

A Hybrid Genetic Programming and Boosting Technique for Learning Kernel Functions from Training Data

Marta Gârdea

“Alexandru Ioan Cuza” University of Iași
Faculty of Computer Science
16 General Berthelot, Iași, România
marta@info.uaic.ro

Liviu Ciortuz*

“Alexandru Ioan Cuza” University of Iași
Faculty of Computer Science
16 General Berthelot, Iași, România
ciortuz@info.uaic.ro

Abstract

This paper proposes a technique for learning kernel functions that can be used in non-linear SVM classification. The technique uses genetic programming to evolve kernel functions as additive or multiplicative combinations of linear, polynomial and RBF kernels, while a procedure inspired from InfoBoost helps the evolved kernels concentrate on the most difficult objects to classify. The kernels obtained at each boosting round participate in the training of non-linear SVMs which are combined, along with their confidence coefficients, into a final classifier. We compared on several data sets the performance of the kernels obtained in this manner with the performance of classic RBF kernels and of kernels evolved using a pure GP method, and we concluded that the boosted GP kernels are generally better.

1. Introduction

In classification and clustering problems, the success of the classifier is determined, among other factors, by the quality of the similarity measure. Placing objects in classes with respect to their features requires a comparison between objects and a partition of the object space according to the results of this comparison. Classification problems may become more difficult if the similarity measure between objects to be classified is not obvious. Therefore, it is often useful to attempt to *learn* the similarity function, rather than use a default one that may not be appropriate in some cases.

This paper concentrates mostly on learning kernel functions that can be used by SVM classifiers [7, 26], in which direction few attempts have been made. Existing ap-

proaches to learning kernel functions use machine learning instruments such as genetic algorithms or genetic programming [8, 9, 25] or boosting [14] in order to design kernel functions that best explain the available data.

We propose a method that uses genetic programming [16] to evolve a kernel function as a combination of linear, polynomial and RBF kernels. A boosting procedure, inspired from InfoBoost, is used to help the evolved kernels concentrate on the most difficult objects to classify, and to obtain a final model with a low generalization error on unseen data.

The organization of this paper is as follows: section 2 describes the problem that the paper aims to solve: learning, from training data, kernel functions for SVMs; section 3 is a brief overview of other approaches having the same purpose; section 4 presents the proposed approach, which involves genetic programming and InfoBoost; some results are presented in section 5 and discussed in section 6; the paper concludes with section 7.

2. Context and problem description

2.1. Support Vector Machines

Support Vector Machines (SVM) [4, 23, 26, 27] are robust instruments for classification. Given the set S ($\{(x_i, y_i)\}, x_i \in \mathbf{R}^d, y_i \in \{-1, +1\}$), each element belonging to the positive (labeled +1) or to the negative (labeled -1) class, a separating hyperplane must be found for the two classes, maximizing the margin (the distance between the hyperplane and each class). The support vectors are the data points on the boundary of each class, the closest points to the separating hyperplane. The hyperplane can be computed as a combination of these vectors, and they are usually significantly less numerous than the entire set of available data points.

*This paper was published in Proceedings of SYNASC 2007, The 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timișoara, Romania, IEEE Conference Publishing, 2007.

2.2. Kernel functions

In real-life problems, we generally deal with classes that are not linearly separable in the given feature space. Non-linear SVMs are based on the idea that two classes which are not linearly separable in a space $X \subset \mathbb{R}^n$ may be linearly separable in a higher dimensional space, \mathbb{R}^m . A function $\phi : X \rightarrow \mathbb{R}^m$ can be used to map instances in the higher dimensional feature space. Considering that, when the separation hyperplane is computed, the instance vectors only appear in dot products. The “kernel trick” [1] is used to implicitly map input vectors to a higher dimensional space where an optimal separating hyperplane is constructed.

By definition, a *kernel* is a function K such that, for all $\mathbf{x}, \mathbf{z} \in \mathbf{X}$,

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$$

where ϕ is a mapping from \mathbf{X} into an inner product feature space \mathcal{F} . According to Mercer’s theorem [19], any continuous, symmetric, positive semi-definite kernel function $K(x, y)$ can be expressed as a dot product in a high-dimensional space. Based on this statement, several functions can be proved to be (Mercer) kernels and are commonly used in practice:

- Linear: $K(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x} \cdot \mathbf{x}' + c)$
- Polynomial: $K(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x} \cdot \mathbf{x}' + c)^q$, for γ, c and $q > 0$
- Radial Basis Function: $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$, for $\gamma > 0$
- Sigmoid: $K(\mathbf{x}, \mathbf{x}') = \tanh(\kappa \mathbf{x} \cdot \mathbf{x}' + c)$, for some (not every) $\kappa > 0$ and $c < 0$

Based on the consequences of the same theorem, in order to verify that a new symmetric function $K : X \times X \rightarrow \mathbb{R}$ is a kernel it suffices to check if it satisfies the requirement that the matrix defined by restricting the function to any finite set of points in X is positive semi-definite. This criterion can be applied to confirm that a number of new, more complicated kernels can be created from other simpler ones. Thus, if K_1 and K_2 are kernels over $X \times X$, with $X \subseteq \mathbb{R}^n$, $\mathbf{x}, \mathbf{z} \in X$, $a \in \mathbb{R}^+$, then the following are also kernels:

$$K(x, z) = K_1(x, z) + K_2(x, z) \quad (1)$$

$$K(x, z) = aK_1(x, z) \quad (2)$$

$$K(x, z) = K_1(x, z)K_2(x, z) \quad (3)$$

2.3. Choosing a kernel

Choosing the right kernel for a specific classification problem can be a difficult task. There is no universal kernel that can be used in any classification. Moreover, for some problems, where the classes are not just linearly unseparable, but also non-convex, with many border irregularities,

the choice of the kernel function becomes a delicate issue, since simple kernels, as well as kernels build “by hand”, based on human observations and intuition, may fail.

Naturally, in such cases, an automated search through the space of possible kernel functions should give a better answer to the optimal kernel problem. The main objective of this paper is to study methods of learning similarities (in this case, kernel functions that express such similarities) from the given data. Practically, we need a tool that can automatically find the best kernel function for the given data set.

3. Related work

There are several other approaches for learning kernels functions, the most notable of them involving boosting techniques and genetic programming.

KernelBoost [14] is an algorithm for learning kernel functions from labeled and unlabeled training data. It builds a strong kernel using boosting on a weak Gaussian Mixture Model (GMM) learner, constrained Expectation Maximization [24]. The boosting process is combined with a dissolving mechanism, which breaks difficult, non-convex classes into subclasses that are more likely to be modeled by a Gaussian Mixture Model.

The paper [25] proposes an **evolutionary strategy for multi-scale radial basis function kernels**. In order to obtain a flexible kernel function, a family of radial basis function (RBF) kernels is proposed. Multi-scale RBF kernels are combined by including weights. Then, the evolutionary strategies are used to adjust these weights and the widths of the RBF kernels.

Two more recent papers [8, 9] present a model for **evolving multiple kernel functions** for Support Vector Machines (SVM), which combines Genetic Programming and an SVM tool. The GP chromosome is a tree encoding the mathematical expression of the kernel function used by an SVM. The quality of a chromosome is the accuracy rate – i.e. the number of correctly classified items over the total number of items – which is obtained from an SVM classification that uses the kernel encoded by the given individual.

The results obtained with these methods are comparable with, or better than those achieved using simple kernels. This encourages further research in kernel learning.

4. InfoBoosted GP Kernels

As explained in section 2, kernels can be built from other kernels as linear combinations or products. Using *genetic programming* [16] and some training data management strategies inspired by *boosting* [3, 13, 22], a composite kernel is evolved from the available data, based on simple linear, polynomial and RBF kernels, aggregated in linear combinations or multiplications. The evolution process searches

for the appropriate structure in which the simple kernels are combined, as well as for the best values for the internal parameters.

The learning process can be summarized as follows:

- each *individual* is a kernel, obtained as a combination of some basic kernels;
- an evolutionary process builds *new kernels* using genetic operators and encourages the propagation of the best features among the generations;
- training instances are sampled according to a *distribution* that emphasizes *the most difficult instances* at the moment; a sampling is used for several generations (a boosting round) as the environment that must be learned;
- the *evaluation* of a kernel consists of obtaining a classification over the sample using that kernel, and computing a “confidence” value for the individual as the correctness of the classification;
- the *best individuals* from each round are kept (offline) if they perform better than an established confidence threshold θ ;
- *update the distribution* over the training data according to how the instances are classified using the best kernel of the round.

Several papers that propose a combination of boosting and genetic programming already exist, most of them choosing AdaBoost as the learning meta-algorithm: [15, 17, 20]. Our work explores the possibility of using ideas from InfoBoost [3], with the purpose of getting a better evaluation of the weak learner’s outcome.

4.1. Genetic Programming elements

Genetic Programming [16] uses principles of darwinian evolution, such as genetic recombinations and survival of the fittest, in order to automatically discover computer programs that perform a user-defined task. Like other evolutionary techniques, this method uses a population of individuals (also called chromosomes), each individual representing a computer program. Several transformations are applied to these chromosomes for obtaining a new population: crossover, mutation, reproduction, gene duplication, gene deletion.

Candidate kernel representation

The individual structures that undergo adaptation in genetic programming are computer programs, traditionally represented in memory as tree structures. The size, the shape and the contents of these computer programs can dynamically change during the process. This aspect is particularly suited to the goal of the kernel function search process. We look

for a kernel function, not knowing in advance the shape or the complexity of this function. Not only parameters need to be found, but also the structure and compoency of the function itself.

The set of possible structures in genetic programming is the set of all possible compositions of functions that can be composed recursively from a set of functions $F = \{f_i\}$ and a set of terminals $T = \{t_i\}$. The choice of the F and T sets strongly depends on the problem that needs to be solved. In the case of kernel function search, the candidate solutions will be mathematical expressions of the kernels.

As explained in section 2, there are several operations that can help building new kernel functions from other kernel functions. We will further make use of the three rules, (1), (2) and (3), presented in that section. Consequently, the function set will contain the two arithmetic operations + and *, each with 2 arguments.

The terminal set will contain the nodes that represent some popular kernels used in practice: *LK* (linear kernel), *PK* (polynomial kernel) and *RK* (RBF kernel), along with randomly generated scalars. *LK* has 2 scalar parameters, *PK* has 3, and *RK* has one, necessarily positive, as presented above. The scalars are Ephemeral Random Constants (ERC), as defined in [16], generated according to an exponential distribution in the range $[e^{-10}, e^{10})$, except q , which is an integer in the range $[1..100]$.

Genetic Operations

New candidate solutions from the search space are generated in genetic programming (like in all evolutionary methods) by means of genetic operators. For each operator, some validity constraints need to be taken into consideration, in order to obtain correct expressions for the kernels.

The *crossover* operation for genetic programming produces two new offsprings that are built of parts taken from each of the two parents. It consists of independently selecting a random point in each parent as the cut point for crossover, and switching between parents the rooted subtrees that lie below the cut points.

Another genetic operator, called *internal crossover*, acts on a single individual, by randomly choosing two cut points and swapping the respective subtrees within the individual’s structure.

The *mutation* operation, which induces random changes in the individuals, ensuring diversity, is present in our method in three variants:

- a classic mutation operator on tree structures, that randomly chooses a node in the tree and replaces the whole rooted subtree that starts in the chosen node with a randomly generated subtree.
- a *one node mutation* operator, that randomly chooses a node in the tree and replaces it with another node, leaving

the rest of the subtree unchanged.

- a *random number mutation*, that only operates on the values of the scalars in the expression tree.

Evaluation

The fitness function of an individual is a measure of the quality of that individual. The higher the fitness, the better adapted the chromosome, and, consequently, in this context, the better the kernel function it represents.

The fitness of a kernel function is here given by the accuracy of the classification performed by a non-linear support vector machine that uses that kernel for the implicit mapping of the instances in the dataset. More precisely, an SVM is trained and tested on the given data set. The 4-fold cross-validation result of the classification, expressed as the number of correctly classified instances, is the fitness of the evaluated individual.

A very important detail of the evaluation process is the compoency of the data set each individual is tested against. This is one of the aspects where *boosting* ideas are involved. At each round (group of generations), the individuals are evaluated on a different distribution over the data set, which will further result into the more probable survival of the kernels that were able to ensure the right classification of the instances that are usually misclassified.

The next section will give a brief overview of the boosting methods and provide technical details about the actual computation of this distribution.

4.2. Boosting elements

Boosting – An overview

Practice shows that, for classification problems, finding a single strong rule that is valid for all (or at least almost all) data is considerably more difficult than finding several weak rules (*rules of thumb*), with moderate accuracy, that only apply in some cases. Moreover, the set of rules of thumb found with a simple method can be combined into a complex, strong rule, of high overall accuracy. This idea led to the boosting method and its variations.

Boosting [12, 22] is a machine learning meta-algorithm for supervised learning, mostly used in classification. It is based on *weak learners* – classifiers with an accuracy somewhat better than random – and by combining their results, it obtains an accurate classifier. The term *to boost* suggests the purpose of this method: to grow, to enhance the accuracy of a learning algorithm.

The weak learner is trained at each boosting round on a different set of data, or on a different distribution, in order to obtain different classifiers that can separate the data in some cases. In the end, the obtained weak rules are combined into a single strong rule.

Within this general structure, some particular aspects must be specified: the choice of the distribution over the training data at each round, and the method for combining the weak rules into a strong rule.

InfoBoost

InfoBoost [3] was proposed as a modification of AdaBoost [13] on the computation of the confidence in the weak classifier at each boosting round. AdaBoost (adaptive boosting) proposed that the distribution on each boosting round can be obtained by associating higher probabilities to the training instances that were misclassified, thus forcing the learner to concentrate on the difficult examples, while the strong rule can be the weighted vote combination of the weak rules, with weights – confidence coefficients – depending on the training error of the weak classifier. In [3] Aslam suggested that the qualitative and quantitative performance of a predictor is not entirely captured by error. Two weak hypotheses with the same error on the current distribution can actually have divergent behaviors, such as a lower false negative rate at the expense of a high positive rate and vice versa. Exploiting more of the information that can be obtained about the weak learner’s performance is the purpose of InfoBoost (see figure 1), which proposes different confidence values for the positive and for the negative answer given by a weak hypothesis respectively.

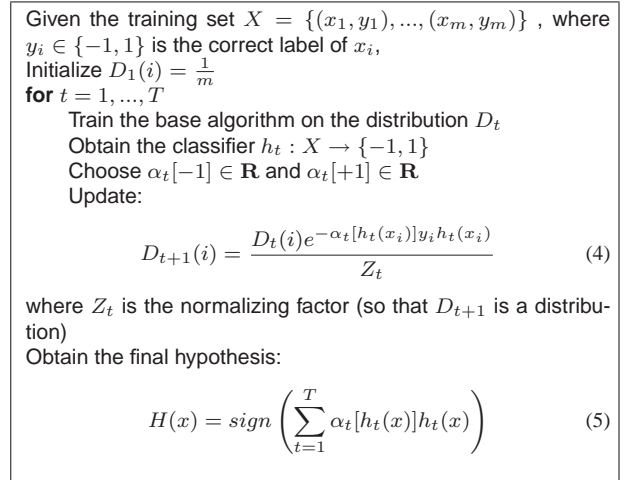


Figure 1. The InfoBoost algorithm

The choice of $\alpha_t[\cdot]$, the confidence factor, is given by the following expressions, in which for simplicity we omit the indices t for α , r , h and D :

$$\alpha[-1] = \frac{1}{2} \ln \left(\frac{1 + r[-1]}{1 - r[-1]} \right) \quad (6)$$

$$\alpha[+1] = \frac{1}{2} \ln \left(\frac{1 + r[+1]}{1 - r[+1]} \right) \quad (7)$$

where

$$r[-1] = \frac{\sum_{i:h(x_i)<0} D(i)y_i h(x_i)}{\sum_{i:h(x_i)<0} D(i)} \quad (8)$$

$$r[+1] = \frac{\sum_{i:h(x_i)\geq 0} D(i)y_i h(x_i)}{\sum_{i:h(x_i)\geq 0} D(i)} \quad (9)$$

InfoBoost-ing the evolved kernels

InfoBoost influences the learning process under two main aspects. The first one was mentioned in section 4.1: at different generations, the individuals are evaluated on different samplings of the data set, extracted according to the current distribution. The motivation is clearly stated by the creators of the boosting methods: helping the learning algorithm to discover and focus on difficult parts of the classification task. From various existing boosting methods, InfoBoost promises to extract most rigorously the information about the classification accuracy that can be obtained with the learned kernel.

The distribution update is given by equation (4). If the training instance is misclassified, i.e. $h_t(x_i) \neq y_i \Rightarrow y_i h_t(x_i) = -1$. Its weight will be multiplied by the factor $e^{-\alpha[h_t(x_i)]y_i h_t(x_i)} > 1$, which means a relative increase of the weight at the next round. Similarly, an instance which is correctly classified will have its weight decreased.

The confidence factors, computed according to equations (6) and (7), are, at an intuitive level, higher when more of the negative instances (or positive, respectively) are correctly classified.

The second influence of InfoBoost is the actual result of the learning process. Generally, in genetic algorithms, the given solution is either the best individual in the last generation, or the best individual ever encountered in the evolution process. In this case, since in different epochs of the evolution the individuals had different objectives, i.e. better classifying a certain subset of the available data, it is natural to combine the efforts of the best individuals that achieved good results on different data sets. Consequently, the weighted vote combination proposed in InfoBoost is used. More precisely, after each weak learner training round, we accept the evolved kernel if its confidence coefficient (either $\alpha[+1]$ or $\alpha[-1]$) exceeds an established threshold. With the obtained kernel we train an SVM on the available training data. The models obtained after training the SVMs, along with the respective confidence coefficients, form the final model that will provide the classification of a new instance according to their weighted vote.

4.3. The InfoBoost GP learning algorithm

The steps of the learning process are presented in figure 2. Some steps are labeled [IB] or [GP], in order to high-

light which of the methods is “active” at that point.

Input: A training set $X = \{(x_1, y_1), \dots, (x_m, y_m)\}$, where $y_i \in \{-1, 1\}$ is the correct label of x_i

[IB] *Input data distribution initialization*, for the boosting component: $D(i) = 1/m$

[GP] *Population initialization*

[GP] $Final_Model = \{\}$

[GP] *Initial population evaluation*, a subroutine that makes use of the data set distribution

while (not stop)

while (train weak learner)

 [GP] *Selection (tournament)* of the next generation

 [GP] *Alteration with genetic operators*

 [GP] *Evaluation*.

 [IB] Compute the error $\epsilon_t^i[+1], \epsilon_t^i[-1]$ and the confidence $\alpha_t^i[+1], \alpha_t^i[-1]$, for each individual ($\epsilon_t^i[\cdot] = 1 - r_t^i[\cdot]$)

 [GP] Find the best individual – $best_t$

if ($\alpha_t^{best_t}[-1] > \theta$ or $\alpha_t^{best_t}[+1] > \theta$)

then

 [IB] Add $(\alpha_t^{best_t}[\pm], K^{best_t})$ to $Final_Model$

 [IB] Update the distribution $D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t[best_t(x_i)]y_i best_t(x_i)}}{Z_t}$

Output: $Final_Model$

Figure 2. InfoBoost GP for Kernel Learning

The basic idea consists of inserting boosting principles into the genetic algorithm that produces new kernels. We obtain a hybrid method that can be perceived as InfoBoost with GP as a weak learner.

Practically, the task is to train a classifier – an SVM with a proper kernel function – to distinguish between two classes, based on the information it can extract from a set of labeled samples $X = \{(x_1, y_1), \dots, (x_m, y_m)\}$, where $y_i \in \{-1, 1\}$ is the correct label of x_i .

The input data must be prepared to accept distribution variations during the learning process. Each instance will have an associated weight, which, at an intuitive level, expresses how difficult it is to properly classify that instance. Initially, no such information is available, so all instances have the same weight. The distribution is thus initialized with $D(i) = 1/m$.

The kernel function is evolved using genetic programming, a process that has several classic steps: the initialization of the candidate solution population, the evaluation of the generated individuals on the initial distribution, the selection procedure for the current individuals, the alteration phase (using the genetic operators presented in section 4.1), and a reevaluation of the population according to the current distribution over the training data. The last three steps are repeated for a number of generations or until convergence (this is the stopping criterion in Figure 2).

The GP acts as a weak learner for InfoBoost on periods of consecutive iterations, during which the kernels are evaluated on the same training subset. After this local training period, the kernel that performs best over the target distribution is chosen. The smallest tree is preferred in case of a

tie. If this individual’s quality, expressed as the confidence parameters (equations (6) and (7) from the InfoBoost section) is better than some threshold θ , then this individual is accepted as a component of the final model. The distribution is updated according to equation (4), considering the classification of the best individual.

4.4. Implementation details

Two open-source research tools were used in the implementation: ECJ [18], a research Evolutionary Computing system written in Java, and LIBSVM [6, 11], an integrated software for support vector classification. In the former, a Boosted GP module was developed, extending the provided basic elements for genetic programming problems, for defining the target problem, and customizing the input data, the tree nodes, and the fitness function. LIBSVM was extended in order to support custom kernels, integrated into the evolutionary computing platform, and called from the evaluation function in the kernel evolution process. Preliminary experimentation with the two support vector optimization methods provided by LIBSVM – C-SVM and nu-SVM – and various parameters for them, led to the choice of C-SVM with $C = 1$. In the next section, the results obtained with our method are compared to the results obtained using an RBF kernel, whose parameters were optimized by the brute-force grid search tool, also available in LIBSVM.

5. Results

The proposed method was tested on several databases for binary classification problems, available in the UCI ML Repository [10]: *heart_scale* with 270 samples and 13 numeric features, *breast_cancer* with 683 samples and 9 features, and *wdbc*, (*Wisconsin Diagnostic Breast Cancer*), with 570 samples and 30 features.

Another test problem for our method was the classification of MSD domains.¹ The domains – subsequences of amino-acids within the sequence of a protein, that carry out the function of the protein – can suffer significant structure modifications during evolution, while still conserving their function. Domains can be grouped in functional families, themselves divided in subfamilies. The MSD (Membrane Spanning Domains) super-family and some of its subfamilies show only little global-sequence conservation, hence alignment tools have a poor performance on recognizing such domains [21].

The protein domains database contains 1234 entries, each with 570 features. The representation of a protein

¹This work is part of a project developed in collaboration with the Databases and Machine Learning team at Laboratoire d’Informatique Fondamentale, Université de Marseille, France [5].

sequence is composed of the *blastp* [2] alignment scores with the positive proteins in the database.

For the tests, we used the following configuration:

```

pop_size = 70 individuals in each generation
tournament selection
mutation rate: p_m = 0.15
one node mutation rate: p_onm = 0.07
random number mutation rate: p_rnm = 0.01
crossover rate: p_c = 0.4
internal crossover rate: p_ic = 0.05.

```

The training set consisted of 88% percent of the data, while 12% were left for testing. 4-fold-cross-validation was performed at the evaluation of each individual kernel.

To obtain a weak kernel, we trained the weak learner for a number of generations on a sample of an established size, chosen according to the current distribution. As expected, the sample size has major effects on the accuracy of the weak learner. Details will be discussed in section 6.

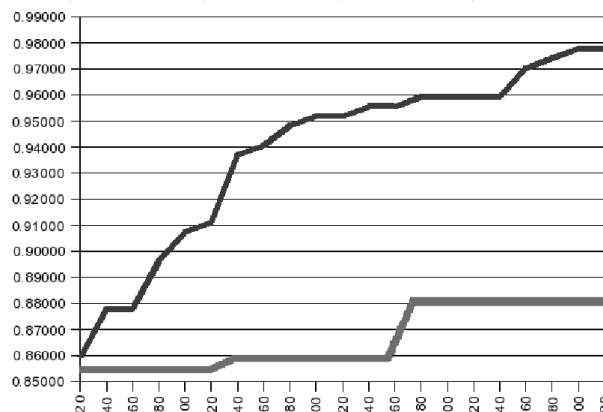


Figure 3. Accuracy growth at each generation on the *heart_scale* database for simple GP and InfoBoosted GP

Database	<i>heart_scale</i>
Generations	20/600
Samples	150/216
Final training accuracy	
with InfoBoosted kernel	100%
Final validation accuracy	
with InfoBoosted kernel	81%
Best RBF training accuracy	100%
Best RBF validation accuracy	54%

Table 1. Results on *heart_scale*

6. Discussion

Training error. When comparing the evolution of the best individual in a boosted GP, in relation with a pure GP,

Database	breast_cancer		
Generations	30/900	20/600	20/600
Samples	100/606	200/606	300/606
Final training accuracy with InfoBoosted kernel	100%	100%	100%
Final validation accuracy with InfoBoosted kernel	94.81%	94.81%	94.81%
Best RBF training accuracy		100%	
Best RBF validation accuracy		87.01%	

Table 2. Results on breast_cancer

Database	wdbc		
Generations	30/900	20/600	20/600
Samples	100/501	200/501	300/501
Final training accuracy with InfoBoosted kernel	100%	100%	100%
Final validation accuracy with InfoBoosted kernel	96.87%	98.43%	95.31%
Best RBF training accuracy		96.84%	
Best RBF validation accuracy		75.71%	

Table 3. Results on wdbc

Database	protein	
Generations	20/1600	30/900
Samples	100/1170	200/1170
Final training accuracy with InfoBoosted kernel	99.91%	100%
Final validation accuracy with InfoBoosted kernel	92.31%	95.38%
Best RBF training accuracy		99.49%
Best RBF validation accuracy		80.1%

Table 4. Results on protein domains

we notice a slight improvement tendency for the pure GP method, and a more pronounced growth of the accuracy in the boosted process (figure 3).

Generalization error. Often, with sufficient boosting rounds, on noise-less data, the training error will eventually reach zero, or a value close to zero. Such cases are not necessarily good, since they may result from overfitting the training data. Even with the encouragement of shorter kernels, the weak classifiers may be tempted to explain in detail the training data, but eventually unable to generalize, and thus perform poorly on unseen data. We notice that the generalization error is, as expected, strongly dependent on the training sample size. Sacrificing time efficiency, we should provide reasonably large samples to the weak learner, in order to obtain good results on generalization.

A small sample results in an apparently good weak kernel with respect to the training subset, but overfitted, and with poor performance when integrated in the final model and tested against the entire data set. Using small samples results in many kernels identifying only small chunks of data, until all the training instances have been covered by at least one such kernel.

On the other hand, larger samples tend to reduce the ge-

Samples	16	200	300	500
Validation accuracy	93.75%	98.43%	95.31%	84.37%

Table 5. Validation accuracy with different sample sizes, on the wdbc database

neralization capability of the learned SVM. Forcing the generalization on a very large dataset, the opposite effect is obtained, by promoting kernels whose corresponding hyperplane is obtained from most of the instances as support vectors, and which perfectly “surrounds” the given samples. Table 5 illustrates these statements on the wdbc database. We note that regardless of the sample size, the training accuracy was 100%.

Support vector machines trained with RBF kernels expose large variations for the training error / validation error ratio. If the SVM is forced to achieve high accuracy on the training data, the accuracy on the validation set may decrease drastically. For example, on the heart_scale database, the validation error can reach 46% for 0% training errors, and a 6% validation error for a 2% training error.

Results versus time. The results exposed in this section support the choice of a kernel learned from the data. However, this is not a good option in case a very quick answer is needed. The accuracy is indeed significantly boosted, but the run time (on an AMD64 CPU, 1.8GHz) varies from tens of minutes to several hours, depending on the complexity of the problem.

7. Conclusions and future work

This paper discussed the necessity of learning inter-instance similarities from data in the context of non-trivial classification tasks and presented a method for learning kernel functions that can be used in non-linear SVM classification. We propose a method that uses genetic programming to evolve a kernel function as an additive or multiplicative combination of linear, polynomial and RBF kernels. A boosting-like procedure, inspired from InfoBoost, is used to help the evolved kernels concentrate on the most difficult objects to classify.

Comparing the results obtained using our method on one side and the results obtained by SVMs employing RBF kernels or kernels evolved by simple GP on the other side, we find a lower training and generalization error for the formers. Moreover, we noticed that the boosting component is indeed effective, since the generalization error tends to decrease at each boosting round, along with the training error.

Future work includes some enhancements of the current method and the development of more advanced methods for the chosen problem. So far, only 2-class problems were

approached with the proposed method. A multiclass version needs to be adapted and tested.

Some scalability issues must also be taken into account. The learning process is complex, involving a costly evolutionary algorithm, with sufficiently many generations so that it can allow the individuals to discover and adapt to the difficult instance set, and an even more time expensive SVM training process for the evaluation of each individual.

Another – more complex – direction concerns the possibility of designing kernels that have different expressions for different areas of the instance space. A kernel function's value gives some similarity measure for its two arguments. In general, distance or similarity functions tend to be considered the same in the entire space. However, practice shows that some subspaces may have different properties, making it difficult to find a general rule. Therefore, the possibility of defining kernels whose expression captures the local particularities of the space, instead of trying to approximate them in a general expression, should be explored.

References

- [1] M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25(821-837):17, 1964.
- [2] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215(3):403–410, 1990.
- [3] J. A. Aslam. Improving algorithms for boosting. In *COLT '00: Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 200–207, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
- [4] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.
- [5] C. Capponi, G. Fichant, Y. Quentin, and F. Denis. Classification of Domains with Boosted Blast. Technical report, LIF, 2005.
- [6] C. Chang and C. Lin. LIBSVM: a library for support vector machines, software available at: www.csie.ntu.edu.tw/~cjlin/libsvm, 2001.
- [7] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.
- [8] L. Dioşan and M. Oltean. Evolving kernel function for support vector machines. In C. C., editor, *The 17th European Conference on Artificial Intelligence, Evolutionary Computation Workshop*, pages 11–16, 2006.
- [9] L. Dioşan, M. Oltean, A. Rogozan, and J. P. Pecuchet. Genetically designed multiple-kernels for improving the SVM performance. In *GECCO 2007*. ACM, 2007.
- [10] D.J. Newman, S. Hettich, C.L. Blake and C.J. Merz. UCI Repository of machine learning databases, 1998.
- [11] R. Fan, P. Chen, and C. Lin. Working Set Selection Using Second Order Information for Training Support Vector Machines. *The Journal of Machine Learning Research*, 6:1889–1918, 2005.
- [12] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, 148:156, 1996.
- [13] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.
- [14] T. Hertz, A. Hillel, and D. Weinshall. Learning a kernel function for classification with small training samples. *Proceedings of the 23rd international conference on Machine learning*, pages 401–408, 2006.
- [15] H. Iba. Bagging, boosting, and bloating in genetic programming. *Proceedings of the Genetic and Evolutionary Computation Conference*, 2:1053–1060, 1999.
- [16] J. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press Cambridge, MA, USA, 1992.
- [17] W. Langdon and B. Buxton. Genetic programming for combining classifiers. *Proceedings of GECCO-2001*, pages 66–73, 2001.
- [18] S. Luke, L. Panait, G. Balan, Z. Skolicki, J. Bassett, R. Hubley, and A. Chircop. ECJ: A Java-based evolutionary computation and genetic programming research system, <http://cs.gmu.edu/~eclab/projects/ecj/>.
- [19] J. Mercer. *Functions of positive and negative type and their connection with the theory of integral equations*. Philos. Trans. Roy. Soc. London, 1909.
- [20] G. Paris, D. Robilliard, and C. Fonlupt. Applying Boosting Techniques to Genetic Programming. *Artificial Evolution 5th International Conference, Evolution Artificielle, EA*, 2310:267–278, 2001.
- [21] Y. Quentin, J. Chabalier, and G. Fichant. Strategies for the identification, the assembly and the classification of integrated biological systems in completely sequenced genomes. *Comput Chem*, 26(5):447–57, 2002.
- [22] R. Schapire. The boosting approach to machine learning: An overview. *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.
- [23] B. Schölkopf, A. Smola, C. Burges, and R. Soentpiet. *Advances in Kernel Methods: support vector learning*. MIT Press, 1999.
- [24] N. Shental, A. Bar-Hillel, T. Hertz, and D. Weinshall. Computing Gaussian mixture models with EM using equivalence constraints. *Advances in Neural Information Processing Systems*, 16:465–472, 2004.
- [25] B. K. Tanasane Phientrakul. Evolutionary strategies for multi-scale radial basis function kernels in Support Vector Machines. In *GECCO'05*, pages 905–911, 2005.
- [26] V. Vapnik. *Statistical learning theory*. Wiley, 1998.
- [27] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24(3):774–780, 1963.