Improving SVM performance by Evolving a Multiple Kernel Function

Laura Dioşan[1,2], Mihai Oltean[1], Alexandrina Rogozan[2] and Jean Pierre Pecuchet[2]
Department of Computer Science,Faculty of Mathematics and Computer Science,
Babeş-Bolyai University, Cluj-Napoca, Romania

Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes,
Institut National des Sciences Appliquées, Rouen, France
{lauras,moltean}@cs.ubbcluj.ro,
{arogozan,pecuchet}@insa-rouen.fr

April 29, 2007

Abstract

Classical kernel-based classifiers use only a single kernel, but the real-world applications have emphasized the need to consider a combination of kernels - also known as a multiple kernel - in order to boost the classification accuracy. Our purpose is to automatically discover the mathematical expression of a multiple kernel by evolutionary means. In order to achieve this purpose we propose a hybrid model that combines a Genetic Programming (GP) algorithm and a kernel-based Support Vector Machine (SVM) classifier. Each GP chromosome is a tree encoding the mathematical expression of a multiple kernel. The multiple kernel which is evolved in this manner is compared to human-designed kernels by using several real-world data sets. Numerical experiments show that the SVM embedding the evolved multiple kernel performs better than the standard kernels for the considered classification problems.

Keywords: Genetic Programming, SVM, Kernel function, Classification

1 Introduction

Various classification techniques have been used in order to detect the labels correctly associated to some items. Kernel-based techniques (such as Support Vector Machine (SVM) [1]) are an example of such intensively explored classifiers. These methods represent the data by means of a kernel function, which defines similarities between pairs of data [2]. One reason for the success of kernel-based methods is that the kernel function takes relationships that are implicit in the data and makes them explicit, the result being that the detection of patterns takes place more easily. Thus, each kernel function extracts a specific type of information from a given data set, thereby providing a partial description or view of the data.

In general, the kernel-based classifiers only use a single kernel (which is chosen from among some classical kernels and which may not always be suitable for the classification problem), but the real-world applications and the recent developments of various kernel methods have emphasized the need to consider a combination of kernels or, more exactly, a multiple kernel (MK) [3, 4]. This combination could be a linear

one (as in [3, 4, 5, 6]) or a more complex combination. For instance, if we have three classical kernels, K_1, K_2 and K_3, a multiple kernel could be $K_1 + K_2 * K_3$ or $\exp(K_1 + K_3) \times K_2$. A multiple kernel could improve the performance of the SVM algorithm (or other kernel-based methods) because it is flexible and reflects the fact that the learning problem often involves multiple, heterogeneous data.

Moreover, the optimal design of the kernel function and of the values of a small number of parameters is known in literature as *model selection* [5]. This task is usually performed by training a number of classifiers with different permutations from a range of kernel functions and parameters and by subsequently retaining the configuration generating the optimal performance on an independent set of validation patterns [3, 5]. There are also two evolutionary approaches [7, 8] that can be mentioned, both of which are based on GP. They evolve the expression of a single kernel used by the SVM algorithm: the input vectors are used as terminals and some mathematical operators (scalar and vectorial) compose the function set.

Instead of repeating the classification task with different kernels and parameter configurations, we propose an evolutionary approach in which the problem itself could choose the best combination of kernels or the multiple kernel expression that best represents all of the information available for a given learning task. Roughly speaking, the problem finds the multiple kernel expression corresponding to a classification task without any outside help, in other words, by itself.

In order to achieve this goal, we combine the Genetic Programming (GP) [9] and the SVM algorithms [1, 2, 10, 11] within a two-level hybrid model. This GP-based approach yields a general methodology for combining many kernels, each GP chromosome encoding a multiple kernel. The evolved multiple kernel (eMK) is then embedded into a standard SVM algorithm which is used for solving a particular classification problem. The eMKs are compared to several well-known human-designed kernel functions. The proposed method is able to deal with thousands of examples while combining hundreds of kernels within reasonable time.

This research is rooted in the need to answer several important questions concerning the expression of a multiple kernel. The most important question is *whether it is possible to learn the MK function by using some training annotated data* and, in the case of a positive answer, *which are the symbols (or the kernels) that have to be used within a MK for a given classification problem?* We had better let the evolution find the answer for us.

The paper is organized as follows: Section 2 outlines the theory behind SVM classifiers giving a particular emphasis to kernel functions. An overview of the related work in the field of finding the optimal expression for the kernel function is made in Section 3. Section 4 describes the proposed technique for evolving multiple kernels. This is followed by Section 5 where the results of the experiments are presented. Finally, Section 6 concludes our paper.

2 Support Vector Machine

SVMs can solve binary or multiple-class problems. SVMs have originally been proposed for solving the binary classification problems [1, 12]. Consequently, the concepts within SVMs will be explained on binary-labelled data. However, our model can be used for solving problems with any number of classes.

Suppose the training data has the following form: $D = (x_i, y_i)_{i=\overline{1,l}}$, where every $x_i \in \Re^d$ represents an input vector and each y_i, $y_i \in \{-1, 1\}$, an output (or a label). We are interested in finding a function f that takes the set of inputs x and provides the output $y = f(x)$. The aim is to find the function f using just the set of observations D. This function can be viewed as a decision frontier (a hyperplane (w, b)) that separates the input data into two regions:

$$f(x) = \langle w, x \rangle + b,$$

where $w = \sum_i \alpha_i x_i$. The problem is an optimization one: finding the solution α^* that satisfies the inequality $y_i \times f(x_i) \geq 1$. The vectors x_i with $\alpha_i > 0$ are called support vectors.

Because not all input data-points are linearly separable, it is possible to use a kernel function. Kernel methods work by embedding data items into a vector space $\mathcal{F}$ called a feature space, and by searching linear relations in such a space [3]. This embedding is implicitly defined by specifying an inner product for the feature space via a positive semi definite kernel function: $k(x, z) = \langle \Phi(x), \Phi(z) \rangle$, where $\Phi(x)$ and $\Phi(z)$ are the embeddings of data items x and z.

Note that if all that is required in order to find those linear relations are inner products, we do not need to have an explicit representation of the mapping Φ. Moreover, we do not even need to know the nature of the feature space. The only requirement is to be able to evaluate the kernel function, which is often much easier than explicitly computing the coordinates of the points. Evaluating the kernel on all the pairs of data items yields a symmetric, positive semi definite matrix K known as the kernel matrix or Gram matrix [13], which can be regarded as a matrix of generalized similarity (on what concerns the measures) among the data points.

By using kernels, the solutions could be expressed like an affine function:

$$f(x) = \langle w, \Phi(x) \rangle + b,$$

for some weight vector w. The kernel can be exploited whenever the weight vector can be expressed as a linear combination of the training points, $w = \sum_{i=1}^{l} \alpha_i \Phi(x_i)$, implying that f can be expressed as:

$$f(x) = \sum_{i=1}^{l} \alpha_i k(x_i, x) + b.$$

The kernel formalism allows different kernels to be combined. Basic algebra operations such as addition, multiplication and exponentiation preserve the key properties for a kernel (the positive semi-definiteness and the symmetry) and thus allow a simple, but powerful algebra of kernels to exist [2, 14].

For instance, given a set of kernels $\{K_1, K_2, \ldots, K_m\}$, we can form the linear combination

$$K = \sum_{i=1}^{m} w_i * K_i,$$

where the weights w_i are constrained to be non-negative to assure semi-definiteness [3].

Or, even better, we can consider complex combinations of kernels:

$$K_1 + K_2 * K_3$$

or

$$c * K_2 + K_3 * exp(K_1),$$

where c is a constant. In this paper we focus on this kind of kernel combination (or a multiple kernel) and we try to discover its expression.

3 Related work

Evolutionary techniques have been used in the past for evolving other complex structures.: the expression of crossover operator applied in genetic algorithms for function optimization [15], a non-generational EA [16] or a generational EA [17] are only few examples.

Regarding the SVM kernel expression, several methods [3, 5, 18] have been proposed for optimizing the parameters of the kernel functions, but, to our knowledge, only two attempts [7] and [8, 19] deal with the shape of a single kernel embedded into a SVM algorithm. In these approaches, the authors have tried to find an optimal kernel function using genetic programming technique. There are several and important differences between their approaches:

D1 *function set* - the model from [8] used only few binary operators ($+$, $-$, $\times$), while that from [7] extended the function set by adding:

- several unary scalar operations (such as $\exp$, $\sin$, $\cos$, $\log$),
- some norm functions (EN, SP, GN - that transform the $\Re^d$ space into $\Re$ space and create the link between the GP tree part reserved for $\Re^d$ space and the GP tree part reserved for $\Re$ space) and
- several element-wise operations ($\oplus, \ominus, \otimes$).

Moreover, the model proposed in [8] has used the same function set for both type of operations (scalar and vectorial). The model developed in [7] has used two different function sets (one for scalar operations and one for vectorial operations).

D2 *terminal set* - in the model from [8] the terminals can be either vectors, either scalar values. In the approach from [7] the terminals are only vectors. Moreover, the authors have used in [8] a trick: their model decides the type of operators (scalar or vector) based on the type of arguments (if both arguments are scalar then the function is scalar and if least one operand is a vector, then the vectorial meaning for the operator is used). The other model (from [7]) decides the type of operands based on the type of functions. Starting with the initialization stage and continuing with the crossover and mutation steps, the set of functions contains both scalar and vector operators which is used for generating/obtaining valid kernel expressions (unlike the other approach were the correctness of the kernel is imposed after the construction stage).

D3 *Mercer conditions* - the positivity and the symmetry of evolved kernels from [7] are verified when a kernel is evaluated (the expressions that do not satisfy these constraints are penalized). Unlike this approach, [8] firstly has evaluated the kernel encoded into a GP tree on two samples x and z. These samples are swapped and the kernel is evaluated again. The dot-product of these two evaluations is returned as the kernel output. In this manner, he obtained symmetric kernels, but does not guarantee that they obey Mercer's theorem. Moreover, the dot-product operation increases the kernel complexity.

The approach presented in this paper also uses GP for evolving kernels. But, the trees encoding kernels are more complex this time since they can contain, in their leaves, other standard kernels whose good performance was already proven.

Regarding a multiple kernel, only three attempts for finding the weights of a combined kernel for SVM algorithm were found in the literature. Recently, two new approaches [3, 4] have considered the conic combinations of kernel matrices for the SVM and have showed that the optimization of the coefficients of such a combination reduces to a convex optimization problem known as a quadratically constrained quadratic program. The authors have shown how the kernel matrix can be learnt from data via semi-definite programming [3] and via semi-infinite linear programming [4]. The other attempt [20] is an improved approach of [3]. It has proposed a novel dual formulation of the quadratically-constrained quadratic problem as a second-order cone-programming problem.

Both approaches are matrix-based techniques because, in both cases, the algorithms learn the kernel matrix (or the Gram matrix) associated to the multiple kernel. The model selection can be viewing in terms of Gram matrices rather than kernel functions.

The paper [21] describes an approach to optimizing kernel parameters for some standard kernels, and then combining those kernels through addition and/or multiplication to yield a multiple kernel.

Another approach [22] uses genetic algorithms (GAs) [23] for evolving the weights of a linear combination of kernels. The model proposed in [22] can be considered an evolutionary alternative to the above approaches. Each GA chromosome encodes the weights of each individual kernel (see Table 1) involved into the linear combined kernel. Each weight is a real value from a particular range. An uniform arithmetical crossover [24] and a Gaussian mutation [25, 26] have been used by 0.8 and, respectively, 0.1 probabilities.

4 Proposed model

The model for evolving the mathematical expression of a multiple kernel is described in this section. Our hybrid approach combines a GP algorithm and an SVM algorithm. Each GP chromosome encodes the expression of a multiple kernel used by the SVM algorithm. The quality of a GP chromosome is given by the classification accuracy obtained through running the SVM algorithm (which involves the multiple kernel encoded in the current GP tree) in order to solve a particular classification problem.

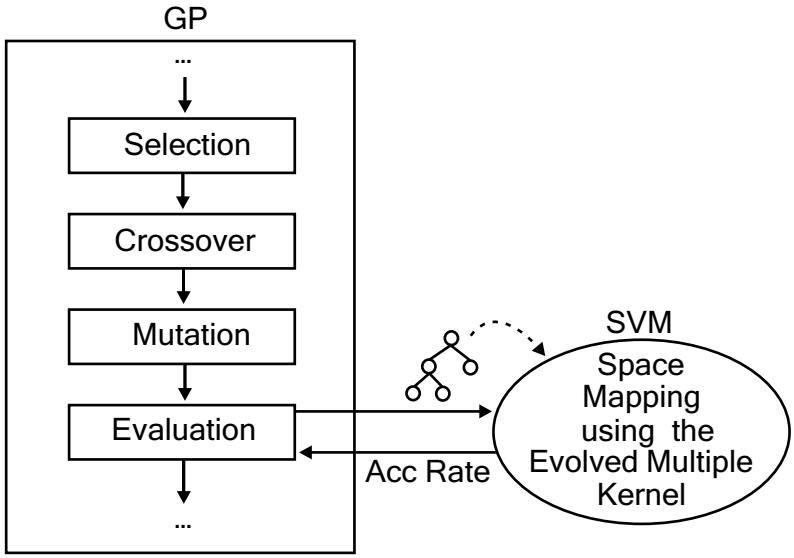

Figure 1: Sketch of the hybrid approach

4.1 The GP representation of a multiple kernel

In our model, the GP chromosome is a tree encoding the mathematical expression of a multiple kernel. The GP individual representation is based on the kernel algebra [2]. More complex kernels (or multiple kernels)

can be obtained from simple kernels by transforming or combining the latter ones according to specific rules (which regard the positiveness and the symmetry of the Gram matrix [13]).

A simple way of combining more kernels is very similar to the method used for evolving mathematical expressions (as in the case of regression problem): the kernels play the part of regression variables while regression operations are replaced with those performed between kernels. Despite the fact that any mathematical operator can be used in the regression problem, in our case we must restrict the functions involved in the MK expression. Why? Because not all the operations performed between kernels provide another valid kernel. Only several mathematical functions preserve the key properties of the kernel functions (the positiveness and the symmetry of the Gram matrix), such as *addition, multiplication* and *exponentiation* [2]. In kernel algebra theory the *power* function is also specified, but this operation (with a natural exponent) can be obtained as a repeated multiplication. Therefore, the function set will contain only these three operations allowed to be performed between kernels: $FS = \{+, \times, \exp\}$.

The terminal set contains several classical kernels (such as linear kernel, polynomial kernel, radial basis function (RBF) kernel) and some ephemeral random constants [9]: $TS = KTS \cup \{c_1, c_2, \ldots c_m\}$, where KTS represents the terminal set of classical kernels. Note that a kernel-GP tree must contain at least one kernel in its leaves (the number of kernel terminals involved in an MK must be greater than or equal to 1), otherwise the obtained expression is only a mathematical one, and not a kernel function.

The proposed model uses a particular type of GP tree: its leaves contain either a kernel or a constant. Some additional information must be added to the kernel-leaves of the tree. More exactly, a kernel is a function with two arguments (x and z) that represent the input instances; the kernel has to map these inputs into another space and it should be a larger one. Therefore, the leaves that contain a kernel will have two other little-leaves associated to them. The latter ones correspond to the input vectors.

An example of a GP chromosome is depicted in Figure 2. We have used $FS = \{+, \times, \exp\}$ and $TS = \{K_1, K_2, K_3, c_1, c_2, \ldots c_m\}$ for this chromosome, but only two functions ($+$ and $\times$), two kernels (K_2 and K_3) and two constants (c_1 and c_2) are actually involved in the MK expression.

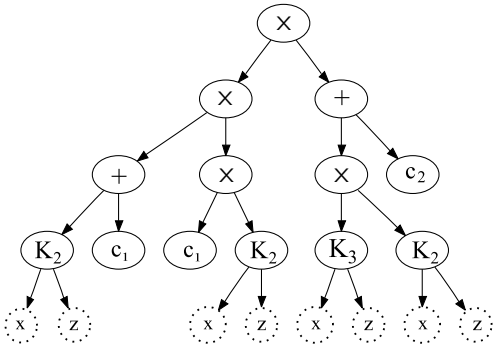

Figure 2: Example of a chromosome - the nodes represented by solid circles contain the GP tree symbols (functions, kernels or constants) and the nodes represented by dashed circles contain the input vectors (x and z) for a kernel. This chromosome encodes the following multiple kernel expression: $(K_2(x,z) + c_1) \times c_1 \times K_2(x,z) \times (K_3(x,z) \times K_2(x,z) + c_2)$.

4.2 Genetic operations

Initialization We use the *grow method* [27] which is a recursive procedure for initializing a GP individual. The root of each GP tree can be a function from FS. If a node contains a function, then its children will be initialized either with another function or with a terminal (a kernel or a constant). The initialization process is stopped at the maximal level of the tree where we put a terminal. For obtaining a valid MK, at least one GP tree leaf has to contain a kernel (otherwise, we only obtain a simple mathematical expression).

Crossover Just like mutation, crossover is performed in a structure-preserving way for ensuring the syntactical validity of the offspring. The proposed model uses a one-cutting point crossover, but with a restriction: the cutting-point must be chosen in such a way that the offspring will contain at least one kernel in its leaves.

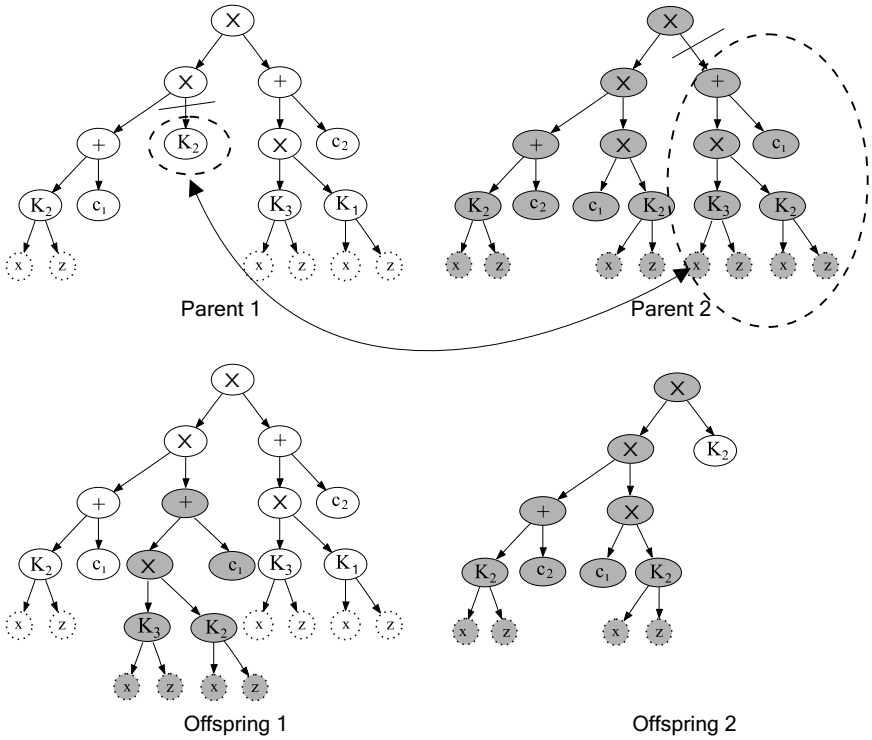

Figure 3: Example of a crossover - the parent chromosomes encode the combined kernels: $(K_2(x,z)+c_1)\times K_2(x,z)\times(K_3(x,z)\times K_1(x,z)+c_2)$ and $(K_2(x,z)+c_2)\times c_1\times K_2(x,z)\times(K_3(x,z)\times K_2(x,z)+c_1)$ and the offspring kernels are: $(K_2(x,z)+c_1)\times(K_3(x,z)\times K_2(x,z)+c_1)\times(K_3(x,z)\times K_1(x,z)+c_2)$ and $(K_2(x,z)+c_2)\times c_1\times K_2(x,z)\times K_2(x,z)$

Mutation A cutting point is randomly chosen: the subtree belonging to that point is deleted and a new subtree is grown there using the same random growth process that was used to generate the initial population. Note that the growth process is limited by a maximal height allowed for the GP trees and, like in Koza's implementation [9], new constants may be generated by the mutation operator at any point in a run.

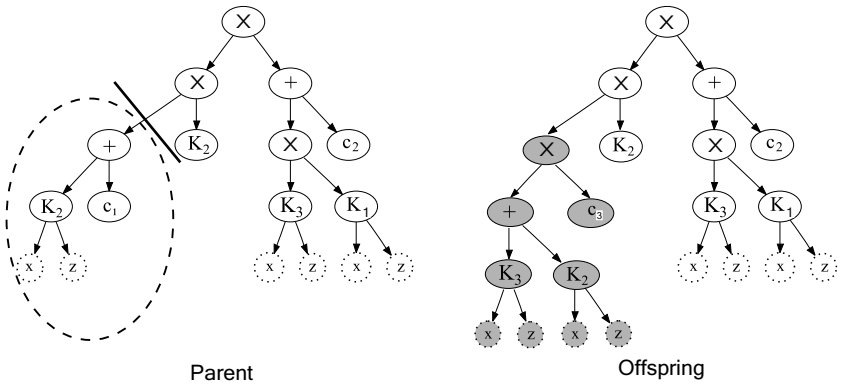

Figure 4: Example of a mutation. Mutating the chromosome $(K_2(x,z)+c_1) \times K_2(x,z) \times (K_3(x,z) \times K_1(x,z)+c_2)$ we obtained the combined kernel $(K_3(x,z)+K_2(x,z)) \times c_3 \times K_2(x,z) \times (K_3(x,z) \times K_1(x,z)+c_2)$

Note that the initialization, recombination and mutation operators always generate valid kernels.

4.3 Fitness assignment

Before presenting the details about the chromosome evaluation, we must provide information about the data sets. Each data set is randomly divided into two subsets: "training" and "testing" sets, data set partitions that correspond to the learning and testing steps of the SVM algorithm. The SVM algorithm (embedding a standard kernel in general and embedding the eMK in our model case) uses the first two thirds of a data set for learning the model and the last third of each data set for testing purposes.

The quality of a GP chromosome is given by the accuracy rate (the number of correctly classified items over the total number of items) computed by the SVM (that uses as kernel the eMK encoded into the current GP individual) on the test data set.

For evaluating the quality of an eMK, we also have to look if that kernel is a valid one (with other words if the eMK satisfies the Mercer conditions [13]). It is possible to use an evaluation method that penalizes the "invalid" eMKs, but we do not need it because the initialization process and the genetic operators guarantee the correctness (from a kernel's point of view) of a GP chromosome.

4.4 The Algorithms

The algorithms used for evolving the multiple kernel are described in this section. We deal with a hybrid technique structured on two levels: a macro level and a micro level.

The macro level algorithm is a standard GP [9, 27] used for evolving the mathematical expression of a multiple kernel. We use a steady-state evolutionary model [28, 29] as underlying mechanism for our GP implementation. The GP algorithm starts by creating a random population of individuals. Each individual is a tree that encodes the expression associated to a multiple kernel. The following steps are repeated until a given number of generations are reached: two parents are selected using a standard selection procedure; the parents are recombined (using one-cutting point crossover) in order to obtain an offspring O; the offspring is considered for the mutation which is performed by replacing a sub-tree with another sub-tree generated

using the same procedure as in the initialization stage; the offspring O replaces the worst individual W in the current population if O is better than W.

The micro level algorithm is an SVM algorithm. It is taken from $libsvm$ [30] and it uses the 1-norm soft-margin approach [2, 31]. Original implementation of the SVM algorithm from $libsvm$ [30] allows the use of several well-known kernels (see Table 1). In our experiments, we used a modified version of this algorithm which is based on our evolved multiple kernel. In order to compute the quality of each GP individual, we run the SVM algorithm embedding the evolved multiple kernel and the accuracy rate computed by the classifier will represent the fitness of the GP tree.

Table 1: The expression of several classical kernels

Kernel name	**Kernel expression**
Linear	$K_{Lin}\ \ (x,z) = x * z$
Polynomial	$K_{Pol}\ \ (x,z) = (\gamma * x * z + coef)^d$
Radial basis function	$K_{RBF}(x,z) = exp(-\gamma * \lvert x - z\rvert^2)$
Sigmoid	$K_{Sig}\ \ (x,z) = \tanh(\gamma * x * z + coef)$

5 Experiments

We have performed several computational experiments: a performance analysis of our hybrid approach, a performance comparison between the eMKs and some other kernels (human-designed) and a testing of the eMKs generalization ability.

In our experiments we have used several data sets taken from Machine Learning Repository UCI [32]. All data sets contain information about real-world problems:

- P_1 – classification of radar returns from the ionosphere,
- P_2 – breast cancer classification,
- P_3 – heart disease diagnosis,
- P_4 and P_5 – classifications of personal income.

For the GP algorithm we have used a simple steady-state model [28, 29] and a population of 50 individuals that are evolved during 50 generations. The maximal depth of a GP tree has been limited to 10 levels. For chromosome selection, we have used the binary tournament mechanism and the crossover and mutation operations have been performed with 0.8 and respectively, 0.3 probabilities.

The general parameter (C) of the SVM algorithm has been set to 1 in all the experiments. On what concerns the other SVM parameters (or better said, the individual kernel parameters), we have used different values, corresponding to different terminal sets involved in our experiments.

5.1 Evolving the MK function

A multiple kernel is evolved in this experiment. In oder to evolve this kind of kernel we use two different terminal set types: a terminal set that contains only several standard kernels KTS and a mixed terminal set that contains standard kernels and constants $MTS = KTS \cup \{c_1, c_2, \ldots c_m\}$.

We have to note several things about the value of constants $c_1, c_2, \ldots c_m$. Mercer conditions [13] impose that these constants must be positive. [3] suggests the $[0, 1]$ range for all the constants c_j. Therefore, we use the interval from 0 to 1 for all the constants c_j, $j = \overline{1, m}$.

Different types of *TS*s are used in the numerical experiments:

- a TS with different standard kernels

$$KTS_1 = \{K_{Lin}, K_{Pol}, K_{RBF}, K_{Sig}\};$$

  in our experiments we use for the standard kernels (see Table 1) the following parameters: $\gamma = 0.9$, $coef = 1$, $degree = 3$;

- a TS that contains the same kernel (but with different parameters); we select the well-known RBF kernel:

$$KTS_2 = \{K_{RBF}^{\gamma=10}, K_{RBF}^{\gamma=1}, K_{RBF}^{\gamma=0.1}, K_{RBF}^{\gamma=0.01}\}.$$

  The difference between these kernels is determined by the value of the γ parameter (10, 1, 0.1 and 0.01, respectively - these values are commonly used in literature [5]);

- a TS with different standard kernels and constants

$$MTS_1 = KTS_1 \cup \{c_1, c_2, \ldots, c_m\};$$

- a TS that contains the same kernel (but with different parameters) and constants

$$MTS_2 = KTS_2 \cup \{c_1, c_2, \ldots, c_m\}.$$

During different runs, we have obtained various expressions for the multiple kernel function, all of which have about the same complexity. For instance, in four particular runs (with different TS compositions) we have obtained the following MKs for the problem P_1:

$$KTS_1\text{:} \qquad eMK = K_{RBF} + K_{Lin} + K_{RBF}$$

$$KTS_2\text{:} \qquad eMK = K_{RBF}^{\gamma=0.1} + K_{RBF}^{\gamma=10}$$

$$MTS_1\text{:} \qquad eMK = K_{RBF} \times K_{Lin} + 0.87 \times K_{RBF} + K_{Lin}$$

$$MTS_2\text{:} \qquad eMK = K_{RBF}^{\gamma=0.1} \times (K_{RBF}^{\gamma=0.01} + 0.82)$$

Figure 5 depicts the evolution of the quality for the best individual from the population along with the number of generations (for all the problems). We can observe that there are only small improvements 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 GP model can find an efficient MK in only a few generations.

The performance of eMKs based on various TSs is presented in Table 2: the first four rows contain the accuracy rates (for each problem) computed by the SVM algorithm involving the best eMK. This eMK is the best MK obtained at the end of the evolutionary process (the best GP chromosome from the last generation). In Table 2 we also present the performance of three classical kernels for the test problems (the last three rows).

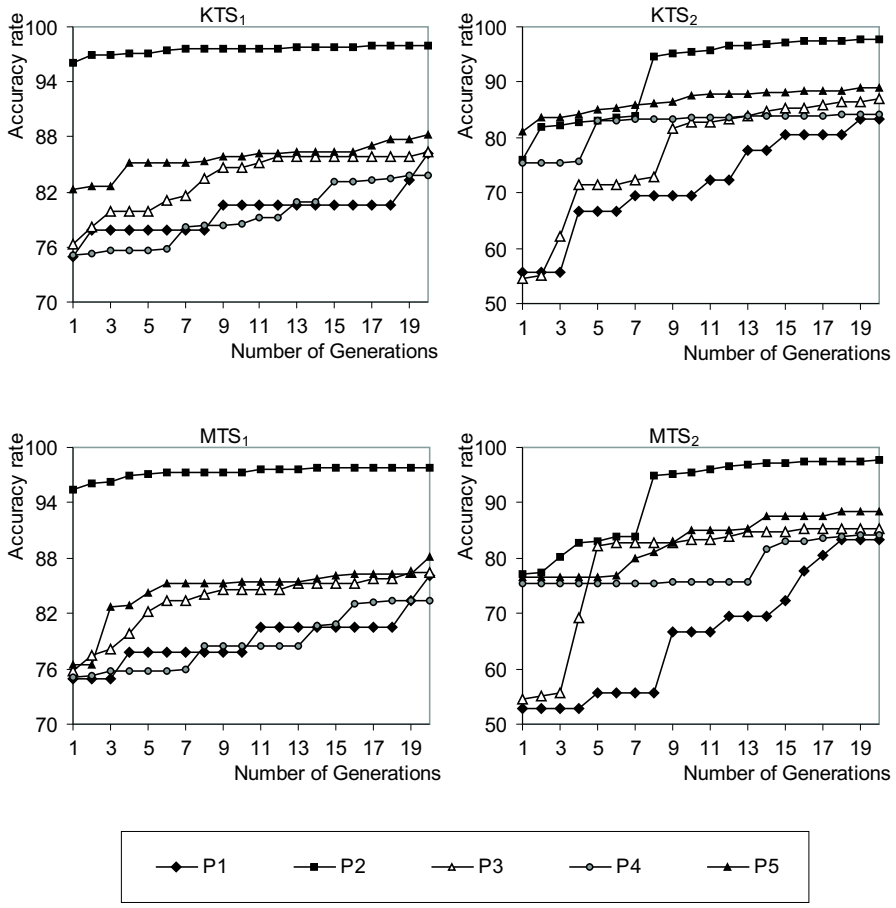

Figure 5: Fitness (Accuracy Rate – Acc) evolution for the best multiple kernel for all the test problems (P_1, P_2, P_3, P_4, P_5) along with the number of generations. We have tested different TS compositions: KTS_1, KTS_2, MTS_1 and MTS_2

Table 2: The accuracy rate of various kernels. The first four rows present the accuracy rates computed by the SVM algorithm embedding the best eMK obtained for each test problem (in this case the training stage is performed on the first two thirds of a data set and the test stage is performed on the last third of each data set). The results are averaged over 10 runs. The last three rows contain the performance of the classical kernels for each test problem.

			P_1	P_2	P_3	P_4	P_5
Train and test on the same problem	KTS	KTS_1	86.11	98.03	86.39	83.73	88.31
		KTS_2	83.33	97.81	86.39	84.22	88.99
	MTS	MTS_1	80.56	98.03	86.39	83.73	88.13
		MTS_2	83.33	97.81	85.21	84.30	88.48
Standard Kernels		K_{Lin}	75.00	96.49	75.74	78.41	86.33
		K_{RBF}	77.78	97.15	77.51	75.69	85.38
		K_{Sig}	50.00	87.72	82.25	75.19	76.10

The values from Table 2 indicate that the eMKs always perform better than the classical individual kernels. Taking into account different TS compositions, we can observe that the eMKs based only on kernels perform better than the mixed eMKs in all the cases only one exception (problem P_4 where the eMK based on MTS_2 performs better than the eMK based on KTS_2). If we considered the homogeneity of the TS (with other words, if the TS contains different kernel types or the same kernel, but with different parameters), we can see that the eMKs embedding different parameterized RBF kernels perform better than the eMKs that embed different kernels types in 6 cases (out of 10).

5.2 Assessing the performance of the eMK function

This experiment serves our purpose of assessing the performance of the evolved multiple kernel. A SVM algorithm is used for this purpose and it has the same structure and parameters as the one used in the previous experiment (the micro level algorithm), the only difference being the employed kernel. We used one of the evolved multiple kernels obtained in Section 5.1 on one of the data sets. This kernel function is embedded into the SVM algorithm, which is run against the other problems. These tests are repeated for all the TS constructions. The accuracy rates obtained for each problem and for each TS composition are then compared with those computed by a similar SVM algorithm, with the same parameters, which uses:

- the classical kernels one by one,
- the simple kernel evolved by using the GP-based model from from [7] (this kernel will be denoted by *eK*),
- the linear combined kernel evolved by using the model from [22] (this kernel will be denoted by *eCK*).

The results of this experiment are presented in Table 3.
Table 3 shows that the eMKs trained on the first problem have a very good generalization ability:

- the eMKs perform better than the classical individual kernels in 19 cases out of 20,
- the eMKs perform better than the evolved individual kernel *eK* in all cases,
- the eMKs perform better than the evolved combined kernel (eCK) in 13 cases out of 20.

Table 3: The accuracy rate of various kernels: evolved kernels (the first four rows and the last two rows) or standard kernels (the rest of rows). The first four rows contain the accuracy rates computed by the SVM algorithm, which uses the best eMK obtained for the first problem P_1 (in this case the training stage is performed on the P_1 data set and the test stage is executed on the other problems). The results are averaged over 10 runs. The last three rows contain the performance of the classical kernels for each test problem.

			P_1	P_2	P_3	P_4	P_5
Train on P_1 and test against the other problems	KTS	KTS_1	86.11	97.60	85.80	83.65	88.56
		KTS_2	83.33	97.60	83.43	83.73	87.88
	MTS	MTS_1	80.56	97.15	84.02	83.65	88.31
		MTS_2	83.33	97.81	83.43	83.31	84.95
Standard Kernels		K_{Lin}	75.00	96.49	75.74	78.41	86.33
		K_{RBF}	77.78	97.15	77.51	75.69	85.38
		K_{Sig}	50.00	87.72	82.25	75.19	76.10
Other evolved kernels		eK	77.78	69.96	54.44	76.14	76.01
		eCK	80.55	97.58	85.21	83.42	85.72

If we perform the same analysis as in the previous experiment, we observe that:

- the eMKs based only on kernels perform better than the eMKs based on a mixed TS in 9 cases (out of 10),
- the homogeneous eMKs determine a higher accuracy rate than the un-homogeneous eMKs only in 4 cases (out of 10).

Note that because the SVM algorithm does not work with random values, it suffices to perform only a single run for the hybrid model. We do not need to average the best accuracy rates obtained in more runs.

In order to emphasize the improvements obtained by using a SVM algorithm which involved a multiple kernel we have computed the average performance ($\overline{P}$) improvement for each multiple kernel type as a mean of the all percent differences Δ for all problems over the number of problems. δ_i is the percent difference between the accuracy rate computed by the SVM algorithm with a multiple kernel (Acc_{MK}) and:

- the accuracy rate computed by an algorithm with a simple kernel (SK) - human-designed or evolved - for the i^{th} problem,
- the accuracy rate computed by an algorithm with a evolved linear combined kernel (eCK) for the i^{th} problem.

$$\delta_i = \frac{Acc^i_{MK} - Acc^i_K}{Acc^i_K}, i = \overline{1,5}, \tag{1}$$

where K could be K_{Lin}, K_{RBF}, K_{Sig}, eK or eCK,

$$\overline{\Delta} = \frac{\sum_{i=1}^{5} \delta_i}{5}. \tag{2}$$

The values of the performance improvement are given in Table 4 and they show that:

- we obtained better evolved MKs in all cases,
- the best average improvement (over all problems) is obtained by a SVM algorithm which involves a multiple kernel based on the KTS_1 (this evolved MK is better than all the simple kernels and also it is better than the evolved linear combined kernel).

Table 4: The average performance improvements for the multiple kernels

$\overline{\Delta}$	K_{Lin}	K_{RBF}	K_{Sig}	eK	eCK
KTS_1	7.70%	7.22%	23.08%	26.84%	2.24%
KTS_2	6.20%	5.76%	21.24%	25.10%	0.85%
MTS_1	5.60%	5.18%	20.26%	24.56%	0.29%
MTS_2	5.46%	5.00%	20.41%	24.27%	0.11%

In order to determine whether the differences between the performance of the eMKs and that of the standard kernels, evolved simple kernel eK and, respectively, evolved linear combined kernel eCK are statistically significant, we use a t-test with a 0.05 level of significance. This analysis regards the values obtained on the second experiment.

Table 5: The values for the F-test (along with to the confidence interval)

	K_{Lin}	K_{RBF}	K_{sig}	eK	eCK
KTS_1	0.36 (0.03,3.47)	0.37 (0.03,3.61)	0.14 (0.01,1.36)	0.31 (0.03, 3.06)	0.70 (0.07, 6.72)
KTS_2	0.45 (0.04, 4.36)	0.47 (0.04, 4.54)	0.17 (0.01, 1.71)	0.40 (0.04, 3.85)	0.88 (0.09, 8.45)
MTS_1	0.50 (0.53, 4.84)	0.52 (0.54, 5.04)	0.19 (0.02, 1.90)	0.44 (0.04, 4.27)	0.97 (0.10, 9.38)
MTS_2	0.48 (0.05, 4.66)	0.50 (0.53, 4.84)	0.19 (0.01, 1.83)	0.43 (0.04, 4.12)	0.94 (0.09, 9.04)

Table 6: The values for the t-test (along with to the confidence interval)

	K_{Lin}	K_{RBF}	K_{sig}	eK	eCK
KTS_1	1.25 (-5.39, 17.29)	1.20 (-5.51, 16.79)	2.03 (-3.57, 31.75)	3.52 (5.48, 29.46)	0.48 (-6.96,10.65)
KTS_2	0.98 (-6.76, 16.36)	0.93 (-6.89, 15.87)	1.84 (-4.74, 30.62)	3.19 (4.15, 28.49)	0.17 (-8.52, 9.92)
MTS_1	0.87 (-7.34, 16.03)	0.82 (-7.48, 15.55)	1.76 (-5.21, 30.18)	3.05 (3.59, 28.15)	0.05 (-9.20, 9.69)
MTS_2	0.84 (-7.47, 15.81)	0.79 (-7.60, 15.33)	1.74 (-5.37, 30.00)	3.04 (3.46, 27.93)	0.01 (-9.29, 9.43)

The values of the *F*-test and *t*-test are given in Tables 5 and 6 and they show that:

- the variances of each paired samples are equal in 18 cases out of 20 - in these 18 cases, the values of the *F*-test are in the confidence interval - see Table 5,
- the true differences in means are equal to 0 in 16 cases out of 20 - taking into account the values of the *t*-test - see Table 6. Therefore, this means that the performances of the eMKs are significant better than those of the simple kernels (human-designed or evolved) or those of the evolved combined kernel.

6 Conclusion and further work

A hybrid technique for evolving multiple kernel functions has been proposed in this paper. The model has been used for evolving the mathematical expression of multiple kernel for the SVM algorithm which solves a binary classification problem.

We have performed several numerical experiments in order to compare our evolved multiple kernel to other kernels (human designed). Numerical experiments have shown that the evolved multiple kernels perform better than the standard (linear, RBF and sigmoid) kernels.

However, taking into account the No Free Lunch theorems for Search [33] and Optimization [34] we cannot make any assumption about the generalization ability of the evolved kernel functions. Further numerical experiments are required in order to assess the power of the evolved expressions.

Further work will be focused on:

- evolving, at the same time, both components of a multiple kernel function: the expression (the kernel shape) and the parameters of the individual kernels (involved into the multiple kernel),
- evolving better kernel functions,
- using multiple data sets for the training stage. This can help us to evolve more robust kernels, but in this case the operation is more time-consuming.

References

[1] Vapnik, V.: The Nature of Statistical Learning Theory. Springer Verlag, New York (1995)

[2] Schoelkopf, B., Smola, A.J.: Learning with Kernels. The MIT Press, Cambridge, MA (2002)

[3] Lanckriet, G.R.G., et al.: Learning the kernel matrix with semidefinite programming. Journal of Machine Learning Research **5** (2004) 27–72

[4] Sonnenburg, S., Rtsch, G., Schafer, C., Scholkopf, B.: Large scale multiple kernel learning. Journal of Machine Learning Research **7** (2006) 1531–1565

[5] Chapelle, O., Vapnik, V., Bousquet, O., Mukherjee, S.: Choosing multiple parameters for support vector machines. Machine Learning **46**(1/3) (2002) 131–159

[6] Ong, C.S., Smola, A., Williamson, B.: Learning the kernel with hyperkernels. Journal of Machine Learning Research **6** (2005) 1043–1071

[7] Diosan, L., Oltean, M.: Evolving kernel function for support vector machines. In Cagnoni, S., et al., eds.: Proceedings of ECAI, ITC-IRST (2006) 11–16

[8] Howley, T., Madden, M.G.: The genetic kernel support vector machine: Description and evaluation. Artif. Intell. Rev **24**(3-4) (2005) 379–395

[9] Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992)

[10] Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines. Cambridge University Press (2000)

[11] Joachims, T.: Making large–scale SVM learning practical. In Scholkopf, B., Burges, C.J.C., Smola, A.J., eds.: Advances in Kernel Methods — Support Vector Learning, Cambridge, MA, MIT Press (1999) 169–184

[12] Burges, J.C.: A tutorial on support vector machines for pattern recognition. In: Knowledge Discovery and Data Mining. Volume 2. (1998) 121–167

[13] Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20** (1995) 273–297

[14] Berg, C., Christensen, J.P.R., Ressel, P.: Harmonic Analysis on Semigroups. Springer, Berlin (1984)

[15] Dioşan, L., Oltean, M.: Evolving the structure of the particle swarm optimization algorithms. In Gottlieb, J., et al., eds.: EvoCOP 2006. Volume 3906 of LNCS., Springer (2006) 25–36

[16] Oltean, M., Grosan, C.: Evolving evolutionary algorithms using multi expression programming. In Banzhaf, W., et al., eds.: Advances in Artificial Life. 7th ECAL. Number 2801 in LNAI, Springer (2003) 651–658

[17] Oltean, M.: Evolving Evolutionary Algorithms using Linear Genetic Programming. Evolutionary Computation (MIT Press) **13**(3) (2005) 387–410

[18] Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., Vapnik, V.: Feature selection for SVMs. In Leen, T.K., Dietterich, T.G., Tresp, V., eds.: NIPS, MIT Press (2000) 668–674

[19] Howley, T., Madden, M.G.: An evolutionary approach to automatic kernel construction. In Kollias, S.D., Stafylopatis, A., Duch, W., Oja, E., eds.: ICANN (2). Volume 4132 of Lecture Notes in Computer Science., Springer (2006) 417–426

[20] Bach, F.R., Lanckriet, G.R.G., Jordan, M.I.: Multiple kernel learning, conic duality, and the SMO algorithm. In: Machine Learning, Proceedings of ICML 2004, ACM (2004)

[21] S. Lessmann, R.S.a.S.C.: Genetically constructed kernels for support vector machines. In: Proc. of German Operations Research. (2005)

[22] Diosan, L., Oltean, M.: Improving svm performance using a linear combination of kernels. In: Adaptive and Natural Computing Algorithms, ICANNGA'07. Volume 4431 of LNCS. (2006) accepted.

[23] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley (1989)

[24] Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer (1992)

[25] Fogel, L.J., Owens, A.J., Walsh, M.J.: Artificial Intelligence through Simulated Evolution. John Wiley & Sons, New York (1966)

[26] Yao, X., Liu, Y., Lin, G.: Evolutionary Programming made faster. IEEE-EC **3**(2) (1999) 82

[27] Banzhaf, W.: Genetic programming: An introduction: On the automatic evolution of computer programs and its applications (1998)

[28] Syswerda, G.: Uniform crossover in genetic algorithms. In: ICGA. (1989) 2–9

[29] Syswerda, G.: A study of reproduction in generational and steady state genetic algorithms. In Rawlins, G.J.E., ed.: Proceedings of FOGA Conference, Morgan Kaufmann (1991) 94–101

[30] Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. (2001) Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[31] Boser, B.E., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In: COLT. (1992) 144–152

[32] UCI: Machine Learning Repository. (2007) `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/data/`.

[33] Wolpert, D.H., Macready, W.G.: No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute (1995)

[34] Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1) (1997) 67–82