Applied Soft Computing 12 (2012) 2550–2565

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A dynamic model selection strategy for support vector machine classifiers

Marcelo N. Kapp[a,*], Robert Sabourin[a], Patrick Maupin[b]

[a] École de technologie supérieure, Université du Québec, Canada
[b] Defense Research and Development Canada - Valcartier (DRDC Valcartier), Canada

ARTICLE INFO

Article history:
Received 6 October 2010
Received in revised form 22 February 2011
Accepted 7 March 2011
Available online 16 April 2012

Keywords:
Support Vector Machines
Model selection
Particle Swarm Optimization
Dynamic optimization

ABSTRACT

The Support Vector Machine (SVM) is a very powerful technique for general pattern recognition purposes but its efficiency in practice relies on the optimal selection of hyper-parameters. A naïve or *ad hoc* choice of values for these can lead to poor performance in terms of generalization error and high complexity of the parameterized models obtained in terms of the number of support vectors identified. The task of searching for optimal hyper-parameters with respect to the aforementioned performance measures is the so-called SVM model selection problem. In this paper we propose a strategy to select optimal SVM models in a dynamic fashion in order to address this problem when knowledge about the environment is updated with new observations and previously parameterized models need to be re-evaluated, and in some cases discarded in favor of revised models. This strategy combines the power of swarm intelligence theory with the conventional grid search method in order to progressively identify and sort out potential solutions using dynamically updated training datasets. Experimental results demonstrate that the proposed method outperforms the traditional approaches tested against it, while saving considerable computational time.

1. Introduction[1]

Support Vector Machines (SVMs) are very powerful classifiers in theory, but their efficiency in practice relies on the optimal selection of hyper-parameters. A naïve or *ad hoc* choice of values for hyper-parameters can lead to poor performance in terms of generalization error and high complexity of the parameterized models obtained in terms of the number of support vectors identified. The task of searching for optimal hyper-parameters with respect to the aforementioned performance measures is the so-called SVM model selection problem.

Over the last years, many model selection approaches have been proposed in the literature. They differ basically in two aspects: the selection criterion and the searching methods used. The selection criterion, i.e. the objective function, is a measure that guides the search. Some of them are specifically related to the SVM formulation, such as: radius margin bound [2], span bound [3], and support vector count [4]. Others are classical ones, such as the well-known cross-validation and hold-out estimations. On the other hand, the most common searching methods applied are: gradient descent [5–7], grid-search [8–10], or evolutionary techniques, such as genetic algorithms (GA) [11–14], covariance matrix adaptation evolution strategy (CMA-ES) [15], and more recently, Particle Swarm Optimization (PSO) [16,17]. Although some of these methods have practical implementations, e.g. gradient descent, their application is usually limited by hurdles in the model selection process. For instance, the gradient descent methods require a differentiable objective function with respect to the hyper-parameters and the kernel, which needs to be differentiable as well. In this same vein, multiple local minima in objective functions also represent a nightmare for gradient descent based methods. To overcome this, the application of grid-search or evolutionary techniques is a very attractive option. Unfortunately, concerning the grid-search method, a good discretization of the search space in fixed values is crucial to reach high performances. So, the choice of objective function, the presence of local minima in the search space, and the computational time required for model selection task have been considered the main challenges in the field.

In addition to the typical parameter estimation difficulties briefly mentioned above, the availability of updates on the knowledge related to the pattern recognition problem to be solved represents a challenge too. These updates typically take the form of data arriving in batches which become available for updating the classification system. In fact, the quality and dynamics of training data can affect the general model selection process in different ways. For example, if knowledge on the problem is limited, or the data are noisy or are arriving in batches over time, the model

* Corresponding author.
E-mail addresses: kapp@livia.etsmtl.ca (M.N. Kapp), robert.sabourin@etsmtl.ca (R. Sabourin), Patrick.Maupin@drdc-rddc.gc.ca (P. Maupin).

[1] This paper is an extension in terms of experimental results, analysis depth and explanations of our first study introduced in [1].

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2012.04.001

selection task and its performance can progressively degrade. In order to avoid the negative effects of uncertainties associated with either the training data or the updates, we believe that an efficient option is to allow on-line re-estimation of the current model's fitness and if required to allow the production of a new classification model more suitable to both historical and new data. This is important because, if the goal is to obtain a performing single classifier, the model selection process must be able to select dynamically optimal hyper-parameters and train new models from new samples added to existing batches. In this work, we propose to study the general SVM model selection task as a dynamic optimization problem in a gradual learning context, where solution revisions are required online to either improve existing models or re-adapted hyper-parameters to train new classifiers from incoming data. These considerations are especially pertinent in applications for which the acquisition of labeled data is expensive, e.g. cancer diagnosis, signature verification, etc., in which case the data available may initially not be available in sufficient quantity to perform an efficient model selection. However, more data may become available over time, and new models can gradually be generated to improve performance. In contrast, as previously mentioned, not only is the optimality of the models estimated a relevant factor, but also the computational time spent to search for their parameter values. Most of related work in the literature has considered cases involving only a fixed amount of data in systems aimed at producing a single best solution. In these approaches whenever the training set is updated with more samples, the entire search process must be restarted from scratch.

In this paper we present a Particle Swarm Optimization (PSO) based framework to select optimal models in a dynamic fashion. The general concept underlying this approach is to treat the SVM model selection process as a dynamic optimization problem, which can have multiple solutions, since its optimal hyper-parameter values can shift or not over the search space depending on the data available on the classification problem at a given instant. This means that the proposed method can also be useful for real-world applications requiring the generation of new classifiers dynamically in a serial way, e.g. those involving streaming data. The key idea is to obtain solutions dynamically over training datasets via three levels: re-evaluations of previous solutions, dynamic optimization processes, or even by keeping the previous best solution found so far. In this way, by shifting among these three levels, the method is able to provide systematically adapted solutions. We implement the proposed method based on three main principles: change detection, adapted grid-search, and swarm intelligence theory (for self-organization capability), where the goal is to solve the model selection by overcoming the constraints of the methods described above.

In addition, we try to answer the following questions: Is PSO really efficient to select optimal SVM models? Can the proposed method be more efficient than the traditional grid-search or even a PSO based strategy? Is it possible to obtain satisfactory results by spending less computational time than is required for the application of PSO for each set of data? What is the impact in terms of classification errors, model complexities, and computational time for the most promising strategies? Experimental results demonstrate that our method can outperform the model selection approaches tested. Moreover, we show that for datasets with a fair number of samples, the gradual application of the proposed method over sets of data can achieve results similar to those obtained by optimization processes with PSO over all data, but with a considerable saving of computational time.

This paper is organized as follows. In Section 2 we briefly describe the support vector machine classifier method. In Section 3 we explain the relation between the model selection problem and dynamic optimization problems. Our proposed method is introduced in Section 4. Finally, the experimental protocol and results are described in Section 5 and our conclusions in Section 6.

2. Support Vector Machines

The Support Vector Machine (SVM) classifier is a machine learning approach based on the structural risk theory introduced by Vapnik in [4]. In particular, an SVM classifier is capable of finding the optimal hyperplane that separates two classes. This optimal hyperplane is a linear decision boundary which separates the two classes and leaves the largest margin between the samples of the two classes. Moreover, unlike most learning algorithms based on empirical risk, the SVM does not depend on probability estimation. This characteristic makes it more robust against the well known *curse of dimensionality*, mainly for small data sets, since classification success does not depend on the dimensions of the input space.

We can summarize the construction of a SVM classifier as follows. Consider a set of training labeled samples represented by $\mathcal{D}$=$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, where $\mathbf{x}_i \in d\Re$ denotes a d-dimensional vector in a space and $y_i \in \{-1, +1\}$ is the label associated with it. The SVM training process, which produces a linear decision boundary (optimal hyperplane) that separates the two classes (−1 and +1), can be formulated by minimizing the training error:

$$\begin{aligned} &\min \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i, \\ &\text{subject to } y_i((w^T\mathbf{x}_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n \end{aligned} \tag{1}$$

while maximizing the margin separating the samples of the two classes. w is a weight vector orthogonal to the optimal hyperplane, b is the bias term, C is a tradeoff parameter between error and margin, ξ_i is a non-negative slack variable for $\mathbf{x}_i$. The optimization problem in Eq. (1) is usually solved by obtaining the Lagrange dual, which can be reformulated as:

$$\begin{aligned} &\max \quad \frac{1}{2}\sum_{i}^{n}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \mathbf{x}_i\mathbf{x}_j, \\ &\text{subject to } 0 \leq \alpha_i \leq C, \ \sum_{i}^{n}\alpha_i y_i = 0 \end{aligned} \tag{2}$$

where $(\alpha_i)_{i\in n}$ are Lagrangian multipliers computed during the optimization for each training sample. This process selects a fraction l of training samples with $\alpha_i > 0$. These samples are called support vectors and are used to define the decision boundary. In extreme case, the number of support vectors will be the same that in the training set. As a result, the w vector can be denoted as $\sum_i^n \alpha_i y_i x_i$. This SVM formulation only works for linearly separable classes. However, since real-world classification problems are hardly solved with a linear classifier, an extension is needed to nonlinear decision surfaces. To solve this problem, the dot products $(\mathbf{x}_i . \mathbf{x}_j)$ in the linear algorithm are replaced by a non-linear kernel function $K(.)$, where $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) . \Phi(\mathbf{x}_j)$ and Φ is a mapping function $\Phi : d\Re \mapsto H$. Such a replacement is the so-called kernel trick. In order to work, the kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ must satisfy the Mercer condition [4]. In particular, the kernel trick enables the linear algorithm to map the data from the original input space $d\Re$ to some different space H (possibly infinite dimensional) called the feature space. In the feature space, non-linear SVMs can be generated since linear operations in that space are equivalent to non-linear operations in the input space. The most common kernels used for this task and their parameters (γ, r, u and τ) are listed in Table 1. The decision

Table 1
Compilation of the most common kernels.

Kernel	Inner product kernel
Linear	$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
Polynomial	$K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^u, u > 0$
Radial Basis Function (RBF)	$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \parallel \mathbf{x}_i - \mathbf{x}_j \parallel^2), \gamma > 0$
Sigmoid	$K(\mathbf{x}_i, \mathbf{x}_j) = tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + \tau)$

function derived by the SVM classifier for a test sample $\mathbf{x}$ and training samples $\mathbf{x}_i$ can be computed as follows for a two-class problem:

$$sign(f(\mathbf{x})) \text{ with } f(\mathbf{x}) = \sum_{i}^{l} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \quad (3)$$

In the same way that this extension deals with non-linear problems, the primary SVM formulation requires additional modifications to solve multi-classes problems ($c > 2$). There are three approaches for handling this: the one-against-one, one-against-others, and all-together. The one-against-one strategy arranges pairs of classifiers to separate classes into an also called pairwise scheme, where the total number of classifiers is $c(c-1)/2$. Given a test sample, the classification result is obtained by comparing the pairs and assigning to it the class with the maximum number of votes. In contrast, the one-against-others strategy yields one classifier for each class c that separates that class from all the other classes. The final decision is made by the winner takes-all method, in which the classifier with the highest output function designates the class. Finally, the all-together strategy attempts to solve one single optimization problem which takes all classes into consideration. In this work, we use the one-against-one strategy, since it has been demonstrated to be faster to train and uses fewer support vectors than the one against all approach [9].

Overall, the SVM is a powerful classifier with strong theoretical foundations and good generalization performance. However, as well as it occurs in most machine learning algorithms, training it requires a fine tuning of its hyper-parameter set (i.e. kernel parameters and regularization parameter C). For instance, C is a penalty parameter of the error term, e.g. a high value punishes the errors too much and the SVMs can either over fit the training data or under fit them. Besides, kernel parameters that are not well tuned can also lead to under fitting or over fitting of the data. In our case of interest, if the RBF kernel parameter γ is improperly set the SVMs easily over or underfit the training data, while a bad setting for C can cause an explosion in the number of support vectors identified, thereby augmenting the complexity of the classifiers obtained. Thus, the tuning of SVM hyper-parameters controls the classifier's power of generalization. Therefore, the problem now is how to find their best values, which is a non trivial task (the so-called model selection problem). In the next section, we explain this problem and relate it to dynamic optimization problems.

3. SVM model selection and dynamic optimization problems

In order to generate high performing SVM classifiers capable of dealing with continuously updated training data an efficient model selection method is required. The model selection task can be divided into two main phases: the searching phase and the final training/test phase. The searching phase involves solving an optimization problem whose goal is to find optimal values for the SVM hyper-parameters considered in this paper (C and γ) with respect to some preference, or selection criterion. In our case this criterion is expressed as an objective function $\mathcal{F}$ evaluated over a training dataset $\mathcal{D}$, in terms of the cross-validation error ϵ. So, our model parameter selection problem takes the following form $min(\epsilon((C, \gamma), \mathcal{D}))$, or for simplification purposes here, $min(\epsilon(\mathbf{s}, \mathcal{D}))$. The final training/test phase in concerned with the production and evaluation on a test set of the final SVM model created based on the optimal hyper-parameter set found so far in the searching phase. On the other hand, the final training and test phase concerns the production and evaluation of the final SVM model $\mathcal{M}$ created based on the optimal hyper-parameter set found so far in the searching phase. In other words, the common process related to these two phases can be summarized in five steps: (1) Collect training data; (2) Start the search for solutions; (3) Find the hyper-parameters that perform best; (4) Train the final model with the best hyper-parameters; and lastly (5) Assess the performance of the final model using the test set. In Table 2 we summarize examples of SVM model selection methods found in the literature organized according to the type of kernel, search methods, and objective functions employed. We note that the RBF kernel has been investigated the most, perhaps due to the fact that the kernel matrix using sigmoid function may not be positive defined. Besides, even though the polynomial kernel may be an attractive alternative, but numerical difficulties tend to arise if a high degree is used, for example, a power of some minor value that 1 tends to 0 and of a major one that tends to infinity. Furthermore, the RBF kernel has often achieved a superior power of generalization with lower complexity than the polynomial kernel [18]. Because of this, the RBF kernel is considered in this study.

Most of effort associated with the approaches listed in Table 2 concentrated on solving the complex SVM model selection problem from one static training dataset available at time k. In this case, it should be convenient to use perfect, i.e. noise-free, data and in a fair amount in order to reach high performances. However, data from real-world applications are usually far from perfect, which

Table 2
Compilation of some related works on SVM model hyper-parameters selection in terms of the type of kernel used, the search method, and the objective function.

Ref.	Kernel[a]	Search method	Objective function
[19]	RBF	Grid-search (GS)	ν-Cross-validation
[6]	RBF	Gradient descent (GD)	Radius-margin, Span bounds, Leave-one-out
[8,9]	RBF	Grid-search (GS)	ν-Cross-validation error (CV)
[11]	RBF	Genetic algorithm (GA)	Radius-margin bound
[12]	RBF	Genetic algorithm (GA)	ν-Cross-validation error (CV)
[7]	RBF	Gradient descent (GD)	Hold out error, radius-margin, Generalized Approximate CV error (GACV)
[20]	RBF	Gradient descent (GD)	Leave-one-out (LOO), span bound
[21]	RBF,POL	Grid-search (GS)	ν-Cross-validation
[22]	POL	Gradient descent (GD)	Generalization error estimation bound
[13]	RBF	Multi-objective GA (MOGA)	Modified radius-margin bounds
[16]	RBF	Particle Swarm Optimization (PSO), GS	Hold out error, $\xi\alpha$-estimator
[10]	RBF	Uniform design (UD), Grid-search (GS)	ν-Cross-validation
[17]	RBF	Particle Swarm Optimization (PSO)	False Acceptance (FA)

[a] RBF: Radial Basis Function kernel whose hyper-parameter is γ. POL: polynomial kernel which hyper-parameters are the degree u and coefficient r. Kernel hyper-parameters and the regularization parameter C are optimized simultaneously.

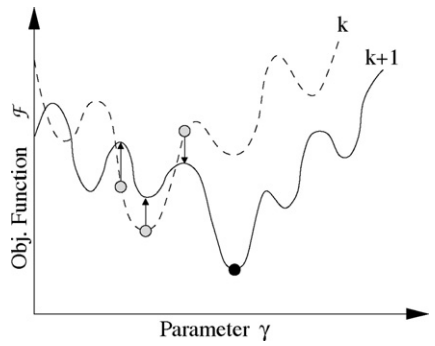


Fig. 1. Illustration of changes in the objective function. In a first moment (k), solutions are approximated for a parameter γ. Next, due to some unexpected change, e.g. new data, noisy, etc., the objective function changes and the solutions (gray circles) stay trapped in a local minimum, what requires some kind of reaction to adapt the solutions to the new function and optimal (dark point) in $k+1$.

gives the model selection process itself the potential for many types of uncertainty. In general, uncertainty is a multifaceted concept which usually involves vagueness, incompleteness, missing values, or inconsistency. Here, we assert that some uncertainties related to the machine learning area, such as missing features, random noise, or data insufficiency, generate uncertainties that can disturb the optimization process responsible for model selection. This is because uncertainties may produce some dynamism in the objective function, and so it is important to understand SVM model selection as a dynamic optimization problem.

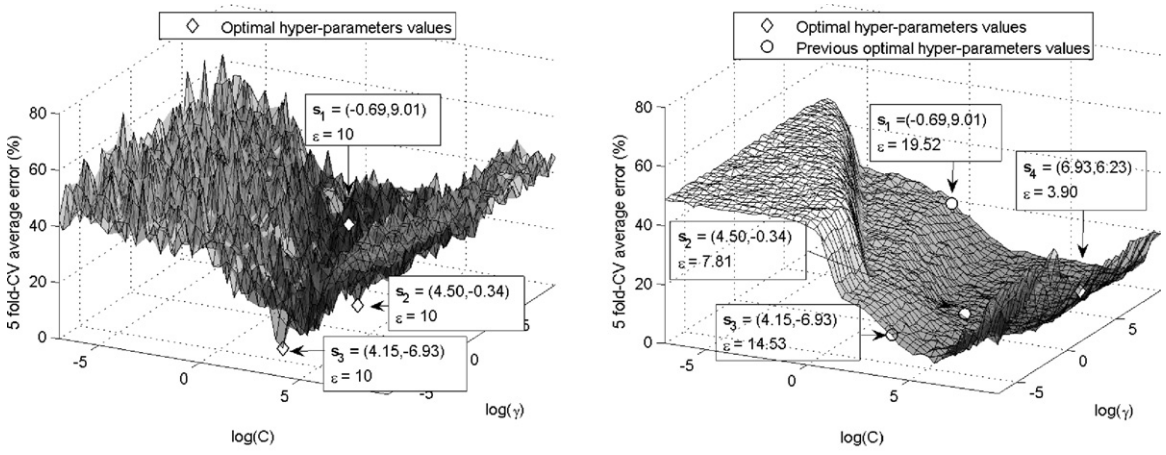
Dynamic optimization problems are complex in that the optimal solution can change overtime in different ways [23]. The changes can result from variations in the objective function, which implies in fitness dynamism. Fig. 1 depicts a conceptual example of fitness dynamism, and its consequences, and shows why dynamic optimization techniques are claimed. One can see that in a first moment (k), the optimization process approximates some solutions for a parameter γ. Then, due to some unexpected change related to the optimization task, e.g. new data, noise, etc., the objective function changes, and the solutions become outdated and trapped into a local minimum in the future (e.g. in $k+1$). This requires that the optimization algorithm be capable of re-adapting the solutions to new functions.

To demonstrate this fact, we depict a case study in Fig. 2 regarding the $min(\epsilon(\mathbf{s}, \mathcal{D}(k)))$ mentioned above. First, in Fig. 2(a) we can see an SVM hyper-parameter search space and optimal solutions obtained with a certain number of data samples from a classification problem. Then, the entire search space was recomputed with the same objective function (five-fold cross-validation average error), but this time from more data. The resulting search space is shown in Fig. 2(b). It can be seen that the search space and the optimal points may actually change depending on the amount of knowledge available about the problem. This applies to both objective function values, since the new objective values of previous optimal solutions $\mathbf{s}^*$ have worsened from $\epsilon = 10\%$ (e.g. $\mathbf{s}_1$ and $\mathbf{s}_3$) or improved (to $\mathbf{s}_2$, for example), once a new optimal solution emerged, that is, $\mathbf{s}_4$ = (6.93,6.23). The classification problem used for this case study is called P2 and is introduced with other datasets in Section 5.3. Through this example, it is easy to see that the search space and optimal points may change in terms of both fitness values and positions.

In order to show the effect that these hyperparameters changes produce in obtaining a final SVM model, we depict in Fig. 3, for this same example, the input spaces and the respective decision boundaries produced by SVM models trained with different hyperparameters values and number of samples. From these results, we can see that despite of $\mathbf{s}_2$ adequately separates the classes given a certain knowledge about the problem (Fig. 3(a)), it is not capable of producing the same satisfactory results (Fig. 3(b)) that a new best evaluated solution (i.e. $\mathbf{s}_4$) can achieve (Fig. 3(c)) if more samples are considered. Moreover, regarding the real-world situations addressed in this paper, the model selection process must also be designed to perform over time, i.e. for many datasets or incoming data. This is another reason why the SVM model selection problem can be seen as a dynamic optimization problem, in which solutions (i.e. hyper-parameters) must be checked and selected over time, since optimal hyper-parameter values can change dynamically depending on the incoming data at different times k. Thus, in addition to the approaches mentioned above which may only partially solve the problem and in order to attend to real-world applications needs, especially for updating and/or generating new models, this problem claims for more sophisticated methods capable of adapting new solutions and saving computational time, rather than for example, starting search processes from scratch every time.

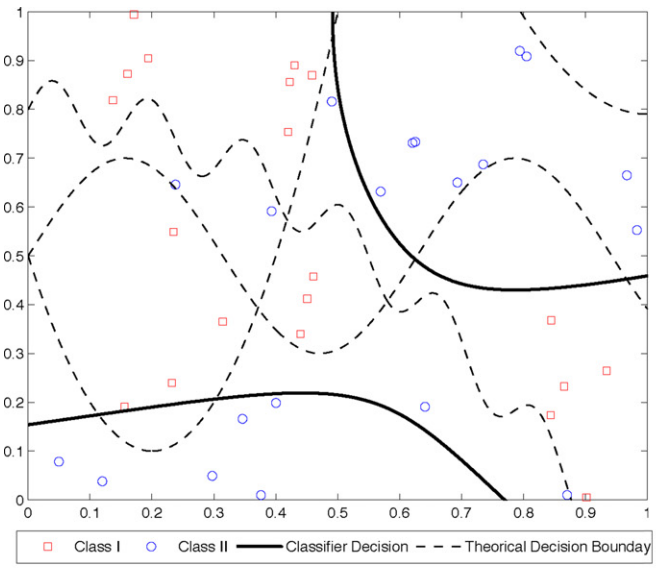
4. The proposed dynamic SVM model selection strategy

The goal of the proposed method is to point out dynamically optimum solutions for sequences of datasets $\mathcal{D}(k)$ by switching

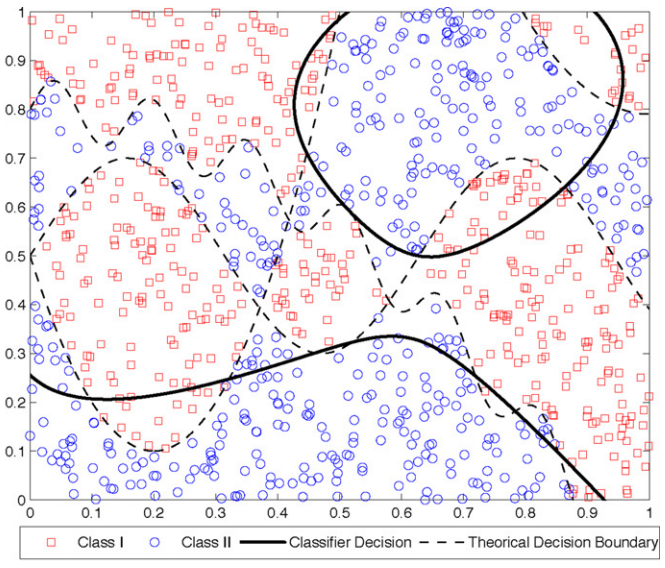


(a) 40 samples (b) 922 samples

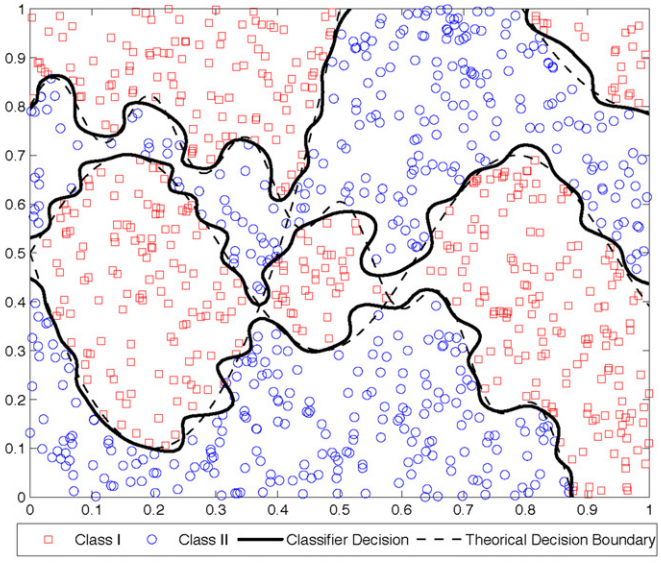
Fig. 2. Hyper-parameter search space for P2 problem with different number of samples. (a) 40 samples and (b) 922 samples.



(a) $\mathbf{s}_2$ and 40 samples, $\epsilon = 10\%$



(b) $\mathbf{s}_2$ and 922 samples, $\epsilon = 7.81\%$



(c) $\mathbf{s}_4$ and 922 samples, $\epsilon = 3.90\%$

Fig. 3. Input spaces and resulting decision boundaries produced by training SVM models with different hyperparameters values and number of samples for the P2 problem. (a) Decision boundaries obtained after training with the solution $\mathbf{s}_2$ and 40 samples, $\epsilon = 10\%$. (b) Decision boundaries obtained for the same optimum solution $\mathbf{s}_2$ for 40 samples, but now training over 922 samples, $\epsilon = 7.81\%$. (c) Final result achieved for the best solution $\mathbf{s}_4$ regarding 922 samples, $\epsilon = 3.90\%$.

among three levels: (1) use the best solution $\mathbf{s}^*(k-1)$ found so far, (2) search for a new solution over an adapted grid composed of a set of solutions $\mathcal{S}(k-1)$, or (3) start a dynamic optimization process. In this paper, each solution $\mathbf{s}$ will represent a PSO particle, which codifies an SVM hyper-parameter set, e.g. (C, γ). The switching among the levels is governed by change detection mechanisms which monitor novelties in the objective function $\mathcal{F}$. Such changes correspond to degradation of performance or no improvement at all (stability) with respect to new data, which will indicate whether or not the system must act. Fig. 4 depicts an overview of the general concept proposed. First, a population of solutions (swarm) $\mathcal{S}(0)$ is initialized by the *optimization algorithm* to search for solutions for the dataset $\mathcal{D}(1)$, after which the optimization process finishes and, a set of optimized solutions $\mathcal{S}(1)$ is stored for future use. Based on fitness re-estimation or according to some other criterion related to the problem, the current status of the best solution (dark circle) will be examined on new data. Following the example, we suppose that the fitness re-estimated from the previous best solution $\mathbf{s}^*(1)$ for the dataset $\mathcal{D}(2)$ is still satisfactory, and apply the same solution to train a new classifier. However, more data can be available and the goodness of the best solution $\mathbf{s}^*(1)$ may no longer be guaranteed, e.g. between datasets $\mathcal{D}(3)$ and $\mathcal{D}(4)$. To solve this, we suggest performing a fine search over the set of optimized solutions $\mathcal{S}(1)$. We call this process an *adapted grid-search*, since it applies solutions already optimized, which are probably located over a good deal of the search space, and are not guessed values as occurs in the traditional grid-search. The advantage is that, in the most of the time, the adapted grid-search can indeed gain in performance if compared with traditional grid methods and also save computational time if compared with full optimization processes. However, when it is not possible to identify a satisfactory solution even after an adapted grid search, the method starts a dynamic optimization process, as

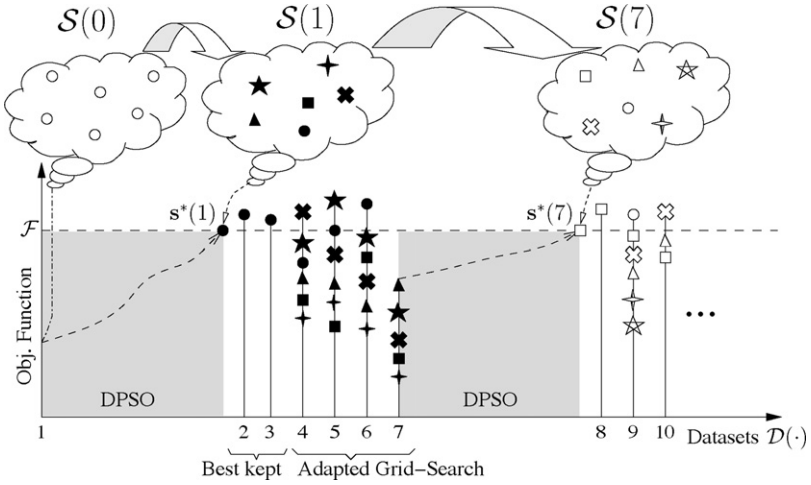


Fig. 4. Overview of the proposed model selection strategy (conceptual idea). Optimum solutions for a current dataset $\mathcal{D}(k)$ are pointed out by switching among three search strategies: (1) use the best solution $\mathbf{s}^*(k-1)$ found so far, (2) search for a new solution over an adapted grid composed of a set of solutions $\mathcal{S}(k-1)$, or (3) start a dynamic optimization process. The symbols represent different solutions from a swarm. The best solution selected for a dataset lies above the dashed line. The white circles in $\mathcal{S}(0)$ denote randomly initialized solutions. Dark and white symbols indicate solutions from different swarms.

denoted for the dataset $\mathcal{D}(7)$. As a result, a new population of solutions, surely better adapted to the problem, will be available for the future. We introduce the framework of the proposed method below.

4.1. Framework

As we mentioned previously, the ideal method of creation of an SVM classifier is composed of two phases: model selection and training/test phases. The first is responsible for searching for the best SVM hyper-parameters and the second phase uses the best hyper-parameters found to train and test a final SVM model $\mathcal{M}$. In this work, based on the conceptual idea depicted in Fig. 4 and also by concepts of dynamic optimization problems introduced in Section 3, we propose a framework for the dynamic selection of SVM models composed of three main modules, as shown in Fig. 5 and listed in Algorithm 1: change detection, adapted grid-search, and dynamic Particle Swarm Optimization (DPSO). We detail these below. The *upgrade_stm* and *recall_stm* functions are respectively responsible for storing and retrieving optimized solutions from the system's Short Term Memory (STM).

Algorithm 1. Dynamic SVM Model Selection

Input: A training set of data $\mathcal{D}(k)$.
Output: Optimized SVM classifier.
recall_stm($\mathbf{s}^*(k-1)$,$\mathcal{S}(k-1)$)
if there is a $\mathcal{S}(k-1)$ **then**
  Check the preceding best solution $\mathbf{s}^*(k-1)$ regarding the dataset $\mathcal{D}(k)$
  if Change_Detection($\mathbf{s}^*(k-1)$,$\mathcal{D}(k)$) **then**
    Activate the adapted grid-search module and get solution $\mathbf{s}'(k)$
    if Change_Detection($\mathbf{s}'(k)$,$\mathcal{D}(k)$) **then**
      Activate the DPSO module
    end if
  end if
else
  Activate the DPSO module
end if
upgrade_stm($\mathbf{s}^*(\cdot)$,$\mathcal{S}(\cdot)$)
Train the final SVM classifier from $\mathcal{D}(k)$ by using the optimum solution found so far by the modules.

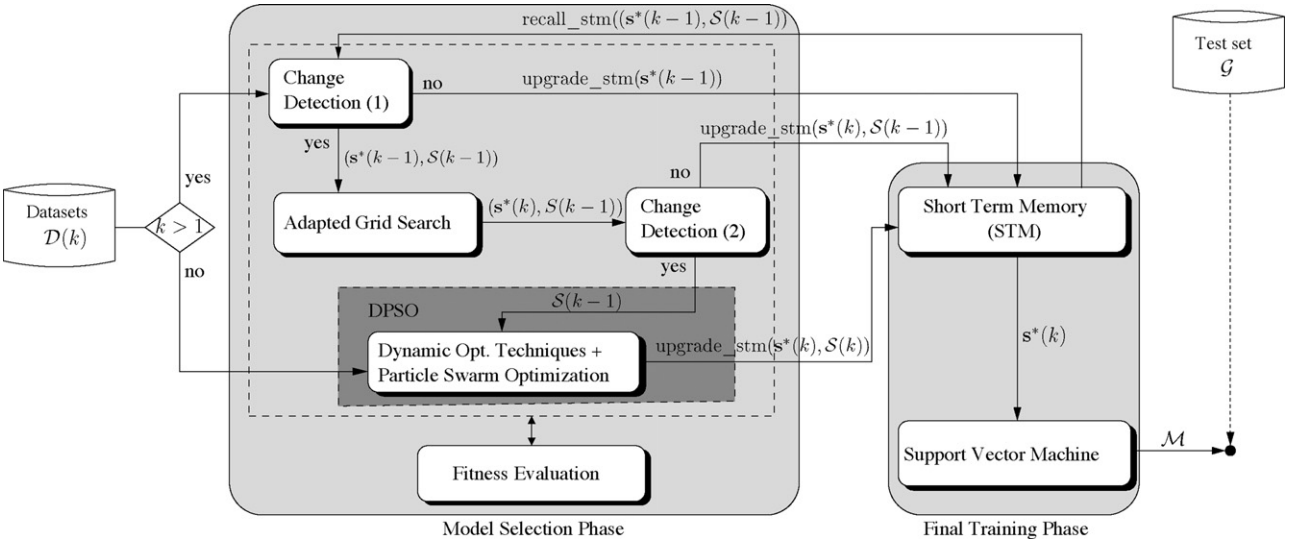


Fig. 5. General framework of the proposed method for the dynamic SVM model selection.

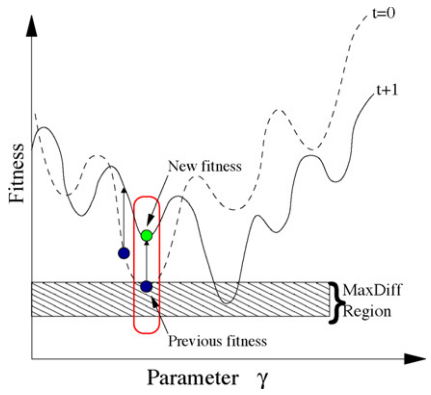


Fig. 6. Illustration of the change detection mechanism. In this case, as the new fitness is situated outside the expected region, a new optimization is carried out in order to find a new better solution.

4.1.1. Change detection module

The change detection module controls the intensity of the search process by pointing out how the solutions are found thereby the levels of the framework. In particular, it is responsible for simultaneously monitoring the quality of the model selection process and avoiding "unnecessary" searching processes. We implement it by monitoring differences in the objective function values, in this case error estimations ϵ obtained for a best solution $\mathbf{s}^*$ on the datasets $\mathcal{D}(k-1)$ and $\mathcal{D}(k)$, for example. We denote this fact as $\epsilon(\mathbf{s}^*, \mathcal{D}(k-1))$ and $\epsilon(\mathbf{s}^*, \mathcal{D}(k))$, respectively. If the solution found is not to be satisfactory for the process, then a further searching level is activated. The adequacy of a solution can be measured in several ways. In this work, as we are interested in finding performing solutions, we consider that further searches are needed. If the objective function value computed does not lie in a "stable" region.

The stable region is computed through the maximum expected difference δ_{max} between the objective function values at the 90% confidence level using a normal approximation to the binomial distribution (see Eqs. (4) and (5)) [24]. In this setting, if there is a degradation of performance ($\epsilon(\mathbf{s}^*, \mathcal{D}(k-1)) < \epsilon(\mathbf{s}^*, \mathcal{D}(k))$) or significant variation in the objective function (i.e. $|\epsilon(\mathbf{s}^*, \mathcal{D}(k-1)) - \epsilon(\mathbf{s}^*, \mathcal{D}(k))| \geq \delta_{max}$), then other levels are activated for additional searches. Fig. 6 depicts an illustration of the δ_{max} stable region idea. In order to make this criterion more robust when small datasets are used, we combine it with a rule related to the compression capability of the classifier. The compression capability is calculated as the proportion of support vectors over the number of training samples. If the δ_{max} rule and a minimal compression required are attained, the situation is characterized as stable and no further searches are computed. Otherwise, the model selection process continues by activating the other modules.

$$\delta_{max} = z_{0.9} \times \sqrt{\sigma} = 1.282 \times \sqrt{\sigma} \tag{4}$$

where σ is computed by Eq. (5), where $W(\cdot)$ is the dataset size:

$$\sigma = \frac{\epsilon(\mathbf{s}^*, \mathcal{D}(k-1)) \times (1 - \epsilon(\mathbf{s}^*, \mathcal{D}(k-1)))}{W(\mathcal{D}(k-1))} + \frac{\epsilon(\mathbf{s}^*, \mathcal{D}(k)) \times (1 - \epsilon(\mathbf{s}^*, \mathcal{D}(k)))}{W(\mathcal{D}(k))} \tag{5}$$

So, the change detection module may sometimes denote a trade-off controller between computational time spent and the quality of solutions. For instance, if we ignore this module, then dynamic re-optimization processes will be always conducted, which can produce indeed good results but to be unnecessarily time consuming for stable cases.

4.1.2. Adapted grid-search

The adapted grid search module provides optimum solutions by re-evaluating the knowledge acquired from previous optimizations performed by the DPSO module. This knowledge is represented by a set $\mathcal{S}(k-1)$ of optimized solutions which are stored in the *short term memory*. Usually, this method finds better solutions than the traditional grid-search method.

Unlike the traditional grid-method, which depends on the discretization of values and requires the evaluation of several combinations (see Fig. 7 for two hyper-parameters (C and γ)), the adapted grid-search module reduces the number of trials by focusing the search in an optimal region. As a result, this module can save a considerable computational time.

Basically, this module uses the best positions of preceding optimized solutions as a grid of new possible candidate solutions to be evaluated over the current data $\mathcal{D}(k)$. At the end of the process, the best candidate is selected. Although we employ this implementation, we can suggest other modifications, such as moving the particles by using a complete iteration of PSO. Such a process seems interesting, but costs more in terms of processing time than simply re-evaluating the best particles' positions, which in most of cases may be enough.

Nevertheless, it is important to note that the module's results are related to the quality of the previous optimizations. Therefore, it is efficient when the current population of solutions is positioned on optimal regions. Otherwise, it may produce sub-optimum solutions that will be not satisfactory for final learning purposes. In light of this, we apply the change detection a second time in order to ensure the quality of the solution obtained at the end of this process, as indicated in the framework in Fig. 5. If the current solution is still not considered satisfactory, the dynamic optimization module is activated.

4.1.3. Dynamic Particle Swarm Optimization-DPSO

The DPSO module is responsible for finding new solutions by means of re-optimization processes. We implement it based on the Particle Swarm Optimization (PSO) algorithm combined with dynamic optimization techniques.

The Particle Swarm Optimization (PSO) method was firstly introduced by Kennedy and Eberhart in 1995 [25]. Briefly, it is a population-based optimization technique inspired by the social behavior of flocks of birds or schools of fishes. It is applied in this work because it has many advantages that make it very interesting when compared with other population-based optimization techniques, e.g. genetic algorithms (GA). For instance, PSO belongs to the class of evolutionary algorithms that does not use the "survival of the fittest" concept. It does not utilize a direct selection function, and so, particles with lower fitness can survive during the optimization and potentially visit any point in the search space. Furthermore, the population size usually employed in PSO gives it another advantage over GA, since the lower population size in PSO favors this algorithm regarding the computational time cost factor [26]. Nonetheless, two main additional characteristics give us further motivation for using it. First, PSO has a continuous codification, which makes it ideal for the search of optimal SVM hyper-parameters. Second, the potential for adaptive control and flexibility (e.g. self-organization and division of labor) provided by the swarm intelligence makes PSO very interesting to be explored for solving dynamic optimization problems.

In this section, we simplify the index notation (e.g. for time or datasets) and use only those needed to understand the PSO technique well. In particular, the standard PSO involves a set

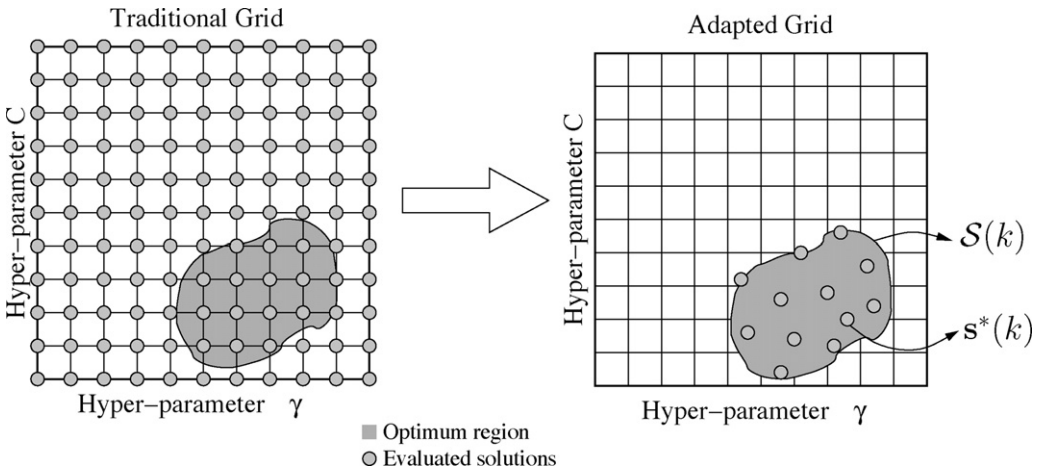


Fig. 7. The traditional grid must try a higher number of combinations than the adapted grid, which profits from the already optimized solutions $\mathcal{S}(k)$ provided by DPSO. $\mathbf{s}^*(k)$ denotes the best solution.

$\mathcal{S} = \{\mathbf{s}_i, \dot{\mathbf{s}}_i, \mathbf{s}'_i\}_{i=1}^{P}$,[2] of particles that fly in the search space looking for an optimal point in a given d-dimensional solution space. The $\mathbf{s}_i = (s_i^1, s_i^2, \ldots, s_i^d)$ which is a vector that contains the set of values of the current hypothesis. It represents the current location of the particle in the solution space, where the number of dimensions is problem dependent. The vector $\dot{s}_i = (\dot{s}_i^1, \dot{s}_i^2, \ldots, \dot{s}_i^d)$ which stores the velocities for each dimension of the vector $\mathbf{s}_i$. The velocities are responsible for changing the direction of the particle. The vector $\mathbf{s}'_i = (s'^1_i, s'^2_i, \ldots, s'^d_i)$ is a copy of the vector $\mathbf{s}_i$ which produced the particle's individual best fitness. Together, $\mathbf{s}'_i$ and $\mathbf{s}_i$ represent the particles' memories. Regarding the model selection problem, the vector positions $\mathbf{s}_i$ encode the SVM hyperparameter set to be optimized and $\mathbf{s}^*$ denotes the best solution found.

PSO starts the search process by initializing the particles' positions randomly over the search space. Then, it searches for optimal solutions iteratively by updating them to fly through a multidimensional search space by following the current optimum particles. The direction of the particle's movement is governed by the velocity vector $\dot{s}_i$, which is denoted by the sum of the information from the best particle's informant found in its neighborhood λ (e.g. $s'_{net(i,\lambda)}(q)$) and the particle's own experience $\mathbf{s}'_i$. For a new iteration $q+1$ and dimension d, the update is computed as follows:

$$\dot{s}_i^d(q+1) = \chi(\dot{s}_i^d(q) + \phi r_1(s'^d_i(q) - s_i^d(q)) + \phi r_2(s'^d_{net(i,\lambda)}(q) - s_i^d(q))) \quad (6)$$

where χ is the constriction coefficient introduced by Clerc and Kennedy [27], and r_1 and r_2 are random values. Constriction coefficient values of $\chi = 0.7298$ and $\phi = 2.05$ are recommended [26]. Eventually the trajectory of a particle is updated by the sum of its updated velocity vector $\dot{s}_i(q+1)$ to its current position vector $\mathbf{s}_i(q)$ to obtain a new location, as depicted in Eq. (7). Fig. 8 depicts an illustration of particle's trajectory during position updating. Therefore, each velocity dimension $\dot{s}_i^d$ is updated in order to guide the particles' positions s_i^d to search across the most promising areas of the search space. In Algorithm 2 we summarize the standard PSO method.

$$s_i^d(q+1) = s_i^d(q) + \dot{s}_i^d(q+1) \quad (7)$$

[2] We use this functional notation for sake of generality. The equivalent to traditional PSO would be: $\mathcal{S} = \{\mathbf{x}_i, \mathbf{v}_i, \mathbf{p}_i\}_{i=1}^{P}$.

Algorithm 2. Standard PSO Algorithm

Input: PSO parameters.
Output: Optimized solutions.
Randomly initialize the particles
$q \leftarrow 0$;
repeat
  for all particles i such that $1 \le i \le P$ **do**
    Compute fitness value for the current position $\mathbf{s}_i(q)$
    Update $\mathbf{s}'_i(q)$ if position $\mathbf{s}_i(q)$ is better ($\mathbf{s}'_i(q) \leftarrow \mathbf{s}_i(q)$)
  end for
  Select the best fitness $\mathbf{s}'_i(q)$
  for all particles i such that $1 \le i \le P$ **do**
    Update velocity $\dot{s}_i(q)$ (Eq. (6)) and current position $\mathbf{s}_i(q)$ (Eq. (7))
  end for
  $q = q + 1$
until maximum iterations or another stop criteria be attained

In the canonical PSO formulation, an entire swarm is considered as a single neighborhood where particles share the best information discovered by any member of the swarm, the so-called *gbest* topology. The main disadvantage is that it forces the particles towards a single best solution, which causes the swarm to lose the ability to explore the search space in parallel more locally. Moreover, it has a premature convergence tendency [26]. Because of this, we implement this module based on PSO with a more sophisticated topology called local best (*lbest*) [26]. This topology creates a neighborhood for each individual containing itself and its λ nearest neighbors in the swarm. The neighborhoods can overlap and every particle can be in multiple neighborhoods. As a result, it allows interactions among the neighborhoods and eventually more series of events may be discovered. With this characteristic, this module is capable of exploring multiple regions in parallel and therefore fits better for

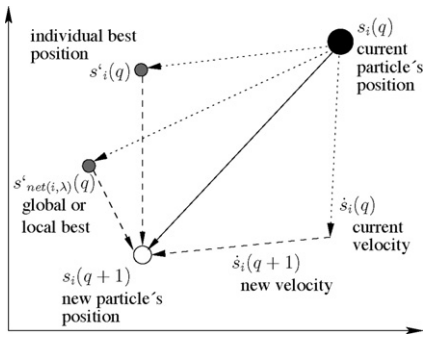


Fig. 8. Example of particle's trajectory during position updating.

functions with possible multiple local optima. Such a parallelism allows distant neighborhoods to be explored more independently, which is important for multi-modal problems. Moreover, the particles are placed in potentially more promising regions, which can allow faster recovery from variations between searching processes and also allow them to be used by the adapted grid search module.

Nevertheless, even though PSO is a powerful optimization method, if the optimization problem suffers some change in the objective function, for example between blocks of data, the particles can get stuck in local minima (see Fig. 2). To avoid this, an alternative should be to start a full PSO optimization process from scratch each time that the module is activated. However, it would be very time consuming and even at times unnecessary if the changes occur around the preceding optimum region. Taking this into account, we enable the module to restart optimization processes from preceding results in order to save computational time. To implement this mechanism, we combine two dynamic optimization techniques: re-randomization and re-evaluation of solutions, and apply them into our PSO based module. In fact, both techniques were already applied in the PSO literature [28,29] to solve dynamic optimization problems, but separately and using the *gbest* topology. In particular, these PSO variants are commonly called DPSO (Dynamic PSO), so for the sake of simplicity, we name this module as DPSO to refer such a combination of approaches. Nevertheless, it is important to distinguish that existing dynamic PSO algorithms apply such techniques and change detection mechanisms in each iteration, since they suppose that objective function changes can happen during the optimization. In here, as the optimization over a dataset $\mathcal{D}(k)$ at a given instant k is indeed static, we apply these dynamic techniques to prepare the optimization module for transitions from preceding optimizations knowledge to launch new ones. As a result, we take advantage of these techniques to provide diversity in the solutions and clues on optimal starting points before the optimization. Thus, unlike actual dynamic PSO versions, no extra computational effort is added at each iteration. In light of this, in Fig. 5 and in the rest of this paper, our DPSO module represents the application of these dynamic techniques to cooperate with the optimization algorithm, but not in its interior in each iteration.

The focus now shifts to the whole implementation, which involves two main steps related to the way that the optimization process restarts. The main steps are listed in Algorithm 3. First of all, once the DPSO module is activated, which uses information from the system's memory (STM) as well, every fitness is updated from the re-evaluation of the current position $\mathbf{s}_i$ and best position $\mathbf{s}'_i$ of each particle $\mathbf{s}_i$ in the swarm $\mathcal{S}(k)$ (steps: 3 to 6). This is done to prevent the particle's memory from becoming obsolete [28]. In fact, the fitness of the best positions $\vec{p}_i$ can be profited from the preceding level (adapted grid-search), what dispenses a second evaluation. Thereafter, a re-optimization process is launched by keeping ρ% of the best particles positions from the swarm $\mathcal{S}(k-1)$, which was computed in the previous optimization, and by randomly creating new particles over the search space [29]. Some of these particles located near to the previous optimum region. In this manner, we guarantee that fine searches are realized based on previous information, which can adapt more quickly to new data than full optimization processes (steps 7 and 8). At the same time, we add more diversity to the algorithm for searching new solutions, which enable us to avoid situations in which the whole swarm has already converged to a specific area. Finally, steps 9 to 23 correspond to the main steps of the PSO implementation, but are slightly modified by adding a mechanism that updates the connections among the particles, if no improvement is observed between iterations (steps 19 to 21). These latter steps were suggested as an alternative by Clerc [30] to improve the adaptability, and hence the performance, of the swarm.

Algorithm 3. Our implementation of Dynamic PSO

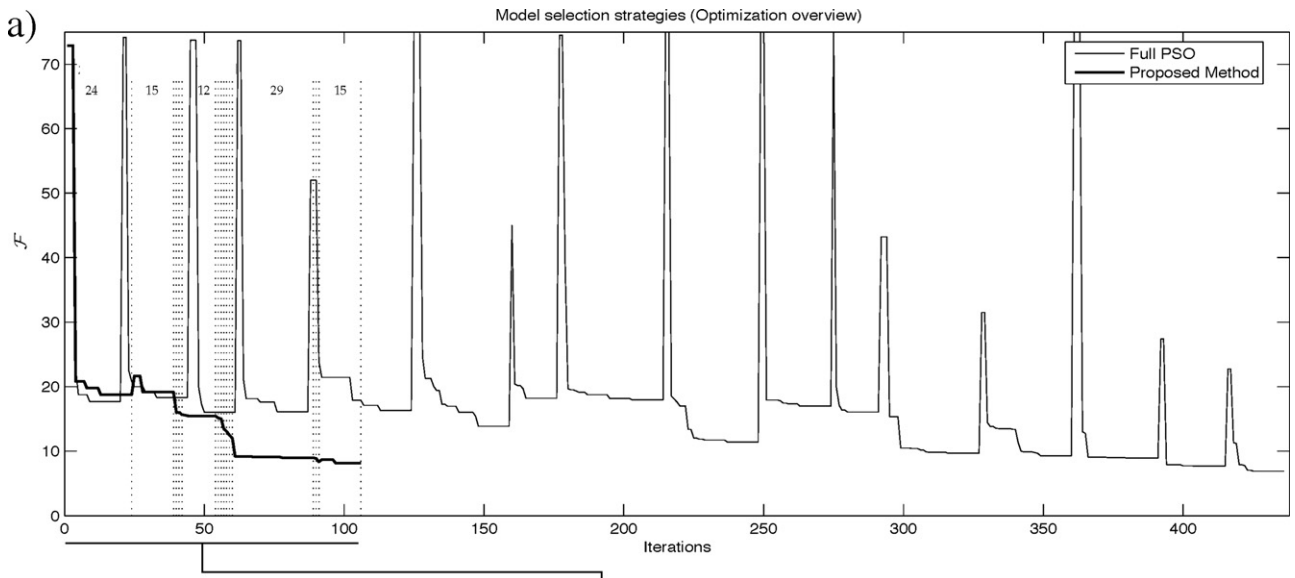
1: **Input:** PSO parameters and previous swarm $\mathcal{S}(k-1)$.
2: **Output:** Optimized solutions.
3: **for all** particles i from $\mathcal{S}(k-1)$ such that $1 \le i \le P$ **do**
4: Compute fitness values for $\mathbf{s}_i$ using $\mathcal{D}(k)$
5: Update $\mathbf{s}'_i$ if $\mathbf{s}_i$ is better ($\mathbf{s}'_i \leftarrow \mathbf{s}_i$)
6: **end for**
7: Initialize dynamically the new swarm $\mathcal{S}(k)$ by keeping ρ% of the best information (positions $\mathbf{s}'_i$) from the preceding swarm $\mathcal{S}(k-1)$ and by creating new particles.
8: Initialize the links among the particles based on a nearest neighborhood rule according to the topology chosen.
9: $q \leftarrow 0$;
10: **repeat**
11: **for all** particles i such that $1 \le i \le P$ **do**
12: Compute fitness value for the current position $\mathbf{s}_i(q)$
13: Update $\mathbf{s}'_i(q)$ if position $\mathbf{s}_i(q)$ is better ($\mathbf{s}'_i(q) \leftarrow \mathbf{s}_i(q)$)
14: **end for**
15: Select the best fitness of this iteration q, i.e. $\mathbf{s}'_i(q)$
16: **for all** particles i such that $1 \le i \le P$ **do**
17: Update velocity $\dot{s}_i(q)$ (Eq. (6)) and current position $\mathbf{s}_i(q)$ (Eq. (7))
18: **end for**
19: **if** $\mathcal{F}(\mathbf{s}^*(q)) = \mathcal{F}(\mathbf{s}^*(q-1))$ { No improvement. Change particle communication structure } **then**
20: Randomly change the particles' links based on the topology chosen.
21: **end if**
22: $q = q + 1$
23: **until** maximum iterations or other stop criteria be attained

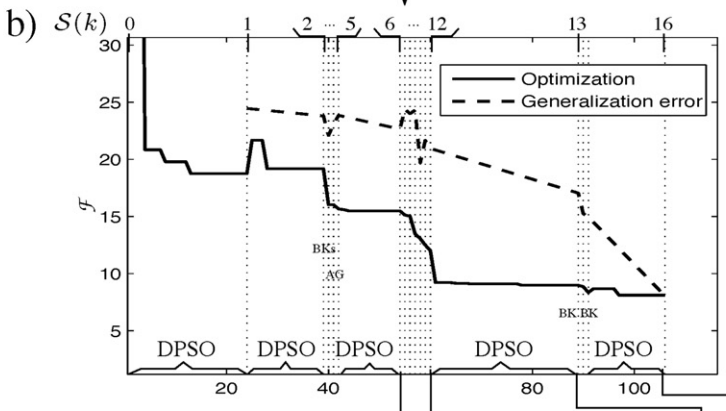
So, through the use of these modules, the proposed method allows the searching process to evolve and adapt itself dynamically. Even though this framework has unique features, there is still room for authors to investigate new strategies for the adapted grid search module, detection mechanisms, and even strategies to re-optimize solutions.

In order to clarify the whole concept, we illustrate the proposed method in a case study in Fig. 9. This case study represents an empirical reference to the general concept illustrated in Fig. 4. In particular, it depicts overviews of searching processes carried out by the proposed method and full optimization processes over cumulative sequences of data increased logarithmically from the Satimage database. Based on these results, it is shown in Fig. 9(a) that the proposed method can achieve similar results to those obtained by full optimization processes with PSO (Full PSO), but more quickly and in fewer iterations if the whole sequence is considered.

Exploring this case study further, we compile a list of activities performed by the proposed method during the searching processes and their effects in terms of the generalization error on a test set, as shown in Fig. 9(b). It is easy to see which module of the framework was responsible for selecting the final solution. In addition, we list the results of searching processes between the datasets $\mathcal{D}(6, 13)$ in a table in order to provide more details. Basically, the results in the table include the use of the optimized swarm $\mathcal{S}(6)$, resulting from a DPSO execution, as a pool of hypotheses for additional datasets, where a particle $\mathbf{s}_i$ is selected as the best one, according to some criteria and via: keeping the same previous best (BK), adapted grid (AG), or DPSO processes.

Some of the main results are depicted in the table in Fig. 9(c), where we have selected the ten most performing particles and presented their best positions in a logarithmic scale. Then, for each set, we indicate the solution pointed out by the method by highlighting its fitness in gray. When a previous best solution remains the same for the next dataset, no evaluation is performed for the other particles. Assuming that the solutions are well-placed in the search space, we have started by reporting the results for the dataset $\mathcal{D}(6)$, where the best solution $\mathbf{s}_9$ in swarm $\mathcal{S}(6)$ was found by DPSO. Next, the solution $\mathbf{s}_9$ found over the dataset $\mathcal{D}(6)$ has been kept for dataset $\mathcal{D}(7)$. We note that the current best solution experienced





c)

$\mathbf{s}_i$	$\mathcal{S}(6)$ $(C\gamma)$	$\mathcal{F}$	Datasets $\mathcal{D}(k)$ 7	8	9	10	11	12	13	$\mathcal{S}(13)$ $(C\gamma)$	$\mathcal{F}$	$\mathcal{S}(16)$ $(C\gamma)$	$\mathcal{F}$
$\mathbf{s}_1$	(3.70,–4.50)	19.84	–	18.20	15.76	–	13.76	13.13	11.07	(4.15,–4.51)	8.97	(4.19,–2.33)	10.35
$\mathbf{s}_2$	(4.07,–4.31)	17.46	–	15.90	13.79	–	12.55	13.01	9.62	(4.07,–4.50)	9.32	(3.90,–1.45)	10.06
$\mathbf{s}_3$	(4.05,–4.50)	17.46	–	15.71	13.79	–	12.55	12.46	10.31	(4.06,–4.51)	9.15	(4.07,–4.51)	9.32
$\mathbf{s}_4$	(3.74,–4.49)	20.63	–	18.39	15.45	–	13.20	12.98	10.09	(4.20,–4.51)	9.03	(4.07,0.11)	8.21
$\mathbf{s}_5$	(3.93,–4.51)	18.65	–	16.86	15.00	–	12.73	12.54	10.49	(4.19,–4.51)	10.02	(4.04,–4.51)	9.62
$\mathbf{s}_6$	(4.04,–4.39)	19.44	–	16.09	14.09	–	12.55	12.54	10.37	(2.89,–4.50)	9.44	(4.19,–4.51)	9.09
$\mathbf{s}_7$	(4.19,–4.51)	15.38	–	15.02	13.64	–	12.64	12.02	10.02	(3.61,–4.51)	10.78	(4.03,0.06)	8.32
$\mathbf{s}_8$	(3.99,–4.51)	18.25	–	16.28	14.39	–	12.83	12.32	10.49	(3.89,–4.51)	9.79	(4.21,–4.51)	9.03
$\mathbf{s}_9$	(4.21,–4.51)	15.35	15.12	15.52	13.48	13.10	12.73	12.17	10.08	(4.04,–4.51)	9.62	(3.84,0.18)	8.12
$\mathbf{s}_{10}$	(3.90,–4.48)	19.84	–	16.86	15.15	–	12.92	12.54	10.49	(3.97,–4.48)	10.31	(4.18,–2.0)	10.23
	DPSO		BK	AG	AG	BK	AG	AG		DPSO		DPSO	

Fig. 9. Case study: operation of the proposed method, dynamic model selection (DMS). In (a), we show an overview of searching processes for SVM models based on the proposed method and on full optimization processes over sequences of incoming data. We can see that DMS can approximate performing solutions by requiring fewer iterations than full optimization processes. The dashed vertical lines indicate when more data were injected and how many iterations were needed to accomplish the searching tasks. Next, in (b) and (c), we show a zoom on the proposed method' activities and generalization errors. These figures empirically depict an analogy to the general concept illustrated in Fig. 4.

a decrease in performance between datasets $\mathcal{D}(7, 8)$ (in next column), which is denoted as a negative behavior. As a consequence, the adapted grid-search module is activated to try to find another satisfactory solution. Following evaluation, the adapted grid module elects a new solution $\mathbf{s}_7$ and no further searches are carried out, since the best current result has improved and there is no indication of any big changes that would justify additional optimizations. Next, between datasets $\mathcal{D}(8, 9)$, the change detection rule is re-activated, and again a fine search is carried out over the other solutions to check whether or not there is a better solution. The new solution returns to $\mathbf{s}_9$ and another application of the rule over the two best results indicates that the DPSO module does not need to be activated. Thereafter, between dataset $\mathcal{D}(9, 10)$, the current best particle $\mathbf{s}_9$ was preserved since no relevant variation has occurred. On the other hand, the same behavior between datasets $\mathcal{D}(8, 9)$ occurs among the datasets $\mathcal{D}(10, 12)$, resulting in $\mathbf{s}_2$ and $\mathbf{s}_7$, respectively. Afterwards, the searching process continues by re-activating DPSO for the dataset $\mathcal{D}(13)$, which results in a new

swarm $\mathcal{S}(13)$ with a new best solution $\mathbf{s}_9$. Therefore, dynamic optimizations are employed whenever the method judges it necessary to update the swarm. Mainly due to performance degradation, or for instance, when the adapted grid is activated and the results are neither improved nor do they characterize changes in the search space.

5. Experimental protocol

A series of experiments were carried out to test the effectiveness of the proposed method. In particular, we have compared our method with other model selection strategies under a gradual learning scenario. In the latter, an SVM classifier must be built gradually from scratch whenever more data become available. We have used datasets generated from synthetic and real-world problems. For each dataset, the following experimental setup was conceived: First of all, the original training sets were divided into sets of data. The total number of samples for each dataset was progressively increased according to a logarithmic rule [31], from about 16 examples per class to the total number of samples available in the dataset. For datasets in which the original distribution of samples was unbalanced among the classes, we have maintained the original class-priors for each dataset.

Then we have applied each SVM model selection strategy over the datasets. Once the model for each dataset had been selected, the performance of the classifiers was assessed in terms of its generalization error on the test set after each simulation. The generalization error was estimated as the ratio of misclassified test set samples over the total number of test samples. This made it possible to observe the effect of the training dataset size for each model selection approach and the final test performance attained. As some strategies tested use stochastic algorithms, the results represent averages drawn over 10 replications. The kernel chosen for the SVM classifier was the RBF (Radial Basis Function), and so, the model selection methods were carried out to find optimal values for the hyper-parameter set (C, γ). Additional specifications on the approaches tested and information on the datasets are provided in next section.

5.1. SVM model selection strategies tested

We have compared the following SVM model selection strategies:

- *Traditional Grid-Search (GS)*: This method selects the best solution by evaluating several combinations of possible values. The best combination is kept to train the final SVM classifier. In this study, we consider a grid of 70 (7×10) positions, where the possible combinations lie within these values: C = {0.01, 0.1, 100, 150, 170, 250, 600}, and γ = {0.08, 0.15, 15, 20, 50, 100, 300, 500, 1000, 1500}.
- *1st Grid-Search (1st-GS)*: This strategy applies a traditional grid-search only over the first dataset and retains the same solution found for the subsequent subsets.
- *Full Particle Swarm Optimization (FPSO)*: The optimal hyper-parameter values are selected by the standard PSO algorithm for each new set of data.
- *Chained PSO (CPSO)*: PSO is applied by this strategy to search for optimal solutions. However, the solutions here are optimized among sequences of datasets in a chained way, like a serial process. This means that the optimization process is performed continuously over the datasets, and not by fully re-initializing the swarm between sets.
- *Dynamic model selection (DMS)*: This strategy is the proposed method introduced in Section 4.

5.2. Experiments parameters setting

The following parameters setting were used in the experiments.

- *Optimization Algorithms Parameters*: The maximum number of iterations and the swarm size were set to 100 and 20, respectively. The dimensions of each particle are denoted by hyper-parameter values for C and γ, where the maximum and minimum values of such dimensions were set to $[2^{-6}, 2^{14}]$, $[2^{-15}, 2^{10}]$, respectively.

  The topology used in PSO and DPSO was *lbest* with $\lambda = 3$. This topology was selected because unlike the *gbest* topology, which has a tendency towards premature convergence because all the particles are influenced by the same global source, the *lbest* topology is more sophisticated for exploring multiple regions in parallel [26]. Furthermore, the parallelism of the *lbest* topology allows distant neighborhoods to be explored more independently. Basically, this topology creates a neighborhood for each individual comprising itself and its λ nearest neighbors in the swarm. A neighborhood may consist of some small group of particles, where the neighborhoods overlap and every particle can be in multiple neighborhoods.

  Two stop criteria were implemented for the optimization processes. The first was implemented based on the maximum iteration permitted. As a result, the optimization might finish whenever the number of iterations reaches the maximum value (100). However, the second criterion was built based on the best fitness value. Generally speaking, if the best fitness value did not improve over 10 consecutive iterations, then the optimization process was stopped. In fact, this last stop criterion was the most active, since the simulations never attained to the maximum number of iterations.
- *Objective Function*: Several objective functions have been proposed in the literature for searching for optimal hyper-parameters, e.g. radius margin bound [2], span bound [3], support vector count [4], etc. More information about them can be found in [6]. Unfortunately, these measures usually depend on certain assumptions, e.g. they are valid for a specific kernel or require a separation of the training set without error. The problem is that these assumptions are quite strong for real-world problems. Thus, the best alternative is to use as objective function measures related to the performance of the classifiers, since no assumptions are needed [15]. Taking this into account, the minimization of the generalization error from cross-validation (CV) procedures over a training set is a good option. In the ν-CV procedure, the original training set is firstly divided into ν portions of data, and then sequentially one dataset is tested by using a classifier trained from the remaining $\nu - 1$ portions of data. To sum up, it means that each instance of the entire training set is predicted once, and the final generalization error is computed as an average over the test errors obtained. In fact, a ν-CV is the best option since it results in a better generalization error estimation than by separating a small dataset into a hold-out procedure and being less computationally expensive than by using leave-one-out procedure (ν = total number of training samples), for example. In this work, we have used $\nu = 5$ (five-fold cross-validation), since it is the most commonly used and is also suggested in [8].

5.3. Datasets

We have used fourteen synthetic and real-world datasets in the experiments. They are listed in Table 3 along with more details. The synthetic problems used were the well-known Circle-in-Square (CiS) [32] and P2 [21] problems. The CiS problem consists of two classes, where the decision boundary is nonlinear and the samples are uniformly distributed in ranges from 0 to 1. A circle inside a square denotes one class, while the other class is formed by the

Table 3
Specifications on the datasets used in the experiments.

Database	Number of classes	Number of features	Number of training samples	Number of sets	Number of test samples
Adult	2	123	3185	19	29,376
Circle-in-Square	2	2	3856	21	10,000
DNA	3	180	1400	15	1186
GermanCredit	2	24	800	13	200
IR-Ship	8	11	1785	10	760
Nist-Digits	10	132	5860	16	60,089/58,646
P2	2	2	3856	21	10,000
Satimage	6	36	4435	15	2000
Segment	7	19	1848	12	462
Svmguide	2	4	3089	20	4000
Splice	2	60	1000	15	2175
Mushrooms	2	112	6498	24	1626
Usps	10	256	7291	17	2007

area outside the circle. The area of the circle is equal to half of the square. The P2 problem is also a two-class problem, where each class is defined in multiple decision regions delimited by one or more than four simple polynomial and trigonometric functions. As in [31], one of the original equations was modified such that the areas occupied by the classes become approximately equal. In both problems, the classes are nested without overlapping, so the total probability of error is 0%.

The real-problems employed are described as follows. The Adult dataset represents a two-class problem from the UCI Repository [33]. The task is to predict whether or not income exceeds $50K/year based on census data. The DNA, German Credit, and Satimage datasets are from the Statlog Project [34]. The DNA dataset is a multi-class problem where each class represents a different protein. The German Credit dataset is a binary-classification problem, where the goal is to classify people as good or bad credit risks based on a set of attributes. The Satimage dataset consists of multi-spectral values of pixels in a satellite image, where the aim is to predict the class of central pixels in 3×3 neighborhoods, given the multi-spectral features. The Nist-Digits is a dataset composed of samples from the NIST Digits Special database 19 (NIST SD19). Composed of handwritten samples of 0 to 9 digit images, this dataset is one of the most popular real-world databases employed to evaluate handwritten digit recognition methods. We have used two distinct test sets denoted as Nist-Digits 1 (60,089 samples) and Nist-Digits 2 (58,646 samples) in this paper. Both are partitions of the NIST's Special Database 19: hsf-4 and hsf-7, respectively. The former is considered to be more difficult to classify than the latter. Samples from hsf-0123 partitions were used as training set. The feature set employed is the same as that suggested by Oliveira et al. [35]. Basically, the features are a mixture of concavity, contour and character surface, where the final feature vector is composed of 132 components normalized between 0 and 1. The IR-Ship database is a military database which consists of Forward Looking Infra-Red (FLIR) images of eight different classes of ships. The images were provided by the U.S. Naval Weapons Center and Ford Aerospace Corporation. The same feature set employed by Park and Sklansky [36] was used in this work. Segment, Splice, Mushrooms, and Usps are also databases from [33]. The Segment database contains instances randomly drawn from outdoor images. Each instance is a 3×3 region, where each region represents a class, such as: brickface, sky and foliage. The Splice database is composed of samples of DNA sequences, where the problem is to classify them into IE (intron/exon) or EI (exon/intron) boundaries. The Mushrooms database includes descriptions of samples corresponding to 23 species of gilled mushrooms. Each species is identified as definitely edible or poisonous. The Usps database is composed of images of isolated digits with 300 pixels/in in 8-bit gray scale on a high-quality flat bed digitizer. Finally, the Svmguide problem is a two-class database that involves an astroparticle application [8].

5.4. Parallel processing

In order to speed up the execution of our experiments, we have implemented the PSO algorithm and our proposed method in a parallel processing architecture (a Beowulf cluster with 20 nodes using Athlon XP 2500+processors with 1 GB of PC-2700 DDR RAM (333 MHz FSB)). The optimization algorithms were implemented using LAM MPI v6.5 in master-slave mode with a simple load balance. It means that while one master node executes the main operation related to the control of the processes, like the updating of particles' positions/velocities, and then switching between the different levels (e.g. adapted grid, DPSO), the evaluations of fitness are performed by several slave processors. The results obtained are given in subsequent sections.

5.5. Obtained results

The results are reported in Tables 4–6, in terms of generalization error rates, number of stored support vectors, and computational time spent, respectively. It is important to mention that these results were tested on multiple comparisons using the Kruskal–Wallis nonparametric statistical test by testing the equality between mean values. The confidence level was set to 95% and the Dunn–Sidak correction was applied to the critical values. The best results for each classification problem are shown in bold. Based on the results, we can see how important a careful selection of hyper-parameters is to generate high performing classifiers. For instance, the results for the GS and 1st-Grid approaches in Table 4 show us that searching for optimal hyper-parameters given a new dataset can achieve better results, in both classification accuracy and model complexity, than those that apply a searching process just once.

In addition, we have observed that PSO based approaches are very promising, since their results have overtaken those of the two grid-search methods (see in Tables 4 and 5). Furthermore, the most important fact is that the proposed method (DMS) was able to attain similar results, but was less time consuming, than the full PSO (FPSO) strategy. As previously mentioned, because some of the model selection strategies (FPSO, CPSO, and DMS) use stochastic algorithms, we have replicated the experiments 10 times. Therefore, the results for these strategies represent averages over 10 replications. All these results, mainly comparing GS *vs* 1st-GS and CPSO *vs* DMS, are particularly interesting because they confirm the importance of tracking optimal solutions when new data are available and show the relevance of the proposed method. By analyzing the results, we can say that by shifting between re-evaluations and re-optimizations of previous swarms can be quite effective for building new solutions. The adapted grid module is less time consuming and performs better than evaluating, a grid randomly composed of 70 different combinations (GS), for instance,

Table 4
Mean error rates and standard deviation values over 10 replications when the size of the dataset attained the size of the original training set. The best results for each data set are shown in bold.

Database	GS	1st-GS	FPSO	CPSO	DMS
Adult	17.54	24.06	**15.55 (0.06)**	23.85 (0.01)	**15.56 (0.05)**
CiS	0.34	0.67	**0.14 (0.03)**	0.19 (0.03)	**0.13 (0.03)**
Dna	12.82	42.24	**5.13 (0.18)**	6.37 (0.44)	**5.16 (0.56)**
GermanCredit	30.00	35.00	**26.6 (0.21)**	30.10 (0.32)	**26.65 (0.31)**
IR-Ship	6.05	7.50	**4.86 (0.35)**	5.66 (0.45)	**4.72 (0.29)**
Nist-Digits 1	2.82	6.84	**2.75 (0.04)**	3.02 (0.23)	**2.74 (0.14)**
Nist-Digits 2	7.38	14.30	**6.68 (0.15)**	7.33 (0.59)	**6.72 (0.39)**
P2	1.79	3.71	**1.64 (0.10)**	2.03 (0.29)	**1.69 (0.14)**
Satimage	10.20	10.50	**8.06 (0.13)**	14.32 (0.30)	**8.26 (0.22)**
Segment	**2.81**	4.33	**2.78 (0.52)**	4.87 (2.04)	**2.80 (0.9)**
Svmguide	13.15	50	**3.10 (0.01)**	3.97 (0.02)	**3.11 (0.07)**
Splice	12.38	12.38	**10.40 (0.92)**	11.9 (2.10)	**10.45 (1.10)**
Mushrooms	0.00	0.00	0.00	0.00	0.00
Usps	10.16	10.21	**6.44 (0.15)**	8.41 (0.19)	**6.35 (0.08)**

Table 5
Mean of support vectors and standard deviation values obtained over 10 replications when the size of the dataset attained the size of the original training set. The best results for each data set are shown in bold.

Database	GS	1st-GS	FPSO	CPSO	DMS
Adult	1508	1572	**1176.50 (12.53)**	3075.00 (10.00)	**1174.80 (12.66)**
CiS	64	476	**35.40 (6.47)**	**43.30 (12.18)**	**37.40 (8.36)**
Dna	1906	1914	628.40 (32.50)	**436.10 (42.83)**	810.60 (31.69)
German Credit	800	516	**418.40 (3.63)**	776.50 (74.31)	**421.30 (9.74)**
IR-Ship	443	661	**320.70 (13.34)**	671.40 (21.74)	**318.70 (9.53)**
Nist-Digits	**880**	2912	**898.40 (30.45)**	1556.30 (62.56)	**947.40 (55.09)**
P2	226	430	**161.40 (26.12)**	383.50 (77.37)	**152.80 (8.47)**
Satimage	1117	**1073**	1888.00 (93.51)	1384.10 (60.64)	1849.00 (99.64)
Segment	**251**	298	**218.30 (79.39)**	381.3 (135.85)	**281.7 (72.75)**
Svmguide	2801	3003	**245.50 (7.90)**	254.5 (3.44)	**246.8 (5.37)**
Splice	959	959	499.80 (176.85)	**326.10 (32.17)**	444.50 (22.39)
Mushrooms	1102	1102	**240.80 (89.15)**	**245.10 (30.48)**	**244.30 (38.21)**
Usps	4200	4199	**1115.20 (91.74)**	1702.20 (164.70)	**1152.50 (58.40)**

or starting a whole new optimization process (FPSO). Besides, it was shown that the DPSO algorithm is capable of tracking optimal solutions by resetting the particles' memories and injecting diversity.

In order to better quantify and visualize the performance of the methods over all the sets, we have also reported the mean error rates across all the subsets and over the 10 replications in the two case studies in Fig. 10 for the IR-Ship and Satimage databases.

For a deeper analysis of the proposed method, we have depicted in Fig. 11 the frequencies of at which a module was responsible for the selection of the final solution. From these results, it is even possible to guess the different degrees of difficulty among the databases. For example, databases whose the final solutions were pointed out more often by the DPSO module, e.g. German Credit and DNA, seem to have a major degree of uncertainty, due perhaps a greater overlapping between classes, than other databases, such as Nist-Digits and CiS, for example.

By comparing the optimization approaches directly, we can see that the results reported in Table 7 demonstrate that our DPSO implementation is advantageous, mainly in terms of the processing time demanded to search for solutions. Unlike FPSO, which requires several iterations, because it starts a new search randomly every time, our dynamic version saves time by applying dynamic optimization techniques, such as: the use of previous knowledge and increasing diversity. As a result, when the DPSO module is activated, it converges faster and with similar results to those obtained with FPSO and better than those obtained with CPSO.

The results also reveal an important advantage of our dynamic model selection strategy (DMS) over the common used FPSO

Table 6
Mean computational time spent (hh:mm:ss) for model selection processes for the entire sequences of datasets with the most promising strategies. Results for the FPSO strategy over the entire databases (FPSO-all data) are also reported.

Database	FPSO-all data	FPSO	CPSO	DMS
Adult	01:28:07 (00:38:13)	02:41:36 (00:02:53)	01:37:21 (00:01:07)	**00:32:31 (00:02:50)**
CiS	02:56:15 (00:55:41)	05:07:17 (00:09:59)	2:23:10 (00:06:12)	**01:35:45 (00:08:34)**
Dna	00:34:59 (00:15:59)	01:07:58 (00:01:34)	00:42:27 (00:00:39)	**00:14:21 (00:01:01)**
GermanCredit	**00:07:51 (00:00:57)**	00:13:43 (00:00:06)	00:11:36 (00:00:02)	00:13:17 (0:00:05)
IR-Ship	00:19:08 (00:07:41)	00:30:42 (00:01:01)	00:15:17 (00:04:09)	**00:11:26 (00:05:00)**
NistDigit	06:47:51 (02:22:15)	13:46:00 (00:16:04)	03:46:24 (00:08:33)	**00:56:38 (00:05:34)**
P2	06:02:28 (00:48:29)	16:04:54 (00:17:44)	10:21:50 (00:13:47)	**05:35:55 (00:33:24)**
Satimage	01:45:55 (00:38:40)	02:46:18 (00:03:41)	01:41:29 (00:02:22)	**01:31:03 (00:05:04)**
Segment	00:01:51 (00:00:38)	00:04:15 (00:00:44)	00:02:11 (00:00:31)	**00:00:38 (00:00:43)**
Svmguide	**00:43:24 (00:33:39)**	01:44:03 (00:38:15)	01:10:12 (00:17:43)	**00:41:30 (00:39:07)**
Splice	**00:00:39 (00:00:12)**	00:01:35 (00:00:20)	00:01:35 (00:00:15)	**00:00:51 (00:00:23)**
Mushrooms	00:51:40 (00:07:37)	02:02:21 (00:08:53)	01:37:23 (00:03:17)	**00:01:39 (00:01:27)**
Usps	06:10:53 (02:14:33)	14:13:41 (03:05:37)	12:35:26 (03:28:36)	**05:31:42 (02:46:18)**

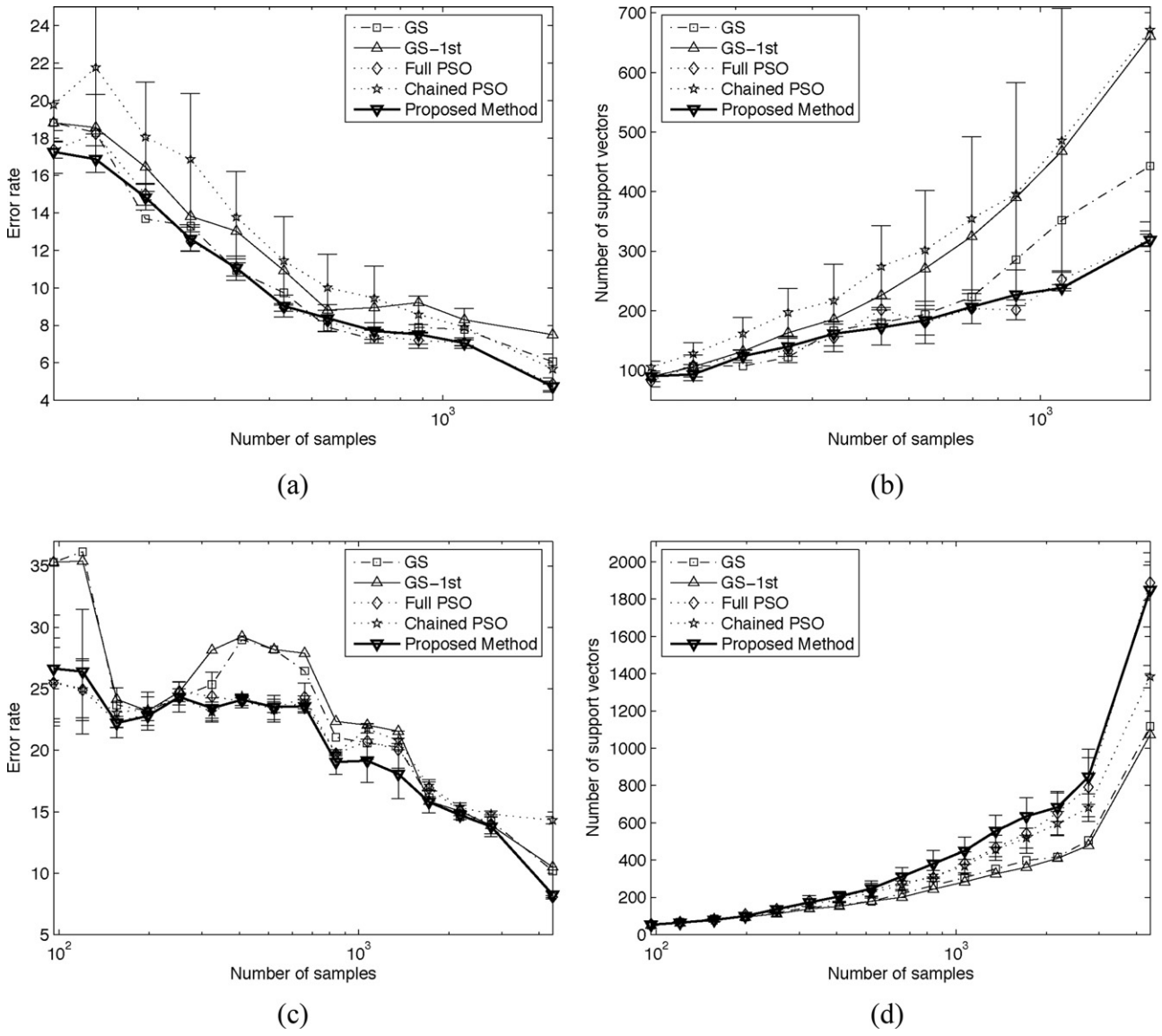


Fig. 10. Error and support vectors rates. For the databases, Ship ((a) and (b)) and Satimage ((c) and (d)). The results were obtained over 10 replications.

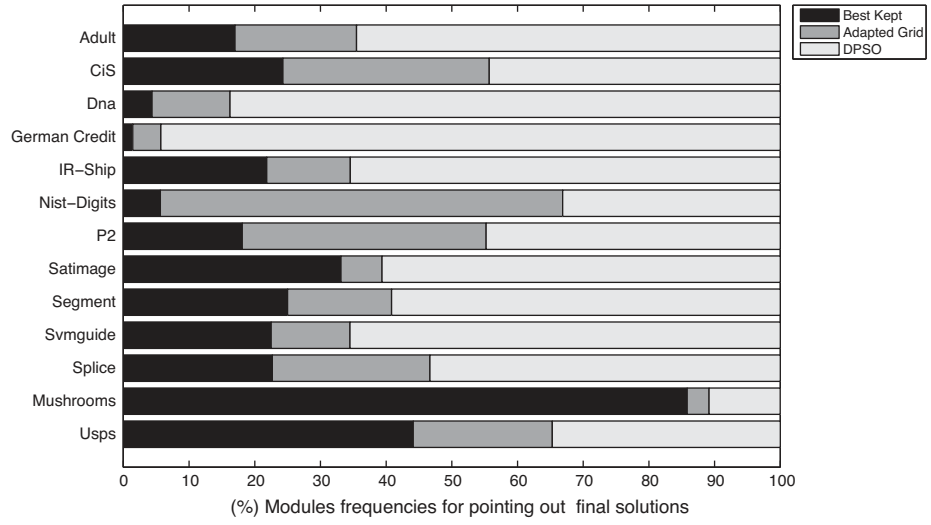


Fig. 11. Average of frequencies which indicates how many times each module was responsible for pointing out the final solution.

Table 7
Mean of number iterations attained and standard deviation values for each optimization algorithm over 10 replications. The results for the Full and Chained PSO strategies were computed over all datasets. In contrast, the results for the DPSO module were computed considering only the datasets where it was activated.

Database	Full PSO	Chained PSO	DPSO module
Adult	18.63 (7.04)	12.00 (1.04)	14.66 (4.37)
CiS	23.53 (7.52)	17.71 (2.92)	17.05 (6.10)
Dna	23.18 (7.07)	15.95 (5.47)	17.08 (5.09)
GermanCredit	21.48 (7.44)	12.61 (2.14)	14.07 (4.34)
IR-Ship	30.45 (8.78)	15.82 (5.34)	17.38 (5.20)
Nist-Digits	31.60 (8.44)	14.17 (4.74)	15.72 (6.41)
P2	27.39 (9.26)	20.72 (5.81)	15.50 (4.61)
Satimage	24.86 (8.19)	16.78 (6.53)	18.14 (6.48)
Segment	29.70 (3.16)	20.32 (1.88)	17.01 (5.34)
Svmguide	40.91 (2.41)	32.11 (1.22)	16.50 (5.05)
Splice	33.24 (2.34)	26.69 (1.70)	17.75 (6.23)
Mushrooms	36.38 (1.13)	29.6 (0.47)	13.65 (1.87)
Usps	38.73 (2.58)	31.31 (2.80)	19.86 (6.09)

strategy. While a huge amount of computational time was required for the FPSO optimization approach to perform the model selection processes, our proposed method was capable of finding satisfactory solutions in less computational time, by mainly considering it for each set of data. This is because the FPSO strategy requires a large number of evaluations than the proposed method, especially over each dataset, or still because when applied gradually over the datasets, the proposed method usually accelerates the searching process by approximating solutions before reaching the total size of training sets. Based on these results, we can see that the proposed method, DMS, has spent less computational time than the other strategies. Besides, it can also be noted that sometimes the application of DMS gradually over subsets of data can be even faster than realizing a full optimization process over the entire original training set.

Ending, the efficiency of the proposed method was demonstrated through the results. Even though the strategies sometimes perform similarly in terms of generalization errors, as in the case of the CiS database, the proposed method is clearly superior with respect to other factors, e.g. the model complexity (number of support vectors) and computational time. Furthermore, by taking fewer iterations and having adaptation capabilities, the use of the proposed method in a fully dynamic environment is very promising, mainly in those applications where the system must adapt itself to new data (time-series data, for example).

6. Conclusion

In this work we presented the SVM model selection problem as a dynamic optimization problem which depends on available data. In particular, it was shown that if one intends to build efficient SVM classifiers from different, gradual, or serial source of data, the best way is to consider the model selection process as a dynamic process which can evolve, change, and hence require different solutions overtime depending on the knowledge available about the problem and uncertainties in the data.

In order to solve the model selection problem and also take into account this dynamism, we proposed a PSO-based framework (DMS) based on the ideas of self-organization, change detection, and dynamic optimization techniques to track the optimal solutions and save computational time. The relevance of the proposed method was confirmed through experiments conducted on six databases. Briefly, the results have shown that: (1) if PSO is applied sequentially over datasets as a whole optimization process (Chained PSO) with the purpose of saving computational time, the resulting optimized solutions may stay trapped in local minima after successive hyper-parameter model selection processes. On the other hand, (2) although full optimization processes with PSO (Full PSO strategy) constitute an efficient way to achieve good results, they are very time consuming, particularly when applied to each new dataset. (3) DMS was very similar to full optimization processes, but less computationally expensive, mainly due to the use of the dynamic optimization techniques.

Ending, in this paper we examined the SVM model selection problem in a gradual learning context where hyper-parameters must be re-estimated in order to retrain an SVM classifier from data at different times k in a cumulative fashion, as occurs in applications where data collection is expensive, such as cancer diagnosis and signature verification. The proposed method is also particularly useful for real-world applications requiring the generation of new classifiers dynamically in a serial way (e.g. those involving streaming data), since by taking fewer iterations and having adaptation capabilities, it promises to be very useful in a fully dynamic environment.

Acknowledgements

This research was supported by Defense Research and Development Canada, DRDC-Valcartier under the contract W7701-2-4425 and in part by grant OGP0106456 to Robert Sabourin from the NSERC of Canada.

References

[1] M.N. Kapp, R. Sabourin, P. Maupin, A PSO-based framework for dynamic SVM model selection, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2009, pp. 1227–1234.
[2] V.N. Vapnik, Statistical Learning Theory, Wiley, NY, 1998.
[3] O. Chapelle, V. Vapnik, Model selection for support vector machines, in: Advances in Neural Information Processing Systems, 1999, pp. 230–236.
[4] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer Verlag, NY, 1995.
[5] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge University Press, 2000.
[6] O. Chapelle, V. Vapnik, O. Bousquet, S. Mukherjee, Choosing multiple parameters for support vector machines, Machine Learning 46 (1–3) (2002) 131–159.
[7] N. Ayat, M. Cheriet, C. Suen, Automatic model selection for the optimization of SVM kernels, Pattern Recognition 38 (10) (2005) 1733–1745.
[8] C.C. Chang, C.J. Lin, Libsvm: A Library for Support Vector Machines, 2005, URL http://www.csie.ntu.edu.tw/cjlin/libsvm/.
[9] C.-W. Hsu, C.-J. Lin, A comparison of methods for multiclass support vector machines, IEEE Transactions on Neural Networks 13 (2) (2002) 415–425.
[10] C.-M. Huang, Y.-J. Lee, D.K. Lin, S.-Y. Huang, Model selection for support vector machines via uniform design, Computational Statistics & Data Analysis 52 (1) (2007) 335–346.
[11] Z. Chunhong, J. Licheng, Automatic parameters selection for SVM based on GA, in: Proceedings of the 5th World Congress on Intelligent Control and Automation, 2004, pp. 1869–1872.
[12] G. Cohen, M. Hilario, A. Geissbuhler, Model selection for support vector classifiers via genetic algorithms. An application to medical decision support, in: Proceedings of the 5th International Symposium on Biological and Medical Data Analysis, 2004, pp. 200–211.
[13] T. Suttorp, C. Igel, Multi-objective optimization of support vector machines, in: Multi-Objective Machine Learning, Vol. 16 of Studies in Computational Intelligence, Springer, 2006, pp. 199–220.
[14] C. Chatelain, S. Adam, Y. Lecourtier, L. Heutte, T. Paquet, Multi-objective optimization for SVM model selection, in: Proceedings of the 9th International Conference on Document Analysis and Recognition, 2007, pp. 427–431.
[15] F. Friedrichs, C. Igel, Evolutionary tuning of multiple SVM parameters, in: Proceedings of the 12th European Symposium on Artificial Neural Networks, 2004, pp. 519–524.
[16] B.F. de Souza, A.C.P.L.F. de Carvalho, R. Calvo, R.P. Ishii, Multiclass SVM model selection using particle swarm optimization, in: Proceedings of the 6th International Conference on Hybrid Intelligent Systems, 2006, pp. 31–34.
[17] M. Jiang, X. Yuan, Construction and application of PSO-SVM model for personal credit scoring, in: Proceedings of the International Conference on Computational Science, Lecture Notes in Computer Science, 2007, pp. 158–161.
[18] G. Valentini, Ensemble Methods based on Bias-variance Analysis, Ph.D. Thesis, University of Genova, Genova, Italy, 2003.
[19] C. Domeniconi, D. Gunopulos, Incremental support vector machine construction, in: Proceedings of the International Conference on Data Mining, 2001, pp. 589–592.
[20] C.P. Diehl, G. Cauwenberghs, SVM incremental learning, adaptation and optimization, in: Proceedings of the International Joint Conference on Neural Networks, 2003, pp. 2685–2690.

[21] G. Valentini, An experimental bias-variance analysis of SVM ensembles based on resampling techniques, IEEE Transactions on Systems Man and Cybernetics Part B 35 (6) (2005) 1252–1271.
[22] A. Shilton, M. Palaniswami, D. Ralph, A.C. Tsoi, Incremental training of support vector machines, IEEE Transactions on Neural Networks 16 (2005) 114–131.
[23] Y. Jin, J. Branke, Evolutionary optimization in uncertain environments—A survey, IEEE Transactions on Evolutionary Computation 9 (3) (2005) 303–317.
[24] L. Cohen, G. Avrahami-Bakish, M. Last, A. Kandel, O. Kipersztok, Real-time data mining of non-stationary data streams from sensor networks, Information Fusion 9 (3) (2008) 344–353.
[25] J. Kennedy, R.C. Eberhart, Particle swarm intelligence, in: Proceedings of the International Conference on Neural Networks, 1995, pp. 1942–1948.
[26] J. Kennedy, Some issues and practices for particle swarms, in: Proceedings of the IEEE Swarm Intelligence Symposium, 2007, pp. 801–808.
[27] M. Clerc, J. Kennedy, The particle swarm—explosion stability and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (1) (2002) 58–73.
[28] A. Carlisle, G. Dozier, Tracking changing extrema with adaptive particle swarm optimizer, in: Proceedings of the 5th Biannual World Automation Congress, Orlando, USA, 2002, pp. 265–270.
[29] X. Hu, R.C. Eberhart, Adaptive particle swarm optimization: detection and response to dynamic systems, in: Proceedings of the Congress on Evolutionary Computation, 2002, pp. 1666–1670.
[30] M. Clerc, Particle Swarm Optimization, ISTE Publishing Company, London, 2006.
[31] P. Henniges, E. Granger, R. Sabourin, Factors of overtraining with fuzzy ARTMAP neural networks, in: Proceedings of the International Joint Conference on Neural Networks, 2005, pp. 1–4.
[32] G.A. Carpenter, S. Grossberg, J.H. Reynolds, ARTMAP: supervised real-time learning and classification of nonstationary data by a self-organizing neural network, Neural Networks 4 (5) (1991) 565–588.
[33] C.L. Blake, C.J. Merz, UCI Repository of Machine Learning Databases, 1998, URL http://www.ics.uci.edu/~mlearn/MLRepository.html.
[34] D. Michie, D.J. Spiegelhalter, C.C. Taylor, Machine Learning. Neural and Statistical Classification, ftp.ncc.up.pt/pub/statlog/.
[35] L.S. Oliveira, R. Sabourin, F. Bortolozzi, C.Y. Suen, A methodology for feature selection using multi-objective genetic algorithm for handwritten digit string recognition, International Journal of Pattern Recognition and Artificial Intelligence 17 (6) (2003) 903–930.
[36] Y. Park, J. Sklansky, Automated design of linear tree classifiers, Pattern Recognition 23 (1990) 1393–1412.