parameter optimisation method based on simulated annealing [36] has also been proposed as a stochastic method for traversing *SVM* free parameter space [4, 32]. In [40, 62], a Bayesian method based on Markov chain Monte Carlo was proposed for estimating kernel parameters as well as the regularisation parameter.

The evolutionary algorithms have been utilised to optimise the hyper-parameters of an *SVM* classifier [20, 21, 31] as well. In [20], the single-objective evolution strategies have adapted the *SVM* hyper-parameters to the problem that has to be solved; in [21] a single-objective *GA* has optimised the regularisation parameter C in a discrete range of values. Igel [31] has proposed an improved evolutionary approach for optimising the hyper-parameters. The *SVM* hyper-parameter optimisation has been viewed as a multi-objective optimisation problem, where the model complexity and the training accuracy define two conflicting objectives (e.g., bias vs. variance, capacity vs. empirical risk). Different optimisation criteria have been evaluated.

Note that all the previous approaches deal only with a classic kernel, which is fixed *a priori*. No kernel combination is considered in all these cases, because in the context of an *MK* also the expression of such kernel combination must be optimised.

Only very few approaches deal with both problems of hyper-parameter optimisation and of *MK* learning. Recently, Cristianini et al. [16] and Lanckrict et al. [39] have for the first time proposed methods of selecting the kernel or kernel matrix by optimising the measure of data separation in the feature space. While the authors in [16] use the measure called "alignment" to evaluate the adaptability of a kernel to the data, those in [39] employ the margin or soft margin as the measure of data separation in the feature space.

In [61], an alternate method is also proposed to optimise the kernel function by maximising a class separability criterion in the empirical feature space.

Although we are having libraries of kernels and several methods for optimising the hyper-parameters, it is possible that no one of them can achieve good performances for a particular problem. Therefore, a new kernel function must be constructed. In this context, new principles have appeared to design particular kernels (e.g. string kernel, graph kernel), or even to "learn" kernels from the observed data. Evolutionary methods have been actually used in order to automatically discover, over several generations and by using biological-inspired operations (selection, crossover and mutation), new mathematical expressions for the kernel functions which are suitable for solving a particular classification problem by an *SVM* algorithm [18, 22, 29]. These evolved kernels have been encoded as tree-expressions, representing actually the chromosomes of a *GP* algorithm. The leaves of the kernel-tree contain input vectors (x or z), while the internal nodes contain different operations (scalar or vectorial) which combine the input vectors such as the obtained expression to be a kernel function (positive definite and symmetric).

In spite of the large computational effort, it was shown that these evolved kernels have reached better performances than the classic ones for several problems [18, 22, 29]. Furthermore, these results have encouraged a new idea: if a low-level combination (of basic elements—input vectors and operators) is able to design a kernel that improves the performances of an *SVM* algorithm, then maybe also a high-level combination (of basic kernels instead of input vectors) could improve the classification performances and reduce the computational cost.

The combination of several well-known kernels actually means a better initialisation of a possible kernel than a random combination of input vectors and operations. Furthermore, starting from some kernels instead of basic kernel elements, the search space is some-how reduced (limited) and the optimal kernel could be found faster. This combination of kernels represents actually a so-called multiple kernel. The reader must note that the above low-level kernel evolved by GP technique (starting from the input vectors) [18, 22, 29] could be a multiple kernel if its expression is more complex.

Several multiple kernels have been proposed. References [17, 39, 45, 50] for a better adaptation to the classification problem and due to the kernel formalism which allows different standard kernels to be combined. Basic algebra operations such as addition, multiplication and exponentiation preserve the key properties for a kernel function (the positive definiteness and the symmetry) and thus allows a simple, but powerful algebra of kernels to exist [47, 48]. In the context of *MK*s, two important combinations could be distinguished: linear multiple kernels (*LMK*s) and non-linear multiple kernels (non*LMK*s).

A simple way to achieve a linear *MK* is to consider a weighted sum of kernels:

$$LMK(x,z) = \sum_{q=1}^{NoK} \mu_q K_q(x,z), \tag{7}$$

where $\mu_q \in [0,1]$ with $\sum_{q=1}^{NoK} \mu_q = 1$. The weighting coefficients μ_q could reflect the relative importance of each classic kernel in the final *MK*.

Non linear combination of kernels could be also considered in order to improve the *SVM* performance. A non-linear *MK* could be either a pure (an un-weighted) combination of well-known kernels: $nonLMK(x,z) = K_1(x,z) + K_2(x,z) \times (K_3(x,z) + K_4(x,z))$, like those proposed in [43, 46, 52] or a weighted combination of some standard kernels: $nonLMK(x,z) = \mu_1 K_1(x,z) + \mu_2 K_2(x,z) \times (\mu_3 K_3(x,z) + \mu_4 K_4(x,z))$, undeveloped until now.

In order to find the optimal weights of the standard kernels included in an *LMK* the convex methods [39, 45, 50, 63] and the evolutionary methods [17] have been utilised. Two important differences must be remarked between these models and that we develop in this paper. The previous approaches [17, 39, 45, 50, 63] impose a linear combination of kernels, while the current one allows generating either a linear or a non-linear efficient *KoK*, which is able to capture many aspects displayed by the actual data.

Furthermore, the objective function is different in these models: the *GP* algorithm from the current work optimises the complex expression of a *KoK* (the shape of the kernel expression, the coefficients and the hyper-parameters), while the previous models have optimised only the weighting coefficients (μ_q) of the linear combination and the hyper-parameters.

While these two distinctions are present, nevertheless both MK generating models are capable of combining different parameterised standard kernels and allow, in this way, selecting the best parameters for the actual well-known kernels involved in the combination.

Concerning the non-linear combination of kernels, the evolutionary methods, such as the Genetic Algorithms (*GA*s) [43, 46] or the *GP* technique [52] have been used in order to learn the expression of such *MK* function, but only without scaling or shifting coefficients. Several remarks could be also done regarding the non-linear *MK*s evolved by using a *GA* [46]. The expressions of the *GA*-based *MK* is actually less complex that the expression of our *eKoK*. The freedom degree of *GP*-based model is larger than that of *GA*-based representation. Therefore, the search space of the optimal *MK* in the *GA*-based model is smaller than the search space in the *GP* case. The hybrid model we propose is able to find a more sophisticate kernel combination because of:

- A larger set of operations—the exponential function is also used (the numerical experiments will show that the exponential function is actually involved in the expression of several *eKoK*s). The power function with an integer exponent involved in the *GA*-based model does not appear in our approach because of the tree-based representation of the *KoK*; this representation is able to generate it by itself (in an explicit manner, as a repeated multiplication);
- A more flexible form of the *KoK*'s expression due to the representation and coefficients. The *GP* tree-based rep-

resentation favours it, while the *GA*-based model (actually an array-based model) [43, 46] supposes a sequential access to kernels which does not take into account the priority of the mathematical operators. This access determines a smaller space for the possible combinations of kernels than the space explored in the *eKoK* case. The non-linear *MK*s obtained by the model proposed in [43, 46] represent un-weighted combinations of kernels ($K_1(x,z) + K_2(x,z) \times K_3(x,z)$). Our model allows evolving more general *KoK*s in which the standard kernels, but also some coefficients are involved ($s \times K_1(x,z) \times K_2(x,z) + K_3(x,z) + o$); this option could generate fine-grained solutions.
- A better adaptability of the *KoK* to the data—the *GA*-based model forces at least the polynomial and the ANOVA kernels to appear in the combination. The *GP*-based approach permits the data and their characteristics to chose the classic kernels involved in combination (maybe all the standard kernels or maybe just a few of them).

We could also hook up our model for evolving *KoK*s to the Sullivan's one [52], both models being based on a *GP* representation and optimising the kernel parameters. However, only pure combinations of kernels are actually generated in [52], while our model is able to evolve a more general kernel combination, which combines not only kernels, but also some scaling and/or shifting coefficients. Furthermore and unlike our approach, Sullivan's model did not optimise the penalty error parameter C.

Another important observation regards the composition of such *MK*s. Any coefficients or weights are involved in the expression of the *MK*. To the best of our knowledge, the model we propose in this paper is the first one that combines both the classic kernels and some coefficients in a non-linear combination. While not yet providing complete solution to the problem of *MK* learning, our model seems to be the most complex proposed until now. It is able to answer all questions addressed in the introduction. Furthermore, the model we propose deal also with the problem of parameter optimisation.

Ong et al. [44] have introduced a general class of hyper-kernels allowing automatic relevance determination. They have learn a new kernel by performing the *kernel trick* on the space of kernels (hence the notion of a hyper-kernel) and by defining a quantity analogous to the risk functional, called the quality functional (that measures the badness of the kernel function).

One of the first attempts to propose a kernel machine based on a linear un-weighted *MK* ($\mu_q = 1$ in *LMK*) was in [26]. More recently, Lanckriet et al. [39] have provided a general framework for learning the linear *MK* matrix based on semi-definite programming. Similar to this idea, Bousquet and Herrmann [6] further restrict the class of kernels involved in the *LMK* to the convex hull of the kernel matrices normalised by their trace. The semi-definite programming makes the problem rapidly intractable as the number of learning examples or kernels become large. Therefore Bach et al. [1] have reformulated the *LMK* learning problem and then proposed a Sequential Minimal Optimisation algorithm for medium-scale problems. Another formulations of this problem based on a semi-infinite linear problem have been proposed by Sonnenburg et al. [50] and Rakotomamonjy et al. [45] for *LMK*. A Bayesian hierarchical model for *MK* learning has been also presented in [23]. All these approaches have optimised the coefficients a_i, $i \in \{1, 2, \ldots, m\}$ of the *SVM* algorithm and the weights μ_q, $q \in \{1, 2, \ldots, NoK\}$ of an *LMK*.

Both *MK* shape and hyper-parameters are optimised either by using a *GA* [43, 46] or a *GP* technique [52]. However, the evolved non-linear *MK*s are actually pure combinations of kernels (no weighting coefficients are involved in the *MK*).

The evolutionary model we propose is able to cover all the optimisation problems related to a kernel combination (linear or non-linear, weighted or un-weighted) and hyper-parameters. All these criteria are considered in a unified framework and optimised simultaneously.

4 Evolutionary kernel of kernels (*eKoK*)

4.1 Model architecture

This section describes our approach for automatic design of *KoK*s. This model is a hybrid one: it uses *GP* [37] to construct positive and symmetric functions (*KoK*s), and optimises a fitness function by using an *SVM* classifier (see Fig. 1). A *GP* chromosome provides the analytic expression of such *KoK*. The model we propose actually seeks to replace the expert domain knowledge concerning the design of the *SVM*'s kernel function and the choice of its parameters, with a *GP* algorithm.

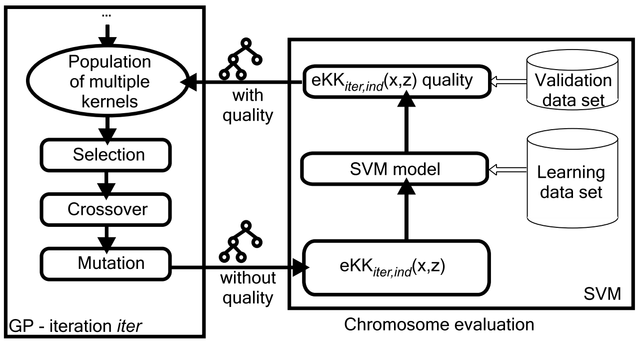

Fig. 1 Sketch of the hybrid approach

Our hybrid model is structured on two levels: a macro level and a micro level. The macro level algorithm is a standard *GP* [37], which is used to evolve the mathematical expression of a *KoK*. The steady-state evolutionary model [53] is involved as an underlying mechanism for the *GP* implementation. A steady state algorithm is much more tolerant of poor offspring than a generational one. This is because in most implementations, the best individuals from a given generation will always be preserved in the next generation, giving themselves another opportunity to be selected for reproduction. The best individuals are therefore given more chances to pass on their successful traits. The *GP* algorithm starts by an initialisation step of creating a random population of individuals (seen as *KoK*s). The following steps are repeated until a given number of iterations are reached: two parents are selected using a binary selection procedure; the parents are recombined in order to obtain an offspring O; the offspring is than considered for the mutation; the new individual O^* (obtained after mutation) replaces the worst individual W in the current population if O^* is better than W.

The micro level algorithm is an *SVM* classifier. It is taken from *LIBSVM* [8] library. The original implementation of the *SVM* algorithm proposed in [8] allows using several well-known kernels (Linear, Polynomial, RBF and Sigmoid—see Table 1). In the numerical experiments, a modified version of this algorithm, which is based on the evolved *KoK* is also used. The quality of each *GP* individual is determined by running the *SVM* algorithm, which uses the *eKoK* encoded in the current chromosome ($KoK_{iter,ind}$ that corresponds to the *ind*th individual from the population during the *iter*th iteration). The accuracy rate estimated by the classifier (on the validation set) represents the fitness of the *GP* chromosome.

4.2 The representation of the *eKoK*

In the model we propose, the *GP* chromosome is a tree encoding the mathematical expression of a *KoK* and its parameters. The tree-based representation of a *KoK* allows for a larger search space of kernel combinations than an array-based representation (like that proposed in [46]—see Sect. 3).

Moreover, the *GP* individual representation is constrained to satisfy the kernel algebra [48] (regarding the positiveness and the symmetry of the Gram matrix required by valid Mercer's kernels). For this purpose, a particular type of *GP* tree is actually used: the leaves contain either a classic parameterised kernel or an ephemeral random constant (viewed as a scaling or a shifting coefficient). Note that a kernel-*GP* tree must contain at least one kernel in its leaves (the number of kernels involved in an *MK* must be greater than or equal to 1), otherwise the obtained expression cannot represent a dot product or a distance between the input vectors x and z. The leaves of the tree form the terminal set (*TS*) and the internal nodes form the function set (*FS*).

For a better adaptation to the classification problem, the terminal set contains not only the classic kernels, but also some ephemeral random constants [37]: $TS = KTS \cup \{o, s\}$, where:

- *KTS*—the terminal set of the standard kernels,
- o—offset (shifting) coefficients that control the threshold of the mapping from the original space into the feature space $\mathcal{F}$ and
- s—scaling (or weighting) coefficients that control the relative influence of the standard kernels in the *eKoK* expression. Both types of coefficients must be positive real values in order to obtain Mercer's kernels.

Some remarks must be made regarding these (scaling or shifting) coefficients. The tree-based representation of a *KoK* is able to generate such coefficients by itself (implicit), but in this case, larger (deeper) trees are required. Therefore, we have chosen to use an explicit representation of a *KoK* with coefficients (the leaves of the *GP* tree could contain such coefficients), which is less computational expensive, hence faster.

Each well-known kernel has associated a set of parameters θ that affect the performance of the *SVM* algorithm. Therefore, more kernels are considered for the TS, but with different parameters. The *RBF* kernel has only a parameter—the bandwidth σ (in this case $\theta = \{\sigma\}$). The Sigmoid kernel has two parameters: the bandwidth or the gain σ and the shifting coefficient r that controls the threshold of the mapping ($\theta = \{\sigma, r\}$). The Polynomial kernel has only a parameter: the degree d ($\theta = \{d\}$)—see Table 1. These kernels will be denoted as parameterised kernels K^θ and different values of θ parameter are actually considered. The *GP* algorithm will provide the most efficient *KoK* expression (in terms of accuracy rates) because it is able to choose and combine the parameterised kernels. Therefore, the purpose of our model is two fold: to discover the most efficient expression of the *KoK* function and to optimise the values of the hyper-parameters.

The function set *FS* contains 3 operations ($FS = \{+, \times, \exp\}$) that preserve the key properties of a Mercer's kernel. The theory of kernel algebra [15] specifies the *power* function also, but this operation (with a natural exponent) can be implicit obtained as a repeated multiplication.

An example of a *GP* chromosome is depicted in Fig. 2. The $FS = \{+, \times, \exp\}$ and $TS = \{K_1^\theta, K_2^\theta, K_3^\theta, o, s\}$ have been used for this chromosome, but only two functions ($+$ and $\times$), two kernels (K_2^θ and K_3^θ), two offset coefficients (o_1 and o_2) and a scaling coefficient (s_1) are actually involved in the expression of *eKoK*.

The *grow method* [2], which is a recursive procedure, is used to initialise a *GP* individual. This initialisation method

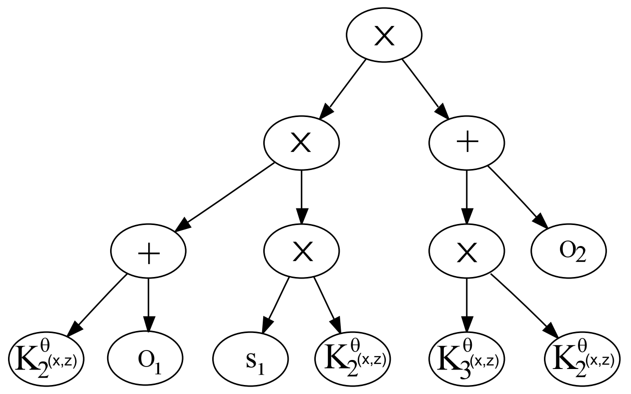

Fig. 2 A *GP* chromosome that encodes the expression of the following *KoK*: $(K_2^\theta(x,z) + o_1) \times s_1 \times K_2^\theta(x,z) \times (K_3^\theta(x,z) \times K_2^\theta(x,z) + o_2)$

is well known in the literature for its robustness. The root of each *GP* tree must be a function from *FS*. If a node contains a function, then its children are initialised either with another function or with a terminal (a kernel or a coefficient). The initialisation process is stopped when is attained a leaf node or at the maximal depth of the tree (the nodes from the last level will be initialised by terminals). In order to obtain a valid KoK, at least one leaf of the *GP* tree has to contain a kernel. Moreover, the maximal depth of a *GP* chromosome has to be large enough in order to assure a sufficient search space for the optimal expression of our evolutionary *KoK*.

4.3 Fitness assignment

The evaluation of the chromosome quality is based on a cross-validation process. Therefore, some information about the data set partitioning must be provided before to describe the fitness assignment process.

The data sample was randomly divided into two sets: a training set (80%)—for model building—and a testing set (20%)—for performance assignment. The training set was then randomly partitioned into learning (2/3) and validation (1/3) parts.

The *SVM* model based on the *eKoK* that is encoded in the current *GP* tree uses the learning subset for training the *SVM* model and the validation subset for classification performance assignment. The quality of an *eKoK* can be measured by the classification accuracy rate estimated on the validation data set. The accuracy rate represents the number of correctly classified items over the total number of items belonging to the validation set. Note that we deal with a maximisation problem: the greater accuracy rate, the better *KoK* is.

Once the *GP* iterations end, the optimal *eKoK*, which corresponds to the best *GP* chromosome is utilised by *SVM* algorithm in order to classify the test items.

4.4 Genetic operations

After two chromosomes are selected from the current population, they are recombined. The crossover is performed in a tree-structure preserving way in order to guarantee the syntactical validity of the offspring: first as a mathematical expression and second as a Mercer's kernel. The proposed model uses the standard cutting-point crossover [37] with the particularity that the offspring has to contain at least one kernel in its leaves. After crossover, the mutation operator is applied. For a *GP*-based *KoK*, a cutting point is randomly chosen: the sub-tree belonging to that point is deleted and a new sub-tree is grown there by applying the same random growth process that was used to generate the initial population. Note that the maximal depth allowed for the *GP* trees limits the growth process. Like in Koza's implementation [37], the mutation operator may generate new constants at any point in a run. As we already said, in *eKoK*'s model, these ephemeral random constants are represented by the scaling and offset coefficients.

5 Experimental validation and discussions

This section reports on the experimental validation of *eKoK*, on a standard set of benchmark problems [19]. These data sets were chosen in order to allow comparisons to the linear *MK*s previously proposed [17, 39] and they are still widely used in the classification community. The hybrid model we proposed is based on TinyGP[1] framework of *GP* algorithm and LIBSVM [8] framework of *SVM* classifier.

All the datasets concern binary-classification problems of different sizes (the number of items and the number of characteristics) and belonging to different domains: medical, economical and geographic fields. A short description of each data set is presented in Table 2. As we already mentioned in Sect. 4.3, each data set has been randomly divided into a training set (80%)—for model building—and a testing set (20%). The training set has been randomly partitioned into learning (2/3) and validation (1/3) parts.

We have chosen to validate our evolved *KoK* on these data sets because our purpose is not only to promote new kernel functions, but also to compare our evolved kernels with those already proposed in the specialised literature [39]. The comparison is performed by tacking into account the results obtained for the first five problems. The last two problems are utilised in order to put in evidence that our approach has considerable promise.

[1] http://cswww.essex.ac.uk/staff/sml/gecco/results/TinyGP/entries/poli/.

Table 2 Description of the data sets

ID	Name	#items	#characteristics	Reference
P_1	ionosphere	351	34	[19]
P_2	breast	683	10	[19]
P_3	heart	270	13	[35]
P_4	a1a	4217	123	[19]
P_5	a2a	2591	123	[19]
P_6	sonar	208	60	[19]
P_7	diabetes	768	8	[19]

5.1 Evolving the *KoK* function

A *KoK* is evolved in this experiment for each problem. In order to evolve this kind of combinations two different terminal set types are used: a terminal set that contains only several standard parameterised kernels *KTS* and a mixed terminal set that contains standard kernels and coefficients $MTS = KTS \cup \{o, s\}$. These coefficients (or ephemeral random constants, by using a *GP* vocabulary) could be either scaling or shifting coefficients. Therefore, the *TS*s actually used in the numerical experiments are:

1. A *TS* composed by several well-known kernels with different parameters

   $$KTS = \{K^{\theta}_{Pol}, K^{\theta}_{RBF}, K^{\theta}_{Sig}\}$$

   where the parameters θ of each standard kernel have been considered in some discrete ranges: for the degree d of the Polynomial kernel 15 values (from 1 to 15) are considered, for the bandwidth σ of the RBF kernel the following values: $\sigma_{qt} = q \cdot 10^t$, $q = \{1, 2, \dots, 9\}$, $t = \{-5, -4, \dots, -1\}$ are considered and for the Sigmoid kernel all the combination between σ_{qt} and r, where $r = 10^u$, $u \in \{-1, 0, 1\}$ are taken into account.
2. A *TS* with different standard kernels and coefficients

   $$MTS = KTS \cup \{o_t, s_p\}.$$

The results found in literature indicate that these discrete spaces of parameters are the most suitable for an efficient classification. The improvements obtained by using a finer discretisation of the parameter space or a continuous space are not relevant (by tacking into account the computational effort that must be performed). Furthermore, the guided search (based on the efficiency of an *MK*) involved by the evolutionary algorithm is able to detect in the discrete space the optimal values of these parameters (and implicit the corresponding kernels).

Several things about the value of coefficients o and s must be remarked. Mercer conditions impose that these coefficients must be positive. The [0, 1] range was suggested in [17, 39] for the coefficients when these values have represented the weights of the individual kernels involved into a linear *MK*. In *eKoK* case there are some scaling and shifting coefficients that could appear or not in the combination. Therefore, several positive intervals have been tested for these coefficients in the numerical experiments, the best of them being [0, 1].

The selection of the kernel parameters has the same importance as the optimisation of the kernel expression. In order to determine good values of these parameters, it is important to search on the right scale. In [60] is suggested a path algorithm for regularisation of C value, but this algorithm can be used only when each of the two classes has the same number of examples. Another method is proposed in [9] proposing as default value for the C parameter is the inverse of the empirical variance s^2 of the data in the feature space:

$$s^2 = \frac{1}{m}\sum_{i=1}^{m} KM_{i,i} - \frac{1}{m^2}\sum_{i=1}^{m}\left(\sum_{j=1}^{m} KM_{ij}\right) \tag{8}$$

from an $m \times m$ kernel matrix KM. This value is actually used in our numerical experiments performed in order to evolve the expression of a *KoK* function.

The steady-state model [53] is used for the *GP* algorithm. A population of 50 individuals is evolved during 50 iterations, which is a reasonable limit to assure the diversity of our *eKoK*s. The binary tournament mechanism is used for chromosome selection. The crossover and mutation operations are performed with 0.8 and 0.3, respectively, probabilities, values that are generally recommended in the specialised literature [25].

The maximal depth of a *GP* tree is limited to 10 levels, which allows encoding till 2^{10} combinations of kernels and coefficients. This maximal depth was fixed by tacking into account the bloat problem (the uncontrolled growth of programs during *GP* runs without (significant) return in terms of fitness [37]). Furthermore, several empirical tests indicated that the efficient kernel-trees do not expand to more than 10 levels. During different runs, various expressions of the *KoK* function have been obtained, all of which have about the same complexity.

Figure 3 depicts the evolution of the *eKoK*'s quality along the number of generations (for all the problems on the vali-

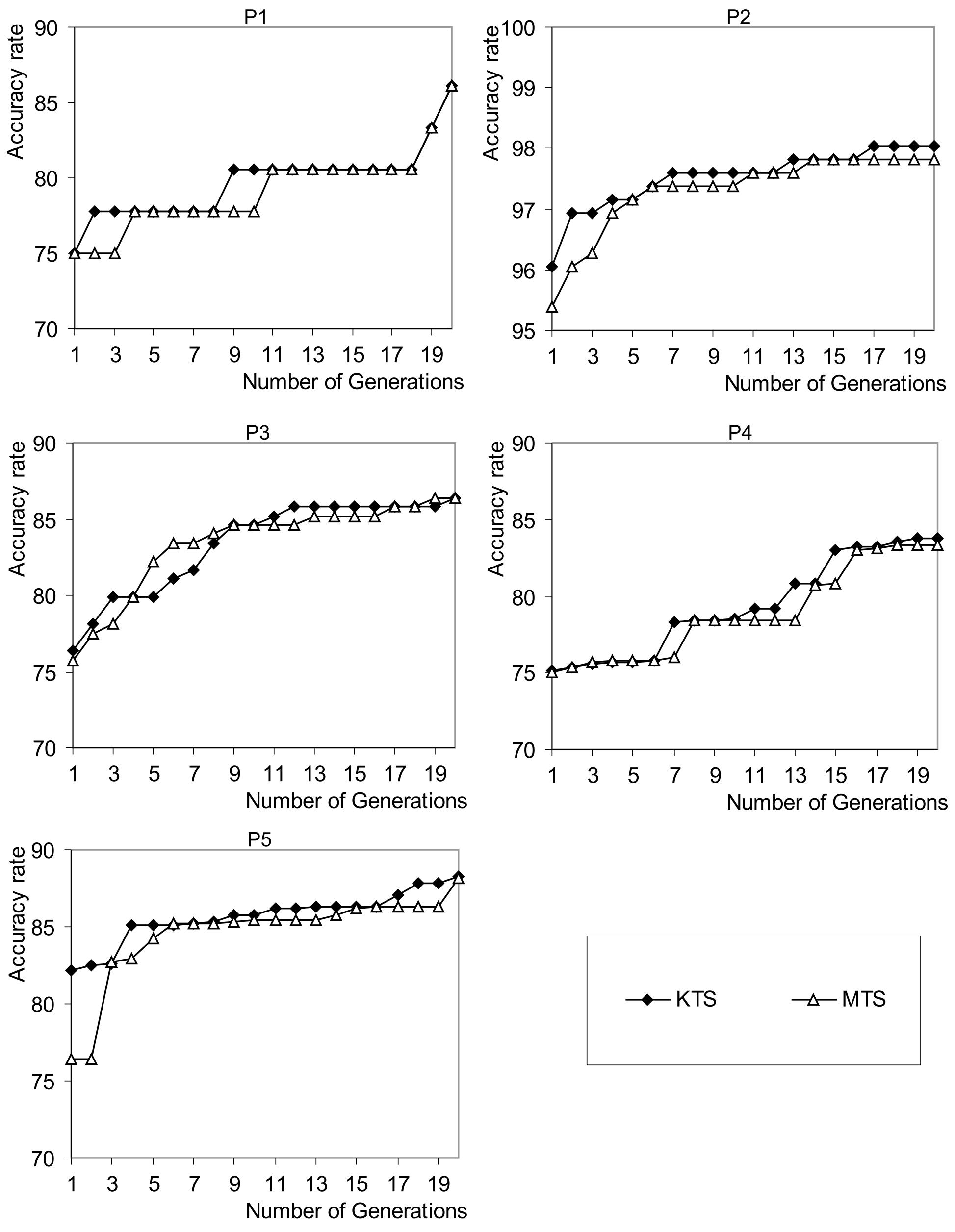

Fig. 3 Fitness (Accuracy Rate—*Acc*) evolution of the *KoK* quality for all the test problems (P_1, P_2, P_3, P_4, P_5) along with the number of generations on the validation data sets. We have tested different TS compositions: *KTS* and *MTS* and for each iteration the best individual (eKoK) is retained

dation data sets). Only the values corresponding to the first 20 generations are depicted in these graphics for a better visualisation. Small improvements can be observed in the chromosomes quality (or in the accuracy rate) after the first 15 GP generations for most of the problems. This aspect is very important and it proves that the proposed model is able to adapt the *KoK* in only a few generations (for instance, by using 3 CPUs, the best *KoK* for problem P_1 is evolved in approximately 60 minutes).

An important remark must be also done regarding the functions included in the evolved expression of a *KoK*. The exponential function is present in several best expression of a *KoK*, proving that the expansion of the function set (from $\{+, \times\}$ in Sullivan's model [52] to $\{+, \times, \exp\}$ in our case) was useful (in terms of the *SVM* performance improvements).

We are interested in studying the performances of the evolved *KoK*s not only on the validation set. For this purpose the best evolved *KoK* (actually, the best GP chromosome from the last iteration of the evolutionary algorithm) is involved again in the SVM algorithm and utilised to classify the test data. Thus, the performances of the best evolved *eKoK*s by using various *TS*s are presented in Table 3: the first two rows contain the accuracy rates (for each problem) estimated by the *SVM* algorithm involving our best evolutionary *KoK*s on the test set (unseen data). Table 3 also presents the

Table 3 The accuracy rate of various kernels. The *first two rows* present the accuracy rates estimated by the *SVM* algorithm embedding the *eKoK*s. The *last three rows* contain the performances of the classic kernels for each test problem

	KTS	*MTS*	K_{pol}	K_{rbf}	K_{sig}
P_1	86.11±1.13	**91.67±0.90**	77.77±1.36	80.55±1.29	66.67±1.54
P_2	97.81±0.13	**98.03±0.13**	97.58±0.14	97.81±0.13	97.81±0.13
P_3	**86.98±0.51**	**86.98±0.51**	85.79±0.53	85.21±0.54	77.91±0.63
P_4	84.27±0.14	**84.38±0.14**	84.26±0.14	83.65±0.14	82.73±0.14
P_5	86.93±0.19	**88.99±0.18**	86.24±0.20	83.49±0.21	84.52±0.21
P_6	**81.25±2.39**	**81.25±2.39**	78.13±2.53	78.13±2.53	78.13±2.53
P_7	80.08±0.30	**81.25±0.29**	79.29±0.31	80.85±0.30	78.90±0.31

performances of three classic kernels for all the test problems (the last three rows). Note that the value of the penalty error C is automatically adapted to the data (cf. (8)) and the other parameters involved in each classic kernel were optimised by grid search in order to achieve the best classification performances. The best parameters of the classic kernels are searched in the ranges already presented for the classic kernels involved in the *KoK*. In this manner we are able to verify if *eKoK*s outperform the optimised standard kernels and, than, to measure the improvements.

The accuracy rate reflects the classification performance of the *SVM* algorithm in a confidence interval. The confidence intervals associated to the performances of the systems must be computed in order to decide if a system outperforms another system. If these intervals are disjoint, then one system outperforms the other ones. A confidence interval of 95% is used in order to perform a statistical examination of the results. Therefore, the probability that the accuracy estimations are not in the confidence interval is 5% (see (9)).

$$\Delta I = 1.96 \times \sqrt{\frac{Acc(100 - Acc)}{N}}\%, \tag{9}$$

where N represents the number of test examples. Therefore, Table 3 displays the corresponding confidence intervals (on the test set of each problem).

The values from Table 3 indicate that the *eKoK*s perform statistically better than the optimised classic kernels in some cases (for problem P_4 there is no statistical difference between *KoK* based on *KTS*, *KoK* based on *MTS* and K_{Pol}).

This is a very important result by taking into account the fact that we compared (*a posteriori*) our *eKoK*s with the best standard kernel for a particular problem. In addition, by taking into account different TS compositions, the results from Table 3 show that the *eKoK*s that contains well-known kernels and coefficients (*MTS*) perform slightly better than the *eKoK*s based only on standard kernels (*KTS*). Therefore, it seems to be efficient to combine kernels with coefficients. Thus, we are tempted to promote the *eKoK* based on *MTS* to the detriment of the *eKoK* based only on kernels (without coefficients).

5.2 Comparison between the evolutionary *KoK*s and the linear *MK*s

Comparison with other *MK*s models is difficult due to different experimental methodologies and databases. Several researchers do not provide information about the partitioning of the database, while for others the databases are not available.

The improvements obtained by the *SVM* classifier which involves our promising *eKoK* based on *MTS* is compared with both the state of the art convex *LMK* [39] and the evolutionary *LMK* already proposed in [17] for the first five problems.

In order to emphasise the improvements obtained by involving a kernel combination in the *SVM* algorithm, an average performance improvement ($\overline{\Delta}$) is computed for each *MK* as the mean of the improvements δ_i for all the problems (the lack of information regarding the data-set partitioning has imposed us to perform only this average comparison). Note that δ_i is the relative difference between the accuracy rate estimated by the *SVM* algorithm with an *MK* (Acc_{MK}) and the accuracy rate estimated by the same *SVM* algorithm, but with a standard kernel (*SK*) for the ith problem:

$$\delta_i = \frac{Acc^i_{MK} - Acc^i_{SK}}{Acc^i_{SK}}, \quad i = \overline{1,5}, \quad \text{and}$$
$$\overline{\Delta} = \frac{\sum_{i=1}^{5} \delta_i}{5}, \tag{10}$$

where *SK* could be one of the considered classic kernels: K_{Pol}, K_{RBF} and K_{Sig} and *MK* could be: *eKoK*—the evolutionary kernel of kernels proposed in this paper based on *MTS*, *eLMK*—the evolutionary linear multiple kernel [17] or *cLMK*—the convex linear multiple kernel [39].

The values of the performance improvements are given in Table 4 and they show that *eKoK*s generally perform better than both the linear *MK*s (convex or evolutionary). This may be because our *eKoK* being more complex and involving the optimal parameters is better adapted to each classification problem than the linear combinations.

Table 4 The average performance improvements for the *MK*s vs. *SK*s. We used *bold letters* to denote the eKoKs that out-perform the eLMK. In addition, *italic letters* indicate that the eKoKs out-perform the cLMKs

$\overline{\Delta}$	K_{Pol}	K_{RBF}	K_{Sig}
eKoK	**7.45%**	***6.84%***	***22.10%***
eLMK	2.00%	3.66%	9.33%
cLMK	8.00%	3.00%	17.66%

In conclusion, the *eKoK* model based on the *MTS* seems to be the most promising one.

5.3 Analysis of the complexity

Even that the kernel matrix are *a priori* computed, the time required to evolve a *KoK* is larger than that from evolutionary linear *MK* or convex linear *MK* cases due to the complexity of kernel function. However, this time is reasonable taking into account that the convex methods are not able to optimise the expression of a non-linear kernel combination and, until now, there are no other solutions for optimising simultaneously the *SVM* hyper-parameters and the *MK* expression.

Furthermore, the optimisation of the hyper-parameters and kernel expression takes place during the learning process of the best adapted *KoK* to the given problem. During the test phase, these optimal values are utilised and no time is needed to tune or to adapt them to the problem.

Moreover, the accuracy rate is utilised to estimate the efficiency of a classification method. However, this function may over fit training data. Sometimes, data contain a lot of noise, and thus if the model fits these noisy data, the learned concept may be wrong. Hence, the kernel combination and the set of hyper-parameters could be validated on many training sets.

6 Conclusion

A new hybrid model has been proposed in order to solve classification problems: a *GP* algorithm combined with an *SVM* classifier for evolving *KoK*s. Several numerical experiments have been performed in order to compare the *eKoK*s to other kernels (classic or evolved, simple or multiple). The numerical results have shown that *eKoK*s perform better not only than the classic kernels, but also than the linear *MK*s (convex or evolutionary *LMK*s). Although the proposed model has a higher computational cost during the learning stage, once the *eKoK*is constructed, the classification stage is as fast as the previous *MK* models.

The main conclusions of this paper can be summarised as follows. The complex (linear or non-linear) kernel functions must include not only different combination of operators ($+$, $\times$, exp) and kernels (Sigmoid, Polynomial, *RBF*), but also some scaling and/or shifting coefficients. The *eKoK*s based on efficient kernels, whose parameters are optimised for the evolved combination of kernels seems to be the most promising approach. The regularisation parameter C of the *SVM*-classifier allows also improving the performance of the classifier. We emphases the fact that, to our knowledge, the approach we propose is the only one capable to achieve these three objectives in the same framework.

7 Future works

We will focus our further work on the validation of *eKoK* model proposed in this paper for large data sets and using multiple data sets for the training stage; this could help to evolve kernels that are more generic. Furthermore, we plan to evolve *KoK*s for future selection tasks and to use them in order to solve classification problems with heterogeneous data. In this way, we should favour the data fusion process.

Acknowledgements The authors thank to anonymous reviewers for their useful suggestions. This research was supported by grant IDEI-543/2007 from CNCSIS, Romania and by project ASICOM, France.

References

1. Bach FR, Thibaux R, Jordan MI (2004) Computing regularization paths for learning multiple kernels. In: NIPS, pp 1–10
2. Banzhaf W (1998) Genetic programming: an introduction: on the automatic evolution of computer programs and its applications
3. Bennett K, Hu J, Ji X, Kunapuli G, Pang J-S (2006) Model selection via bilevel optimization. In: IJCNN'06. International joint conference on neural networks. IEEE Computer Society, Los Alamitos, pp 1922–1929
4. Boardman M, Trappenberg T (2006) A heuristic for free parameter optimization with SVM. In: IJCNN 2006. IEEE, New York, pp 1337–1344
5. Boser BE, Guyon I, Vapnik V (1992) A training algorithm for optimal margin classifiers. In: COLT, pp 144–152
6. Bousquet O, Herrmann DJL (2002) On the complexity of learning the kernel matrix. In: Becker S et al (eds) NIPS. MIT Press, Cambridge, pp 399–406
7. Chang BR, Tsai H-F (2007) Composite of adaptive support vector regression and nonlinear conditional heteroscedasticity tuned by quantum minimization for forecasts. Appl Intell 27(3):277–289
8. Chang C-C, Lin C-J (2001) LIBSVM a library for SVM. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
9. Chapelle O (2004) Support vector machines: induction principle, adaptive tuning and prior knowledge. PhD thesis, UPMC
10. Chapelle O, Vapnik V, Bousquet O, Mukherjee S (2002) Choosing multiple parameters for Support Vector Machines. Mach Learn 46(1/3):131–159
11. Cho S-B, Shimohara K (1998) Evolutionary learning of modular neural networks withgenetic programming. Appl Intell 9(3):191–200

12. Chung K-M, Kao W-C, Sun C-L, Wang L-L, Lin C-J (2003) Radius margin bounds for Support Vector Machines with the RBF kernel. Neural Comput 15(11):2643–2681
13. Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20:273–297
14. Crammer K, Singer Y (2002) On the algorithmic implementation of multiclass kernel-based vector machines. J Mach Learn Res 2:265–292
15. Cristianini N, Shawe-Taylor J (2000) An introduction to support vector machines. Cambridge University Press, Cambridge
16. Cristianini N, Shawe-Taylor J, Elisseeff A, Kandola JS (2001) On kernel-target alignment. In: Dietterich TG, Becker S, Ghahramani Z (eds) NIPS 2001. MIT Press, Cambridge, pp 367–373
17. Dioşan L, Oltean M, Rogozan A, Pécuchet JP (2007) Improving SVM performance using a linear combination of kernels. In: ICANNGA'07. LNCS, vol 4432, pp 218–227
18. Dioşan L, Rogozan A, Pécuchet J-P (2007) Evolving kernel functions for SVMs by genetic programming. In: ICMLA'07, Ohio, USA
19. Frank A, Asuncion A (2010) UCI machine learning repository
20. Friedrichs F, Igel C (2005) Evolutionary tuning of multiple SVM parameters. Neurocomputing 64:107–117
21. Fröhlich H, Chapelle O, Schölkopf B (2003) Feature selection for SVM by means of GAs. In: ICTAI. IEEE, New York, pp 142–148
22. Gagne C et al (2006) Genetic programming for kernel-based learning with co-evolving subsets selection. In: Runarsson TP et al (eds) 9th PPSN'06. Springer, Berlin, pp 1008–1017
23. Girolami M, Rogers S (2005) Hierarchic Bayesian models for kernel learning. In: ICML, pp 241–248
24. Gold C, Sollich P (2003) Model selection for Support Vector Machine classification. Neurocomputing 55(1–2):221–249
25. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison Wesley, Reading
26. Gunn S, Kandola J (2002) Structural modelling with sparse kernels. Mach Learn 48:137–163
27. Hastie T, Rosset S, Tibshirani R, Zhu J (2003/2004) The entire regularization path for the SVM. J Mach Learn Res 5:1391–1415
28. Hooke R, Jeeves TA (1961) Direct search solution of numerical and statistical problems. J ACM 8:212–229
29. Howley T, Madden MG (2005) The genetic kernel Support Vector Machine: description and evaluation. Artif Intell Rev 24(3–4):379–395
30. Huang Y (2009) Advances in artificial neural networks—methodological development and application. Algorithms 2(3):973–1007
31. Igel C (2005) Multi-objective model selection for SVM. In: Coello Coello CA et al (eds) EMO 2005. LNCS, vol 3410. Springer, Berlin, pp 534–546
32. Imbault F, Lebart K (2004) A stochastic optimization approach for parameter tuning of SVM. In: ICPR (4), pp 597–600
33. Joachims T (2001) The maximum-margin approach to learning text classifiers. Künstl Intell 15(3):63–65
34. Keerthi S, Sindhwani V, Chapelle O (2006) An efficient method for gradient-based adaptation of hyperparameters in SVM models. In: NIPS'06. IEEE Computer Society, Los Alamitos, pp 1–10
35. King RD (1992) Statlog databases
36. Kirkpatrick S, Gelatt CD Jr, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
37. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
38. Lacerda E, Carvalho AC, Braga AP, Ludermir TB (2005) Evolutionary radial basis functions for credit assessment. Appl Intell 22(3):167–181
39. Lanckriet GRG et al (2004) Learning the kernel matrix with Semidefinite Programming. J Mach Learn Res 5:27–72
40. Mallick BK, Ghosh D, Ghosh M (2005) Bayesian classification of tumours by using gene expression data. J R Stat Soc Ser B 67(2):219–234
41. Mercer J (1909) Functions of positive and negative type and their connection with the theory of integral equations. Philos Trans R Soc 209:415–446
42. Momma M, Bennett KP (2002) A pattern search method for model selection of SV Regression. In: Grossman RL et al (eds) SIAM 2002. SIAM, Philadelphia, pp 2–16
43. Ohn S-Y, Nguyen H-N, Chi S-D (2004) Evolutionary parameter estimation algorithm for combined kernel function in SVM. In: Content computing, AWCC 2004. Springer, Berlin, pp 481–486
44. Ong CS, Smola A, Williamson B (2005) Learning the kernel with hyperkernels. J Mach Learn Res 6:1043–1071
45. Rakotomamonjy A, Bach FR, Canu S, Grandvalet Y (2007) More efficiency in multiple kernel learning. In: ICML, pp 775–782
46. Lessmann RS, Crone S (2005) Genetically constructed kernels for SVM. In: Proc. of GOR. Springer, Berlin, pp 257–262
47. Schölkopf B (2000) The kernel trick for distances. In: Leen TK, Dietterich TG, Tresp V (eds) NIPS. MIT Press, Cambridge, pp 301–307
48. Schölkopf B, Smola AJ (2002) Learning with kernels. MIT Press, Cambridge
49. Simon HA (2001) The sciences of the artificial, 3rd edn. MIT Press, Cambridge
50. Sonnenburg S et al (2006) Large scale multiple kernel learning. J Mach Learn Res 7:1531–1565
51. Staelin C (2003) Parameter selection for Support Vector Machines. Tech Rep HPL-2002-354R1, Hewlett Packard Laboratories
52. Sullivan K, Luke S (2007) Evolving kernels for SVM classification. In: Lipson H (ed) GECCO 2007. ACM, New York, pp 1702–1707
53. Syswerda G (1991) A study of reproduction in generational and steady state Genetic Algorithms. In: Rawlins GJE (ed) FOGA. Morgan Kaufmann, San Mateo, pp 94–101
54. Taylor JS, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge
55. Tsuda K, Rätsch G, Mika S, Müller K-R (2001) Learning to predict the leave-one-out error of kernel based classifiers. In: LNCS, vol 2130, pp 331–338
56. Vapnik V (1995) The nature of statistical learning theory. Springer, Berlin
57. Vapnik V, Chapelle O (2000) Bounds on error expectation for SVM. Neural Comput 12(9):2013–2036
58. Verma B, Hassan S (2009) Hybrid ensemble approach for classification. Appl Intell, 1–21
59. Wahba G, Lin Y, Zhang H (1999) GACV for support vector machines. In: Smola B, SchRolkopf S (eds) Advances in large margin classifiers. MIT Press, Cambridge
60. Wang G, Yeung D-Y, Lochovsky FH (2007) A kernel path algorithm for SVM. In: ICML 07. ACM Press, New York, pp 951–958
61. Xiong H, Swamy M, Ahmad M (2005) Optimizing the kernel in the empirical feature space. IEEE Trans Neural Netw 16(2):460–474
62. Zhang Z, Jordan MI (2006) Bayesian multicategory support vector machines. In: The twenty-second conference on uncertainty in artificial intelligence (UAI), 2006
63. Zhang Z, Kwok JT, Yeung D-Y (2006) Model-based transductive learning of the kernel matrix. Mach Learn 63(1):69–101

Laura Dioşan is a Lecturer Professor in Artificial Intelligence at Babeş Bolyai University. She obtained her Ph.D. in Artificial Intelligence in 2008, graduating on the topic 'Exploring Hybrid Approaches based on Nature-inspired Algorithms'. Since 2004, her research has been situated in the intersection of Evolutionary Computation, Machine Learning and Optimisation. Her main research interest is mature-inspired optimisation (e.g., evolutionary algorithms, particle swarm ants) for theoretical and practical purposes. She is or has been member of the programme committee of several of the main conferences, workshops and journals in these areas.

Alexandrina Rogozan since September 2000 is Assistant Professor in Artificial Intelligence at LITIS of INSA Rouen, France. She obtained her Ph.D. in Artificial Intelligence at University of Orsay, France, graduating on the topic 'Fusion of heterogeneous data for automatically recognition of speech'. Her main research interest is about multiple kernels, hybrid models, fusion sketches for classification and adaptive methods. The preferred fields of application concern document and image indexation, as well as the intelligent vehicle.

Jean-Pierre Pecuchet since 1988 is Full Professor at LITIS of INSA Rouen, France. He graduated in 1982 with the title of his Ph.D. thesis in Theoretical Computer Science at LITP, France. His current research focuses on the environment for human learning, modelling and simulation of discrete systems, cooperative systems and knowledge modelling.