STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

EVOLUTIONARY OPTIMISATION OF KERNEL FUNCTIONS FOR SVMS

LAURA DIOŞAN, ALEXANDRINA ROGOZAN, AND JEAN-PIERRE PECUCHET

Abstract. The kernel-based classifiers use one of the classical kernels, but the real-world applications have emphasized the need to consider a new kernel function in order to boost the classification accuracy by a better adaptation of the kernel function to the characteristics of the data. Our purpose is to automatically design a complex kernel by evolutionary means. In order to achieve this purpose we develop 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 the kernel function. The evolved kernel is compared to several human-designed kernels and to a previous genetic kernel on several datasets. Numerical experiments show that the SVM embedding our evolved kernel performs statistically better than standard kernels, but also than previous genetic kernel for the considered classification problems.

Key words: Support vector machines, Evolutionary optimisation, Genetic programming.

1. Introduction

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. In 1995, Support Vector Machines (SVMs) marked the beginning of a new era in the paradigm of learning from examples. Rooted to the Statistical Learning Theory and the Structural Risk Minimization principle developed by Vladimir Vapnik at AT&T in 1963 [19, 20], SVMs gained quickly attention from the Machine Learning community due to a number of theoretical and computational merits.

SVMs are a group of supervised learning methods that can be applied to classification or regression. SVMs arose from statistical learning theory; the aim being to solve only the problem of interest without solving a more difficult problem as an intermediate step. SVMs are based on the structural risk

Received by the editors: September 10, 2007.
2000 *Mathematics Subject Classification.* 68T05,91E45.
1998 *CR Categories and Descriptors.* I.2.6 [**Learning**]: – *Concept learning*.

minimisation principle, closely related to regularisation theory. This principle incorporates capacity control to prevent over-fitting and thus is a partial solution to the bias-variance trade-off dilemma.

One issue with SVMs is finding an appropriate positive definite kernel (and its parameters) for the given data. A wide choice of kernels already exists. Many data or applications may still benefit from the design of particular kernels, adapted specifically to a given task (i.e. kernels for vectors, kernels for strings, kernels for graphs, Fisher kernels or rational kernels). There are only some hints for working with one or another of these classic kernels, because there is no rigorous methodology to choose *a priori* the appropriate one between them. Moreover, the kernel parameters influence the performance of the SVM algorithm. The selection of the penalty error for an SVM (that controls the trade-off between maximizing the margin and classifying without error) is also critical in order to obtain good performances. Therefore, one has to optimise the kernel function, the kernel parameters and the penalty error of the SVM algorithm in order to guarantee the robustness and the accuracy of an SVM algorithm. Chapelle [4] has proposed to denote the kernel and SVM parameters as hyper-parameters.

On the other hand, the evolutionary algorithms are able to search in a continuous space without respecting some conditions (requirements) as those regarding the differentiable of the score function. We do not have the certitude that the solution provided by an EA is the optimal one, but it is very close to the optimal one. The solutions proposed by an evolutionary algorithm allow for better SVM performances.

Therefore, we choose to use the evolutionary framework in order to discover the optimal expression of a new kernel and its parameters for several classification problems. The best (adapted) kernel is learnt by the algorithm itself by using the data of a particular problem. For this aim the Genetic Programming (GP) technique [12] is combined with an SVM algorithm [6, 11, 19, 15] within a two-level hybrid model. The GP-kernel is involved into a standard SVM algorithm to be trained in order to solve a particular classification problem. The optimal expression of a kernel is discovered by involving a guided search process based on genetic operations: the selection has to provide high reproductive chances to the fittest kernels, the crossover has to enable kernel-children to inherit quickly beneficial characteristics of their kernel-parents and the mutation has to ensure the diversity of the population and the exploration of the search space. After an iterative process, which runs more generations, an optimal evolutionary kernel (eK) is provided.

The paper is organized as follows: Section 2 outlines the theory behind SVM classifiers with a particular emphasis on the kernel functions. Section 3 describes the hybrid technique used in order to evolve the SVM kernels. This

is followed by a special section (Section 4) where the results of the experiments are presented. Section 5 describes some related work in the field of automated generation of kernel functions. Finally, Section 6 concludes our paper.

2. Support Vector Machine

2.1. **Generalities.** Initially, *SVM* algorithm has been proposed in order to solve binary classification problems [19]. Later, these algorithms have been generalized for multi-classes problems. 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 [19]. Learning the *SVM* means (implies) to minimize the norm of the weight vector (w in Eq. (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:

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

where the primal decision variables w and b define the separating hyper-plane.

The items that satisfy Eq. (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 Eq. (1), the soft margin formulation of *SVM* has introduced some slack variables $\xi_i \in \Re$ [5].

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 Eq. (2), which is the primal formulation of it:

$$\begin{array}{ll} \text{minimise}_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i & \\ \text{subject to:} & y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \forall i \in \{1, 2, \ldots, m\}. \end{array} \tag{2}$$

[1] where v^T represent the transpose of v

The coefficient C (usually called *penalty error* or *regularization parameter*) is a tuning parameter that controls the trade off between maximizing 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.

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

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

In Eq. (3), a_i denotes the Lagrange variable for the i^{th} constraint of Eq. (2).

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

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

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 [19]. However, in 1992, Boser, Guyon and Vapnik [2] have 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 [2]. 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 [16]. Note that all we required is the result of such an inner product. Therefore we do even not need to have an explicit representation of the mapping ϕ, neither to know the nature of the feature space. The only requirement is to be able to evaluate the kernel function on all the pairs of data items, which is much easier than computing the coordinates of those items in the feature space.

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 [13] - 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* [2].

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

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

In this case, Eq. (4) becomes:

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

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 Eq. (6). The selection of a kernel has to be guided by the problem that must be solved. 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 [18]; the most utilised of them by an *SVM* algorithm are listed in Table 1.

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	$K_{RBF}(x, z) = exp(-\sigma\|x - z\|^2)$	radial
Polynomial	$K_{Pol}\ (x, z) = (x^T \cdot z + coef)^d$	projective

3. Evolved Kernels

This section describes our approach for automatic design of kernels. The model's idea was initially proposed in [8] and here we try to detail it and to performed a more deeply analysis of the new evolved kernels. The model is a hybrid one: it uses *GP* [12] to construct positive and symmetric kernel functions, and optimizes a fitness function by using an *SVM* classifier (see Figure 1). A *GP* chromosome provides the analytic expression of such evolved kernels. The model we describe actually seeks to replace the expert domain

knowledge concerning the design of the *SVM*'s kernel function with a *GP* algorithm.

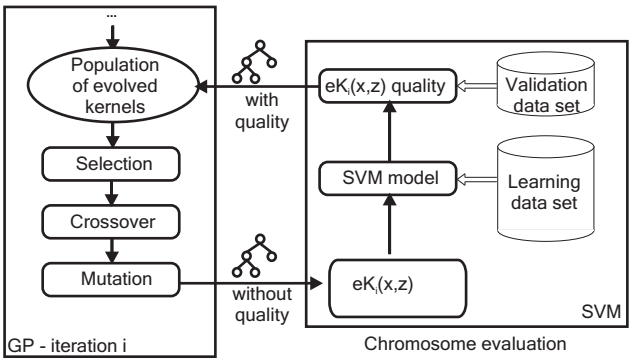

FIGURE 1. Sketch of the hybrid approach

Our hybrid model is structured on two levels: a macro level and a micro level. The macro level algorithm is a standard *GP* [12], which is used to evolve the mathematical expression of a kernel. The steady-state evolutionary model [17] 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 kernel functions). 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* [3] library. The original implementation of the *SVM* algorithm proposed in [3] allows using several well-known kernels (Polynomial, RBF and Sigmoid – see Table 1). In the numerical experiments, a modified version of this algorithm, which is based on the evolved kernel (eK) is also used. The quality of each *GP* individual is determined by running the *SVM* algorithm, which uses the eK encoded in the current chromosome ($K_{iter,ind}$ 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.

3.1. **Kernel representation.** In our model, the GP chromosome is a tree encoding the mathematical expression of the kernel function. Unlike a classic GP tree, our model uses a particular category of trees that can contain two types of nodes: scalar nodes and vectorial nodes. The terminal set is composed only by vectors from $\Re^d$: $VTS = \{x | x \in \Re^d\}$ (which correspond to the input data). Since a kernel function operates only on two samples the resulting terminal set VTS contains only two vector elements: x and z.

The function set (FS) contains two types of operations: scalar operations and vectorial ones. The scalar function set (SFS) contains several well-known binary ($+$, $-$, $\times$, $/$) and unary (sin, cos, exp, log) operators. The vectorial function set VFS (see Table 2) contains two types of primitive functions: operators that transform the initial multi-dimensional space into an $\mathcal{R}$ space (known as *norm* functions) and operators that preserve the dimensionality of the initial space. We also include several element-wise operations (EWOs) in this last function category.

TABLE 2. The vectorial function set - VFS: the norms and the element-wise operations

Type	**Elements**	**Definition**
Norms	EN	$EN(x,z) = \sum_{i=1}^{d}(x_i - z_i)^2$
	SP	$SP(x,z) = \sum_{i=1}^{d} x_i z_i$
	GN	$GN(x,z) = e^{-\gamma \times \sum_{i=1}^{d}(x_i - z_i)^2}$
EWOs	$\oplus$	$x \oplus z = v,\ v_i = x_i + z_i,\ i = \overline{1,d}$
	$\ominus$	$x \ominus z = v,\ v_i = x_i - z_i,\ i = \overline{1,d}$
	$\otimes$	$x \otimes z = v,\ v_i = x_i * z_i,\ i = \overline{1,d}$

Starting with a bottom-up tree reading, the functions operate in this way:

- the element-wise operations transform the d-dimensional space of input instances into another d-dimensional space.
- the norms (e.g., Euclidean, Gaussian, Scalar Product) transform the d-dimensional space into an one-dimensional space. The norms link the vector space with the scalar space in our GP tree.
- the scalar operations are used to combine the outputs of different norms.

Note that these EWOs are performed in a manner that preserves a valid dimension for the resulted vector. Our vectorial multiplication operation $\otimes$

is an element-wise operation; it is different from the cross product, which is a geometrical vector multiplication. The cross product performs the transformation $(\Re^d, \Re^d) \longmapsto \Re^d \times \Re^d$, or in our model we must have a transformation $(\Re^d, \Re^d) \longmapsto \Re^d$.

An example of a GP chromosome is depicted in Figure 2. The nodes that contain scalar symbols form a connex region. The rest of the nodes form one or more connex sub-regions.

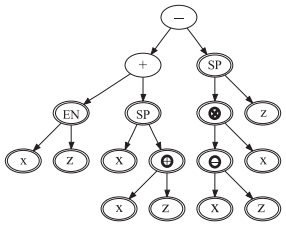

FIGURE 2. Example of a chromosome - the nodes designed by only a circle contain scalar symbols and the nodes designed by two circles contain vectorial symbols (functions and terminals). The vectors x and z, representing two data items, are the kernel inputs.

The *grow method* [1], which is a recursive procedure, is used to initialize a *GP* individual. This initialisation method 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 initialized either with another function or with a terminal. The initialization 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). 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 kernel.

3.2. **Genetic Operations.**

3.2.1. *Selection.* The selection operator chooses from the current population which individuals will act like parents in order to create the next generation of kernels. Selection has to provide high reproductive chances to the fittest kernels but, at the same time, it has to preserve the exploration of the search space. The choice of which kernels are allowed for reproducing determines which regions of the search space will be visited next. Indeed, achieving equilibrium between the exploration and the exploitation of the search space is very important for an evolutionary algorithm. When performing selection for

recombination, the kernels are compared by means of a fitness function that evaluates how good a potential solution is for the given problem.

3.2.2. *Crossover.* The crossover operator assures the diversity of the kernels and is performed in a tree-structure preserving way in order to ensure the validity of the offspring: first as a mathematical expression and second as a Mercer's kernel. The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents.

The model we describe uses an one-cutting point crossover [12]. Having two parent trees, we randomly choose a cutting point in the first parent, another cutting-point in the second parent and then, we exchange the subtrees rooted to these points. This crossover type has been used because it is able to guarantee a quite quickly convergence of the *GP* algorithm.

The cutting-point in the first parent is chosen randomly, but the other cutting-point is constrained by the position of the first one. Thus, if the first cutting-point is chosen right above a node that contains a scalar operator, then the other cutting point must be chosen above another scalar node (from the second parent). A similar procedure must be applied if the first cutting point is chosen right above a node that contains a vector operator.

Why we need this restriction? Because if we choose the first cutting point between a scalar function and a norm, then we must replace the sub-tree whose root is the norm with a sub-tree from the other parent. If the cutting-point in the second parent is chosen right above a node that contains a scalar function, then there are two possibilities: the $\Re$ function from the first parent acts on a sub-tree rooted by a scalar function or on a sub-tree rooted by a norm. These two possibilities are correct and they ensure a valid offspring. If the second cutting-point is chosen right above a node that contain a vector operator, then it is possible that scalar function from the first parent act on a vector (which is an impossible operation).

3.2.3. *Mutation.* The purpose of the mutation operator is to create new individuals by small and stochastic perturbations of a chromosome. Mutation operator aims to achieve some stochastic variability of an evolutionary algorithm in order to get a quicker convergence. Furthermore, the mutation operator aims to produce diversity of the population of candidate solutions and to reconsider the lost genetic material of the population. Mutation is therefore responsible for exploring new promising regions of the search space and not for exploiting those already discovered. This genetic operation, also as the crossover, preserves the syntactical validity of the new individual. For a *GP*-based kernel, 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.

3.3. **Fitness Assignment.** We must provide several information about the datasets, before to describe the chromosome evaluation. The data sample was randomly divided in two disjoint sets: a training set (80%) - for models building - and a testing set (20%) - for performances evaluation. The training set was than randomly partitioned into learning (2/3) and validation (1/3) parts.

The SVM algorithm kernel uses the learning subset to construct the SVM model and the validation subset for the evolved kernel performance assignment. The quality of an evolved kernel, which is the current GP chromosome, can be measured by the classification accuracy rate computed on the validation data set. The accuracy rate represents the number of correctly classified items over the total number of items. Note that we deal with a maximization problem: greater accuracy rates are, better evolved kernels are.

In order to evaluate the quality of a GP tree, it is also necessary to take into account if the expression encoded into the current chromosome is a valid kernel or not. We must verify therefore if the current expression satisfies the Mercer conditions [5] regarding the positivity and the symmetry of the Gram matrix associated to a kernel function. We have used the penalty method to involve these restrictions in the evaluation process. More exactly, two important steps are performed:

- (1) kernel positivity and symmetry verification - if a GP tree does not satisfy these conditions then the fitness of the GP tree will be 0; otherwise, we can go to step 2.
- (2) SVM algorithm running - there are two stages in this run: in the first stage the SVM algorithm embedding the evolved kernel is constructed by using the learning data; in the second stage, the SVM algorithm with the evolved kernel classifies the items from the validation set. The accuracy rate estimated on this subset will represent the quality of the GP chromosome.

4. Numerical experiments

In this section, several numerical experiments about evolving kernel functions for different classification problems are detailed. After evolving it on the validation set, the kernel is embedded into an SVM classifier, which is run on the test dataset. The SVM algorithm based on the classical kernels are also used to classify the test data set. Finally, the performances of different classifiers are compared.

Four data sets [9] are used in thee experiments. All the data sets contain information about the real-world problems (DS_1 and DS_2 – classification task is to determine whether a person makes over \$50K/year or not, DS_3 and DS_4 – classifying whether a web page belongs to a category or not). A binary classification problem must be solved in each of these cases. The structure of the problems is presented in Table 3.

TABLE 3. The structures of the data sets - each dataset is split into training set and testing set. For each of these two subsets it is specified: the total number of items, the number of items from the first class and the number of items from the second class, respectively

Data	**Training**			**Testing**		
set	**Total**	**1^{th} class**	**2^{nd} class**	**Total**	**1^{th} class**	**2^{nd} class**
a1a	1604	395	1209	30995	7446	23549
a2a	2264	572	1692	30295	7269	23026
w1a	2477	2404	73	47272	45864	1408
w2a	3470	3362	108	46279	44906	1373

The steady-state model [17] 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 *eK*s. The maximal depth of a *GP* tree is limited to 10 levels, which allows encoding till 2^{10} combinations. 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 [14]). Furthermore, several empirical tests indicated that the efficient kernel-trees do not expand to more than 10 levels. A binary tournament selection, a probabilistic crossover, and a probabilistic mutation are performed in order to obtain a new generation of chromosomes. The values of the crossover and mutation probabilities ($p_c = 0.8$ and $p_m = 0.3$) are chosen in order to assure a sufficient diversity of the population. The values used for the population size and for the number of generations have been empirically chosen based on the best results computed on the validation set. The soft margin hyper-parameter C, which weights the misclassification errors, has been set to 1 for all the classifiers used in our experiments.

4.1. **Experiment 1.** New kernel functions are evolved in this experiment. As we already mentioned, there are two different stages in this experiment: in the first stage the kernel expression is learnt and in the second stage the best evolved kernel function is tested.

We obtain various expressions of the kernel function, during different runs, all of them having a similar complexity.

Figure 3 depicts the evolution of the quality for the best evolved kernels along the number of iterations (for all the problems on the validation data sets). Only the values corresponding to the first 20 generations are depicted in this graphic for a better visualisation. Small improvements can be observed in the chromosomes quality (or in the accuracy rate) after the first 15 GP generations. This aspect is very important and it proves that the proposed model can discover an efficient kernel in only a few generations.

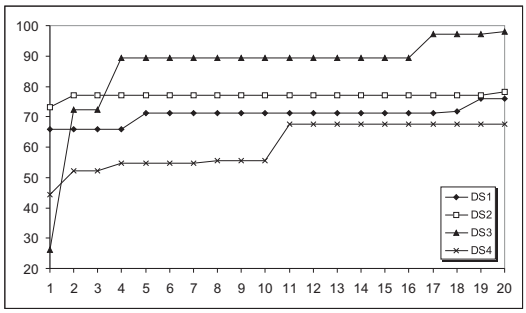

FIGURE 3. Evolution (on the validation set) of the best evolved kernel for all the problems.

Table 4 presents the accuracy rates estimated on the test data for each of the problem and the expression of the best evolved kernel (provided by the training process).

TABLE 4. The accuracy rates obtained on test datasets with an SVM algorithm that involves the best kernel expression learnt during the training stage.

Problem	**Acc**	**Kernel Expression**
DS_1	72.43	$K_1^*(x,z) = GN(z, x \ominus z) \times GN(x, z \ominus x)$
		$K_1^*(x,z) = e^{-\gamma \sum_{i=1}^{d} [z_i - (x_i - z_i)]^2} e^{-\gamma \sum_{i=1}^{d} [x_i - (z_i - x_i)]^2}$
DS_2	79.60	$K_2^*(x,z) = SP(z, x \otimes z \oplus x)$
		$K_2^*(x,z) = \sum_{i=1}^{d} z_i [x_i (z_i + x_i)]$
DS_3	89.39	$K_3^*(x,z) = GN(x,z) + SP(z,x)$
		$K_3^*(x,z) = e^{-\gamma \sum_{i=1}^{d} (x_i - z_i)^2} + \sum_{i=1}^{d} z_i x_i$
DS_4	90.27	$K_4^*(x,z) = SP(z, x \otimes z \oplus x)$
		$K_4^*(x,z) = \sum_{i=1}^{d} z_i [x_i (z_i + x_i)]$

4.2. **Experiment 2.** This experiment serves our purpose to compare the best evolved kernels to three commonly used kernels (the sigmoid kernel, the polynomial kernel and the radial basis function (RBF) kernel, respectively) and to the genetic kernel (GK) proposed in [10].

The SVM algorithm is run by the same error penalty as that from the evolving stage, but using the sigmoid kernel, the polynomial kernel, and the RBF kernel, respectively. For the Sigmoid kernel the parameter values are $\sigma = 0.01$ and $r = 0$, for the Polynomial kernel the degree d is set to 2 and for the RBF kernel the bandwidth value σ is 0.01. These values have been optimised by a parallel grid search method.

The results obtained by running the SVM with our evolved kernel (the second column), the Polynomial kernel, the RBF kernel, the Sigmoid kernel (the next three columns) and the evolved kernel described in [10] (the last column), respectively, are presented in Table 5.

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 Equation (7)).

$$\Delta I = 1.96 \times \sqrt{\frac{Acc(100 - Acc)}{N}}\%, \tag{7}$$

where N represents the number of test examples. In addition, Table 5 displays the corresponding confidence intervals (on the test set of each problem).

TABLE 5. The accuracy rates and their confidence intervals for each test data set using different kernel expressions.

Dataset	eK	K_{Pol}	K_{RBF}	K_{Sig}	GK
DS_1	72.43±0.50	73.40±0.49	71.28±0.50	71.28±0.50	75.62±0.48
DS_2	79.60±0.45	78.19±0.47	77.66±0.47	78.72±0.46	76.00±0.48
DS_3	89.39±0.28	74.24±0.39	84.47±0.33	37.12±0.44	88.25±0.29
DS_4	90.27±0.27	88.11±0.29	86.49±0.31	83.78±0.34	88.11±0.29

Table 5 shows that the accuracy rates computed by our hybrid approach are globally better than those computed by an SVM classifier that involves the classic kernels and also than the GK method proposed in [10] (in 3 cases out of 4). These results prove that evolving a new kernel adapted to the

classification problem is more promising than computing the results by using several well-known fixed kernels and picking the best. However, more extended experiments are need in order to validate our hybrid approach. Furthermore, it is clear (Table 5) that the evolved kernels perform better (from a statistically point of view) than the simple ones (the corresponding confidence intervals are disjointed) for three problems (out of four).

5. Related Work and Discussions

Evolutionary techniques have been used in the past in order to evolve complex functions in different domains. For instance, the expression of the crossover operator used by the genetic algorithms for function optimization [7] is only an example.

Although several methods [4, 21] have been proposed for optimizing the parameters of the kernel functions, to our knowledge, only one attempt, performed by Howley and Madden [10], yields of evolving the kernel function. The authors [10] have tried to find an optimal kernel function by using the genetic programming technique also. There are several and important differences between their approach and the model we describe. These differences regard: the function set, the terminal set and the Mercer conditions.

Howley and Madden have used only a few binary operators ($+$, $-$, $\times$), whereas we extend the function set by adding: several unary scalar operations (exp, sin, cos, log); several norm functions (EN, SP, GN - that transform the $\Re^d$ space into an $\Re$ space and create the link between the GP tree part reserved for the $\Re^d$ space and the GP tree part reserved for the $\Re$ space) and several element-wise operations ($\oplus, \ominus, \otimes$). Moreover, they have used the same function set for both type of operations (scalar and vectorial). The model we develop uses two different function sets: one for the scalar operations and one for the vectorial ones.

The terminals can be either vectors, or scalar values in the model proposed in [10]. Our approach uses only vectors as terminals in the GP tree. By contrast a trick has been used in [10]: the model decides the type of operators (scalar or vectorial) based on the type of arguments (if both arguments are scalar then the function is a scalar one and if at least one operand is a vector, then the vectorial meaning for the operator is used) - a bottom-up approach. The current model is a top-down one: it decides the type of operands based on the type of functions.

In the model we describe, the set of functions contains both scalar and vector operators which are used in order to generate valid kernel expressions (starting by the initialization stage and continuing by the crossover and mutation steps); this is different to the Howley's approach [10] were the correctness

of the kernel is imposed after the construction stage. The positivity and the symmetry of the evolved kernels learnt by our approach are verified when a kernel is evaluated (the expressions that do not satisfy these constraints are penalized). Unlike to this, in [10] the authors have first evaluated the kernel encoded into a GP tree on two samples x and z. These samples have been swapped and the kernel has been evaluated again. The dot product of these two evaluations has been returned as the kernel output. In this manner, the obtained kernels are symmetric, but there is no guarantee that they obey Mercer's theorem. Moreover, the dot-product operation has increased the kernel complexity.

Comparing to the standard SVM with a fixed kernel, the hybrid model we describe involves certainly more computations because of the additional GP step, which is needed in order to evolve an optimal kernel function. However, more computations are performed only for the training stage, whereas our hybrid model may give better accuracy when classifying the unlabelled data.

6. Conclusion and Further Work

A hybrid technique for evolving kernel functions has been described in this paper. The model has been used in order to discover and adapt (optimise) the mathematical expressions of the kernel function involved in an SVM algorithm used for binary classification problems.

We have performed several numerical experiments in order to compare our evolved kernel to others kernels (human designed or not). The numerical experiments have shown that the evolved kernels perform slightly better than the standard kernels (the sigmoid, the polynomial and the RBF kernels) or the genetic kernels proposed by Howley and Madden [10]. However, for a more pertinent conclusion regarding that the proposed method is a good one, it should be supported by a stronger statistical analysis and by a new set of experiments (especially for large data sets).

Further work will be focused on evolving better kernel functions, by taking into account different initialization strategies in order to improve the quality of the selected kernels and/or by using a multiple kernel approach.

References

[1] Banzhaf, W. Introduction. *Genetic Programming and Evolvable Machines 6*, 1 (2006), 5–6.

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

[3] Chang, C.-C., and Lin, C.-J. *LIBSVM: a library for SVM*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] Chapelle, O., Vapnik, V., Bousquet, O., and Mukherjee, S. Choosing multiple parameters for SVM. *Machine Learning 46*, 1/3 (2002), 131–159.
[5] Cortes, C., and Vapnik, V. Support-vector networks. *Machine Learning 20* (1995), 273–297.
[6] Cristianini, N., and Shawe-Taylor, J. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.
[7] Dioşan, L., and Oltean, M. Evolving the structure of the particle swarm optimization algorithms. In *EvoCOP 2006* (2006), J. Gottlieb and et al., Eds., vol. 3906 of *LNCS*, Springer, pp. 25–36.
[8] Dioşan, L., Rogozan, A., and Pécuchet, J.-P. Evolving kernel functions for SVMs by Genetic Programming. In *ICMLA'07, Ohio, USA* (2007).
[9] D.J. Newman, S. Hettich, C. B., and Merz, C. UCI repository of machine learning databases, 1998.
[10] Howley, T., and Madden, M. G. The genetic kernel SVM: Description and evaluation. *Artif. Intell. Rev 24*, 3-4 (2005), 379–395.
[11] Joachims, T. Making large–scale SVM learning practical. In *Advances in Kernel Methods — Support Vector Learning* (1999), B. Scholkopf, C. J. C. Burges, and A. J. Smola, Eds., MIT Press, pp. 169–184.
[12] Koza, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
[13] Mercer, J. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society 209* (1909), 415–446.
[14] Poli, R., Langdon, W. B., and McPhee, N. F. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008.
[15] Schoelkopf, B., and Smola, A. J. *Learning with Kernels*. The MIT Press, 2002.
[16] Schölkopf, B. The kernel trick for distances. In *NIPS* (2000), T. K. Leen, T. G. Dietterich, and V. Tresp, Eds., MIT Press, pp. 301–307.
[17] Syswerda, G. A study of reproduction in generational and steady state genetic algorithms. In *Proceedings of FOGA Conference* (1991), G. J. E. Rawlins, Ed., Morgan Kaufmann, pp. 94–101.
[18] Taylor, J. S., and Cristianini, N. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
[19] Vapnik, V. *The Nature of Statistical Learning Theory*. Springer, 1995.
[20] Vapnik, V. *Statistical Learning Theory*. Wiley, 1998.
[21] Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., and Vapnik, V. Feature selection for SVMs. In *NIPS* (2000), T. K. Leen, T. G. Dietterich, and V. Tresp, Eds., MIT Press, pp. 668–674.

LITIS, EA - 4108, INSA, Rouen, France and Computer Science Department, Babeş Bolyai University, Cluj Napoca, Romania
E-mail address: `lauras@cs.ubbcluj.ro`

LITIS, EA - 4108, INSA, Rouen, France
E-mail address: `arogozan@insa-rouen.fr`

LITIS, EA - 4108, INSA, Rouen, France
E-mail address: `pecuchet@insa-rouen.fr`