A Dynamic Optimization Approach for Adaptive Incremental Learning

Marcelo N. Kapp,[1,2,*] Robert Sabourin,[1,†] Patrick Maupin,[3,‡]
[1]*École de technologie supérieure, Université du Québec, Montréal (Québec) H3C 1K3, Canada*
[2]*Universidade Federal da Integração Latino-Americana, Foz do Iguaçu (Paraná) 85856-970, Brazil.*
[3]*Defence Research and Development Canada–Valcartier: G3J 1X5, Saint-Gabriel-de-Valcartier, Quebec, Canada*

A fundamental problem when performing incremental learning is that the best set of a classification system's parameters can change with the evolution of the data. Consequently, unless the system self-adapts to such changes, it will become obsolete, even if the application environment seems to be static. To address this problem, we propose a dynamic optimization approach in this paper that performs incremental learning in an adaptive fashion by tracking, evolving, and combining optimum hypotheses overtime. The approach incorporates various theories, such as dynamic particle swarm optimization, incremental support vector machine classifiers, change detection, and dynamic ensemble selection based on classifiers' confidence levels. Experiments carried out on synthetic and real-world databases demonstrate that the proposed approach actually outperforms the classification methods often used in incremental learning scenarios.

1. INTRODUCTION

A fundamental problem when performing incremental learning is that the best set of a classification system parameters can vary overtime due to changes in the incoming data. An example of such changes would be minor fluctuations (random or systematic[1]) in the underlying probability distributions. These usually result from either sample shifting or the natural evolution of classification problems, considering that new knowledge comes partially from new observations made at different times. Therefore, the sample distributions of training datachunks may change and affect the system in several ways, since they serve as the basis for estimating its decision boundaries. In the literature, these possible data changes are defined as *population drifts*.[2,3] The problem with these changes in incremental learning scenarios is that

*Author to whom all correspondence should be addressed: e-mail: kapp@livia.etsmtl.ca.
†e-mail: robert.sabourin@etsmtl.ca.
‡e-mail: patrick.maupin@drdc-rddc.gc.ca.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 26, 1101–1124 (2011)

View this article online at wileyonlinelibrary.com. • DOI 10.1002/int.20501

they are unavoidable, even if the application environment seems to be static (i.e., the number of classes and features always remains constant).

Consequently, the incremental updating of a classification system might require a review of its existing models in terms of not only the knowledge acquired and the new data but also setting its internal parameters with respect to such data variations. Otherwise, the whole system may become obsolete, and hence fail to achieve a better adaptation in the future. This assumption might explain why, despite significant research effort[3–7] into the design of incremental learners, the results are not often as satisfactory as batch mode results (i.e., when all data are considered). Taking this into account, unlike the traditional incremental learning approaches that consider the adjustment of parameters as a static process (i.e., constant parameter values are employed infinitely), we propose to optimize them over time to increase the system's power of generalization and decrease its complexity.

To achieve this, our underlying hypothesis in this paper is to consider incremental learning as a dynamic optimization process, in which optimum hypotheses are dynamically tracked, evolved, and combined over time. On this basis, we introduce a method to perform adaptive incremental learning based on these two principles: (1) incremental accommodation of new data by updating models and (2) dynamic tracking of new optimum system parameters for self-adaptation. The proposed method relies on a new framework incorporating various techniques, such as single incremental support vector machine (ISVM) classifiers, change detection, dynamic particle swarm optimization (DPSO), and, finally, dynamic selection of classifier ensembles (EoC). Thus, the goal is to update, evolve, and combine multiple heterogeneous hypotheses (i.e., models with different parameters and knowledge) over time to maintain the optimality of the system with respect to its internal parameters, computational cost, and generalization performance.

In particular, the use of ISVM ensembles in this study is justified based on two main pieces of evidence found in the literature. First, the fact that the classification success of the SVM does not depend on the dimensions of the input space makes it a robust classifier against the well-known *curse of dimensionality* , which mainly involves small data sets. This in turn makes it very advantageous for incremental learning situations. Second, SVM ensembles are used because their performances can often surpass those of single models, especially when the level of uncertainty is high, i.e. when only small sample sets are available.[8] Furthermore, the proposed framework contributes to strategies to optimize and overproduce classifiers, as well as to the application of memory-based mechanisms for solving dynamic optimization processes. This latter is a promising and ongoing research area.

Finally, we validate the proposed method and show its efficiency through experiments with synthetic and real-world databases. Results in single and multiple classifier configurations are compared with those obtained with these strategies: SVMs optimized with PSO in batch mode, ISVMs with parameter values fixed beforehand, and two increment-capable classifiers (1-NN (nearest neighbours) and naïve Bayes), which are widely applied in incremental learning studies. These classifiers are tested because they are considered to perform no less well than their batch versions, i.e. "no less" classifiers.[9] An incremental ensemble strategy with optimized parameters and different combination rules is also applied for comparison purposes.

Additional objectives of this study are to verify whether or not: (1) incremental learning with SVMs can achieve similar performances to those obtained in batch mode, (2) the adaptation of the system's parameters over time is actually a dynamic optimization problem, and hence important to achieving high performances, and (3) the dynamic selection of ensembles can lead to better results than simply combining all the pools of classifiers available.

The remainder of this paper is organized as follows. Section 2 presents the incremental learning concept, problems typically faced, and the main related approaches in the literature. In Section 3, we introduce the proposed method for adaptive incremental learning. Experimental results are reported in Section 4. Conclusions and suggestions for future work are presented in Section 5.

2. RELATED WORKS

Incremental learning approaches are very attractive for solving several real-world classification problems, especially those where (1) the data acquisition process is expensive, so that only a few samples become available over time; (2) the data generation is itself time dependent, an example being time series data; and (3) the training data set available is too large to be loaded into computer memory.[3] The main incremental learning approaches introduced in the literature can be organized in two groups: (1) single incremental learning classifiers and (2) ensembles of classifiers.

1. *Single Incremental Learning Classifiers:* Approaches in this group use classifier algorithms capable of incremental learning. For example, there are some traditional classifiers that are naturally suited to incremental learning, such as naive Bayes and K-nearest neighbors[10] classifiers, others specifically developed for this task, e.g., ARTMAP neural networks,[4] and, finally, incremental versions of classical classifiers, such as decision trees[11] and support vector machines (SVMs). In the case of SVMs, various versions have been proposed, grouped here in two categories, according to the way the incremental process is conducted:

    - *Manipulating sample sets (MSS):* These incremental SVM methods update a classifier by merging new data, old support vectors, and optional additional samples considered relevant in an iterative training procedure. Nonimportant samples are discarded after being used for recursively training and testing the model generated.[3,12–14]
    - *Preserving Karush–Kuhn–Tucker conditions (PKC):* These SVM incremental training algorithms are an attempt to incrementally approximate an optimal decision boundary by adding a new sample to the solution through "adiabatical" updates in Lagrangian coefficients (α_i) and retaining the Karush–Kuhn–Tucker conditions on all data previously seen.[15,16] A sample-discarding procedure is implemented based on the leave-one-out estimate of generalization error on the whole training set. Despite incremental training, the leave-one-out procedure makes these methods computationally expensive.

    It is worth noting that, for most ISVM methods, the term "incremental" denotes the iterative building of an SVM model from recursive procedures over a large data set. Thus, their goal is to accelerate the training of an SVM classifier by avoiding solving a larger quadratic optimization problem, as required by the standard SVM formulation.[17]

2. *Ensemble Learning:* Ensemble of classifiers (EoC) methods incrementally learn new data by adding new classifiers or by updating the combination rule of existing classifiers from datachunks.[7,18–20] The idea is to train a base classifier from each datachunk and then combine them for future predictions. The combination function may or may not be updated from new data. This general scheme is called as *random aggregation* or *block evolution*.[21] Examples of ensemble approaches would be some variants of the traditional Bagging algorithm, e.g., Ref. 9, variations of the Adaboost algorithm, such as in Refs. 5 and 7, or even the pasting small votes method,[22] which trains consecutive classifiers from filtered data.

However, from this literature review, we note that, whatever the incremental learning approach, no consideration has been given to the tuning of system parameters over time. In other words, system updating is always performed based on the same fixed parameter values or at the classifier combination level in the ensemble approaches. Thus, the updating of classifiers with the adaptation of their parameters has not yet been investigated. Furthermore, recent results indicate that using well-tuned incremental learners could achieve better performances than just moderate ones.[7]

From the above, we can see that the incremental updating of existing classifiers without compromising their performances remains a major challenge. This is because updating can be affected by possible variations in a problem's data distributions (i.e., population drifts), which disturb the process of selection of the system parameters, and hence estimation of decision boundaries at different times. This may happen mainly when classification problems involve complex decision boundaries or overlapping between classes.

To overcome this challenge, an incremental learning system must be able not only to accommodate new data at no detriment to knowledge already acquired[5] but also to try to better adapt its parameters. Taking this into account, we propose an approach for adaptive incremental learning that regards the incremental learning process as a dynamic optimization process. In particular, it applies knowledge acquired from previous optimization processes to decrease the computational cost of frequent reoptimizations. From an optimization point of view, our assumption is that the natural data changes mentioned above are sources of uncertainty reflected in dynamisms in the parameter search space. Such uncertainties are intensified even more when the search for optimum parameter values must be performed over time. In the literature, dynamic optimization problems are categorized in three groups: Those where the location of the optimum changes over time and the amount of shift is quantified by a severity parameter (type I); those where the location remains fixed, but the value of the objective function changes (type II); and those where both the location and the value change (type III).[23] In this paper, we will demonstrate empirically that the optimization of classifiers over time can actually be seen as a type-III problem (Section 4.3.3).

In addition, from the analysis of SVM ensembles presented in Ref. 8, which shows that a single SVM classifier's performances can be improved over small data sets by combining "heterogeneous" SVMs in terms of parameters, the proposed approach is implemented for evolving and combining a population of optimum solutions. Likewise, we explore the use of multiple classifiers in an effort to achieve

better performances than with a single learner. However, instead of pickingup parameter values from an arbitrary grid of options, as in Ref. 8, we explore the self-organization power of swarm intelligence theory and the dynamic optimization techniques. In this way, our method is able to dynamically move a population of solutions toward optimum regions in the system parameter search space. Finally, our aim is to combine an optimized population of hypotheses that is well placed over the search space, and can even be multimodal, and hence to make the system more robust than if only a single model is used. The proposed method is introduced in the next section.

3. PROPOSED ADAPTIVE INCREMENTAL LEARNING METHOD

The proposed method herein for adaptive incremental learning (AIL) optimizes, selects, and combines incremental SVM classifiers over time. More specifically, it is designed to dynamically point out optimum solutions for sequences of data sets $\mathcal{D}(k)$ by using best solutions found so far, or by starting new dynamic optimization processes. As we employ incremental support vector machine as our base classifiers and dynamic particle swarm optimization for searching for optimum hyperparameter values, each solution $\mathbf{s}$ represents a particle codifying an SVM hyperparameter set, e.g., $\{C, \gamma\}$. Change detection mechanisms monitor novelties in the objective function $\mathcal{F}$ and indicate how the system must act. The models generated are updated from incoming data and then dynamically selected and combined into an ensemble $\mathcal{C}^*$. A complete description of the method as well as details on its framework is presented below.

3.1. Framework for Adaptive Incremental Learning

To implement the proposed method, we have designed a framework which is composed of five main modules, as shown in Figure 1. These modules are change detection, adapted grid search (AG), DPSO, ISVMs, and decision fusion. In particular, this framework incorporates the many upgrades made since our first version, introduced in Ref. 24, such as the use of incremental classifiers, dynamic selection, and the building of ensembles from optimized models. Below, details on each module are provided. The *upgrade_stm* and *recall_stm* functions are responsible for storing and retrieving optimized solutions and important data from the system's short-term memory (STM). Δ represents a set of data sv composed of support vectors and relevant samples rs selected during the training of the final classifier from the best particle $\mathbf{s}^*$. Therefore, $\Delta = \{sv^* \cup rs\}$, where sv^* refers to support vectors obtained from the final incremental model $\mathcal{M}^*$ trained with hyperparameters found by the best particle $\mathbf{s}^*$. SV denotes the set of support vectors sv from incremental models obtained after final training of all P particles from a swarm $\mathcal{S}(k-1)$, i.e., $SV = \{sv_j\}_{j=1}^{P}$. $\mathcal{C}$ represents an ensemble composed of all models (i.e., classifiers) $\mathcal{M}_i$. So, $\mathcal{C} = \{\mathcal{M}_i\}_{i=1}^{P}$, where P is the maximum number of optimized solutions. Finally, for the sake of simplicity, in the equations, $\mathcal{D}(k)$ represents the merging of

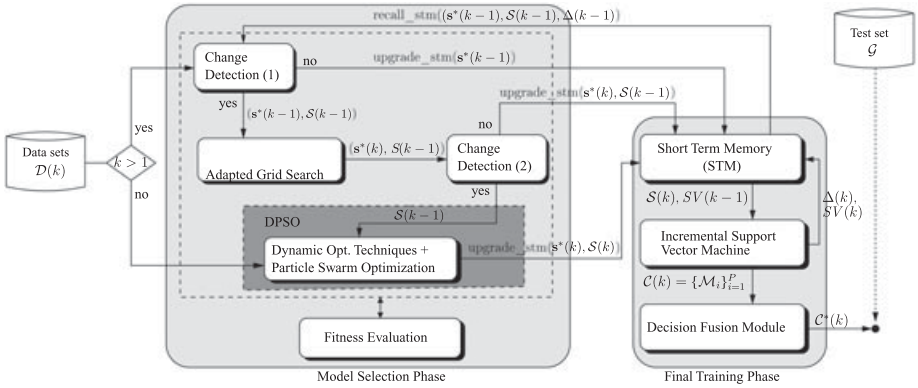


Figure 1. General framework of the proposed method for incremental learning with dynamic SVM model selection. Δ represents a set of data sv^* composed of support vectors and relevant samples rs selected during the training of the final model $\mathcal{M}$ from the best particle $\mathbf{s}^*$. So $\Delta = \{sv^* \cup rs\}$, where sv^* means support vectors obtained specifically from final model $\mathcal{M}$ trained with hyper-parameters found by best particle $\mathbf{s}^*$ and $SV = \{sv_j\}_{j=1}^{P}$ denotes the set of support vectors sv from models obtained after final training of all P particles from a swarm $\mathcal{S}(k-1)$.

new data and the current knowledge stored by the method (i.e., Δ, as defined above, is composed of relevant samples and support vectors detected by the best solution found so far).

In particular, the goal of this framework is to perform adaptive incremental learning by dynamically optimizing, selecting, and combining optimum solutions over time. It works as summarized in Algorithm 1. For each new data set $\mathcal{D}(k)$, the method is designed to decide whether (1) new dynamic optimization processes must be activated beforehand to search for new solutions $\mathbf{s}$ (i.e., classifier parameters) stored in the system memory, or (2) a previous population of solutions $\mathcal{S}(k-1)$ can be reused to save computational time (steps 1–4). The decision is made based on previously acquired knowledge, new data, and through change detection mechanisms that monitor the behavior of the system with respect to novelties in the parameter search space and indicate how the system must act (steps 5–14). Finally, the system's memory with new solutions and incremental classifiers, which are generated from such solutions, are updated and then dynamically selected and combined into an ensemble of classifiers $\mathcal{C}^*$ to improve the overall performance of the classification system (steps 15 and 16).

3.1.1. Change Detection Module

The change detection module controls the intensity of the search process by indicating how the solutions are found through the search levels of the framework. In particular, it is responsible for simultaneously monitoring the quality of the model selection process and avoiding "unnecessary" search processes. We implement it by monitoring differences in the objective function values, in this case the error estimations ϵ obtained for a best solution $\mathbf{s}^*$ on the data sets $\mathcal{D}(k-1)$ and $\mathcal{D}(k)$,

Algorithm 1 Adaptive Incremental Learning (AIL)

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 data set $\mathcal{D}(k)$
  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/update/combine the final incremental SVM classifiers from incoming data $\mathcal{D}(k)$, $\Delta(k)$, and SV.

for example. We denote this by $\epsilon(\mathbf{s}^*, \mathcal{D}(k-1))$ and $\epsilon(\mathbf{s}^*, \mathcal{D}(k))$, respectively. If the solution found is not satisfactory for the process, then a further search level is activated. The adequacy of a solution can be measured in several ways. If the objective function value computed does not lie in a "stable" region, then, as we are interested in finding effective solutions, we will consider that further searches are needed. The stable region is computed through the maximum expected difference $\delta_{\max}$ between the objective function values at the 90% confidence level using a normal approximation to the binomial distribution (see Equations 1 and 2).[6] In this setting, if there is a degradation in performance ($\epsilon(\mathbf{s}^*, \mathcal{D}(k-1)) < \epsilon(\mathbf{s}^*, \mathcal{D}(k))$) or significant variation in the objective function (i.e., $\mid \epsilon(\mathbf{s}^*, \mathcal{D}(k-1)) - \epsilon(\mathbf{s}^*, \mathcal{D}(k)) \mid \geq \delta_{\max}$), then other levels are activated for additional searches. If the $\delta_{\max}$ rule is respected, the situation is characterized as stable and no further searches are computed. Otherwise, other modules are activated.

$$\delta_{\max} = z_{0.9} \times \sqrt{\sigma} = 1.282 \times \sqrt{\sigma} \tag{1}$$

where σ is computed by, where $W(\cdot)$ is the data set size:

$$\sigma = \frac{\epsilon(\mathbf{s}^*, \mathcal{D}(k-1)) \times (1 - \epsilon(\mathbf{s}^*, \mathcal{D}(k-1)))}{W(\mathcal{D}(k-1))} + \frac{\epsilon(\mathbf{s}^*, \mathcal{D}(k)) \times (1 - \epsilon(\mathbf{s}^*, \mathcal{D}(k)))}{W(\mathcal{D}(k))} \tag{2}$$

So, the change detection module may sometimes signal a trade-off between computational time spent and the quality of solutions. If we ignore this module, then dynamic reoptimization processes will always be conducted. This can produce good results, although it is unnecessarily time consuming for stable cases.

3.1.2. Adapted Grid Search

The AG module identifies optimum solutions through the reevaluation of knowledge acquired from previous optimizations carried out by our DPSO module with respect to the current data $\mathcal{D}(k)$. This knowledge is represented by a set $\mathcal{S}(k-1)$ of optimized solutions, which are stored in STM. Therefore, unlike a traditional grid search, which depends on the discretization and evaluation of several combinations of hyperparameter values, the AG module reduces the number of trials by focusing the search in an optimum region. In other words, it uses the best positions of preceding optimized solutions as a grid of candidate solutions to be evaluated over the current data. At the end of the process, the best candidate is selected, potentially saving considerable computational time. However, if the best candidate is still not satisfactory for final learning purposes, then the dynamic optimization module is activated, as shown in the framework in Figure 1.

3.1.3. Dynamic Particle Swarm Optimization

The DPSO module offers new optimum solutions by means of dynamic optimization processes, the goals of which are (1) to work on multimodal search spaces and (2) to track changes of fitness landscapes and optimum point positions, since these can vary depending on the incoming data received over time. We implement this module based on the PSO algorithm combined with dynamic optimization techniques. The PSO method was first 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 fish. We apply it in this work because of its many advantages, such as the following: (1) It performs continuous codification, making it ideal for the search of optimal SVM hyperparameters and (2) its potential for adaptive control and flexibility (e.g., self-organization and division of labor) provided by swarm intelligence theory, making it very interesting for solving dynamic optimization problems. In this section, we simplify the index notation (e.g., for time or data sets) and use only those needed to understand the PSO technique. Standard PSO involves a set $\mathcal{S} = \{\mathbf{s}_i, \dot{s}_i, \mathbf{s}'_i\}_{i=1}^{P}$ [a] of particles that flies in the search space looking for an optimal point in a given d-dimensional solution space. $\mathbf{s}_i = (s_i^1, s_i^2, \ldots, s_i^d)$, which is a vector containing the set of values of the current hypothesis, represents the current location of the particle in the solution space, where the number of dimensions is problem dependent. The vector $\dot{s}_i = (\dot{s}_i^1, \dot{s}_i^2, \ldots, \dot{s}_i^d)$ stores the velocities for each dimension of the vector $\mathbf{s}_i$ that are responsible for changing the direction of the particle. The vector $\mathbf{s}'_i = (s_i'^1, s_i'^2, \ldots, s_i'^d)$ is a copy of the vector $\mathbf{s}_i$ that 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

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

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 comprises the sum of the information from the best particle's informant found in its neighborhood λ (e.g., $s'_{\text{net(i,}\lambda)}(q)$) and the particle's own experience $\mathbf{s}'_i$. For a new iteration $q+1$ and dimension d, the update is computed as follows:

$$\dot{s}_i^d(q+1) = \chi\left(\dot{s}_i^d(q) + \phi r_1\left(s_i'^d(q) - s_i^d(q)\right) + \phi r_2\left(s'^d_{\text{net}(i,\lambda)}(q) - s_i^d(q)\right)\right) \quad (3)$$

where χ is the constriction coefficient,[26] and r_1 and r_2 are random values. Constriction coefficient values of $\chi = 0.7298$ and $\phi = 2.05$ are recommended.[27] 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:

$$s_i^d(q+1) = s_i^d(q) + \dot{s}_i^d(q+1) \quad (4)$$

In this study, we implement this module with a PSO based on the local best (*lbest*) topology,[27] which 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, interactions are allowed among neighborhoods, and potentially more series of events may be discovered. This enables the module to explore multiple regions in parallel, which works better for functions with possible *multiple local optima*. Such a parallelism allows distant neighborhoods to be explored more independently, which is an important characteristic of our method, since the particles can be located in the optimum region for use by the AG module or to restart the dynamic search process and save computational time for future optimizations. Nevertheless, even though PSO is a powerful optimization method, if the optimization problem suffers any change in the objective function, for example, between blocks of data, the particles can become stuck in local minima.

An alternative that would prevent this is to start a full PSO optimization process from scratch each time the module is activated. However, this 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 to save computational time. To implement this mechanism, we combine two dynamic optimization techniques: *rerandomization* and *reevaluation* of solutions and apply them in our PSO-based module. In fact, both techniques have already been applied in the PSO literature[28,29] to solve dynamic optimization problems, but separately and using the *gbest* topology. These PSO variants are commonly called DPSO (dynamic PSO), and so, for the sake of simplicity, we call our module DPSO to indicate that it comprises such a combination of approaches. It is nevertheless important to point out that existing DPSO algorithms apply such techniques and change detection mechanisms at each iteration, since they suppose that objective function changes can occur during optimization. Here, as the optimization over a data set $\mathcal{D}(k)$ at a given instant k is static, we apply these dynamic techniques to prepare the optimization module for transitions from preceding optimization knowledge to launch new ones.

As a result, we take advantage of these techniques to provide diversity in the solutions and clues as to optimal starting points before the optimization. Thus, unlike actual DPSO versions, no extra computational effort is added at each iteration. In light of this, in Figure 1 and in the rest of this paper, our DPSO module represents the application of these dynamic techniques to work with the optimization algorithm, but not within it, at each iteration. Therefore, the entire implementation involves two main steps related to the way in which the optimization process restarts. The main steps are listed in Algorithm 2. First, with activation of the DPSO module, which also uses information from the system's memory (STM), every fitness value is updated from the reevaluation 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 3–6). This is done to prevent the particle's memory from becoming obsolete.[28] In fact, the fitness of the best positions $\mathbf{s}'_i$ can profit from the preceding level (AG), which triggers a second evaluation. Then, a reoptimization process is launched, keeping $\rho\%$ of the best particle positions from the swarm $\mathcal{S}(k-1)$ computed in the previous optimization, and randomly creating new particles over the search space.[29] Some of these particles are located near the previous optimum region. In this way, we guarantee that fine searches are conducted 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 for new solutions, which enables us to avoid situations where the whole swarm has already converged to a specific area. Finally, steps 9–23 correspond to the main steps of the PSO implementation, but are slightly modified with the addition of a mechanism that updates the connections among the particles, if no improvement is observed between iterations (steps 19–21). These latter steps were suggested as an alternative to improve the adaptability, and hence the performance, of the swarm.[30]

3.1.4. Incremental Support Vector Machine

The SVM classifier is a machine-learning approach based on the structural risk theory introduced by Vapnik in Ref. 17, which is capable of finding the optimal hyperplane that separates two classes. In this study, we implement an incremental SVM version based on the Syed et al. method[3] for three reasons: (1) It focuses on the updating of models over sequences of data sets over time; (2) it has produced the best results in this comparative study[13]; and (3) it does not require the tuning of extra parameters, which may need careful setting, as occurs in Refs. 12–14. This is important because the setting of extra parameters can be critical, since they control when samples might be exchanged among temporary sets or when learning processes should stop. Moreover, the SVM implementation used here[31] already provides mechanisms to accelerate SVM training through the sequential minimal optimization (SMO) technique, which requires less computational effort than traditional quadratic programming solvers. As in Ref. 3, an incremental SVM model $\mathcal{M}_i(k)$ is trained on the current training datachunk $\mathcal{D}(k)$ and its historical support vectors $sv(k-1)$ identified from previous learning at a given time k. However, in contrast to Ref. 3, where only support vectors are stored, our incremental SVM module retains additional training samples, relying on a "relevant region," which

Algorithm 2 Our implementation of Dynamic PSO

Input: PSO parameters and previous swarm $\mathcal{S}(k-1)$.
Output: Optimized solutions.
for all particles i from $\mathcal{S}(k-1)$ such that $1 \leq i \leq P$ **do**
  Compute fitness values for $\mathbf{s}_i$ using $\mathcal{D}(k)$
  Update $\mathbf{s}'_i$ if $\mathbf{s}_i$ is better ($\mathbf{s}'_i \leftarrow \mathbf{s}_i$)
end for
Initialize dynamically the new swarm $\mathcal{S}(k)$ by keeping $\rho\%$ of the best information (positions $\mathbf{s}'_i$) from the preceding swarm $\mathcal{S}(k-1)$ and by creating new particles.
Initialize the links among the particles based on a nearest neighborhood rule according to the topology chosen.
$q \leftarrow 0$;
repeat
  for all particles i such that $1 \leq i \leq P$ **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 of this iteration q, i.e. $\mathbf{s}'_i(q)$
  for all particles i such that $1 \leq i \leq P$ **do**
    Update velocity $\dot{s}_i(q)$ (Equation 3) and current position $\mathbf{s}_i(q)$ (Equation 4)
  end for
  if $\mathcal{F}(\mathbf{s}^*(q)) = \mathcal{F}(\mathbf{s}^*(q-1))$ {No improvement. Change particle communication structure} **then**
    Randomly change the particles' links based on the topology chosen.
  end if
  $q = q + 1$
until maximum iterations or other stop criteria be attained

exceeds the SVM margins by half their size. It is interesting to note that, even if we fix this region at this size, the size of this region will vary according to the difficulties that arise in classification problems (e.g., complex decision boundaries, overlapping between classes) and in the hyperparameters selected. Although the storage of additional samples is not a desirable property in incremental learning algorithms,[5] it is necessary because such samples can become support vectors during optimization of SVMs in the future.

3.1.5. Decision Fusion Module

The decision fusion module dynamically selects incremental classifiers and combines them into ensembles. Our dynamic selection strategy is implemented based on a generalization bound introduced in Ref. 32, which we first studied in its application to "static" SVM ensembles in Ref. 33. With this strategy, only a combination of classifiers that minimizes this bound (called the CI measure here) is selected to compose the final ensemble. In particular, this measure is computed as $CI = \sigma(\tau)/\mu(\tau)^2$, where σ and μ denote the variance and the average calculated over the set of margins τ from samples of the current training set, respectively.

The margin of a sample $\mathbf{x}_i$ represents a degree of confidence in its classification. Basically, it is calculated as the difference between the decision support ϑ assigned to the true class t minus the highest support estimated for any other class j, i.e., $\tau_i = \vartheta_t(\mathbf{x}_i) - \max_{j=1,\ldots,c\, j\neq t}\{\vartheta_j(\mathbf{x}_i)\}$. Here, for a single classifier, the decision support for a class j is denoted as the posterior probability assigned to it. In the same way, for an ensemble composed of classifiers with output probabilities, the decision support for a class j is the average over the posterior probabilities assigned to it by each member. The selection process is performed as follows: First, the pool of classifiers $\mathcal{C}(k)$ generated from $\mathcal{S}(k)$ is sorted according to each classifier's individual confidence level (average margins). Then, the selection process starts by adding one classifier at a time until the maximum number of classifiers is reached, i.e., a number of particles P. Each time a classifier is added; the CI selection criterion is recomputed. The best ensemble selected, $\mathcal{C}^*$, is that with a minimum CI value, the key idea being to select the ensemble with the strongest, i.e., highest, confidences, and fewest correlated classifiers over the current training set. Finally, once the best ensemble, $\mathcal{C}^*$, is selected, the classifiers are combined using weighted average voting based on their performances. With different criteria, however, forward searches for the best ensembles seem to be very promising.[34] To calibrate the outputs of SVM classifiers in terms of probability estimates, we have used the approach introduced by Wu et al.,[35] which is the one implemented in the LIBSVM software.[31]

4. EXPERIMENTAL PROTOCOL

To validate the concept of our adaptive incremental learning system as well as to show the efficiency of the proposed method, the following experimental protocol has been carried out. First of all, to characterize the occurrence of population drifts with greater impact, the original training sets were divided into small data sets. The total number of data sets and their sizes were determined based on a minimum number of samples required for each class, which was set to at least 16. The distribution of samples was first separated for the class with fewer samples, and then proportionally for the other classes. In this way, the total number of chunks was determined, and the same original proportion of samples per class was kept in each datachunk. In other words, if the original problem contained unbalanced classes, then that real scenario is simulated in this experimental protocol. A detailed description of the data sets and the number of chunks used are listed in Table I. We selected classification problems with different numbers of features, classes, and training and testing samples. As the proposed method uses a stochastic algorithm, the results represent averages drawn over 10 replications.

4.1. Strategies Tested

The following incremental learning strategies were tested:

1. *Batch SVM-PSO*: In this strategy, the entire original training data sets are used for selecting optimum SVM hyperparameters and training the final model. The hyperparameter

Table I. Specifications on the data sets used in the experiments.

Databases	Number of classes	Number of features	Number of chunks	Number of training samples	Number of test samples
Adult	2	123	48	3,185	29,376
Circle-in-Square	2	2	120	3,856	10,000
DNA	3	180	29	2,000	1,186
German	2	24	15	800	200
IR-Ship	8	11	8	1,785	760
Nist-Dig 1/2	10	132	36	5,860	60,089/58,646
P2	2	2	120	3,856	10,000
Satimage	6	36	25	4435	2,000

selection process is carried out with the PSO algorithm. This strategy represents an empirical lower bound computed for each problem, which allows us to compare the results obtained for incremental strategies with those obtained for a batch strategy.

2. *Incremental* no-less *classifiers (1-nearest neighbor (1-NN) and naive Bayes (NB))*: These two classifiers were tested because they are widely used in both the incremental learning and concept drift literature,[9,10,19] since they are considered *no less* incremental learners, i.e. their results in incremental mode are similar to those obtained in batch mode.[9]
3. *Incremental SVM (ISVM)*: In this strategy, an ISVM classifier tailored from Ref. 3 is updated from successive datachunks $\mathcal{D}(k)$. Its hyperparameters are first tuned with PSO over the first datachunk $\mathcal{D}(1)$ and then kept fixed over all the other datachunks. No relevant samples are kept during the incremental learning process.
4. *Optimized random aggregation (ORA)*: This is a common incremental ensemble approach, specifically, a random aggregation approach as described in Section 2. Here, it consists of combining SVM classifiers with optimum hyperparameter values trained from independent datachunks in a serial way (i.e., one classifier per datachunk). Two combination rules were tested with this scheme: majority voting and simple average voting. We set the maximum ensemble size to 20. When the total number is reached, the oldest model is replaced by the new one.
5. *Single incremental (IS-AIL)*: This is the proposed approach in single classifier mode (i.e., only the best solution found so far is used by the decision fusion module).
6. *Incremental EoC-AIL (IEoC-AIL)*: The proposed approach in EoC mode.

4.2. Experiment Parameter Setting

- *Optimization algorithm parameters*: The maximum number of iterations and the swarm size were set to 100 and 20, respectively. The maximum and minimum values of the dimensions of the (C and γ) parameter search space were set to $[2^{-6}, 2^{14}]$ and $[2^{-15}, 2^{10}]$, respectively. The DPSO topology used was the $lbest$, with $\lambda = 3$. We also consider stopping the optimization if the best fitness value does not improve over 10 consecutive iterations.
- *Objective function*: Several objective functions have been proposed for searching for optimum SVM hyperparameters, e.g., radius margin bound and span bound.[36] Unfortunately, these measures depend on certain assumptions, e.g., they are related to a specific kernel or require a separation of the training set without error, which are quite strong for real-world problems. Thus, the minimization of the generalization error from the ν-cross-validation procedure is a

good option. $\nu = 5$ is used here, as suggested in Ref. 31. The results for each strategy are presented in the next section.

4.3. Results

4.3.1. Performance Evaluation

First, we report the generalization errors achieved by each strategy tested in Table II. These results were tested with multiple comparisons using the Kruskal–Wallis nonparametric statistical testing of 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 and incremental learning strategy are shown in bold. The underlined values indicate when one incremental strategy is significantly better than the others. Analysis of these results shows that SVM is very promising for incremental learning, since there is a relevant difference between results on the first datachunks, i.e., SVM-PSO ($\mathcal{D}(1)$), and results after all the datachunks have been learned. This is the case even with the hyperparameters kept fixed at a value found on $\mathcal{D}(1)$ (ISVM). More importantly, we observe the efficiency of the proposed method and conclude that adaptive incremental learning clearly leads to better performances. That is because the single-classifier version of our proposed method (IS-AIL) has obtained better results than the common ISVM strategy. This shows the importance of the adaptation of hyperparameters and the use of relevant samples during the incremental learning process. Furthermore, it can be observed that the proposed method (IEoC-AIL) achieved results similar to SVM-PSO results in batch mode, and sometimes even better than those. This proves that the dynamic selection and combination of optimum solutions can actually improve the overall performance of the system. Figures 2a and 2b illustrate these results with two case studies concerning these generalization error results with the best performing strategies during different incremental learning steps at times k.

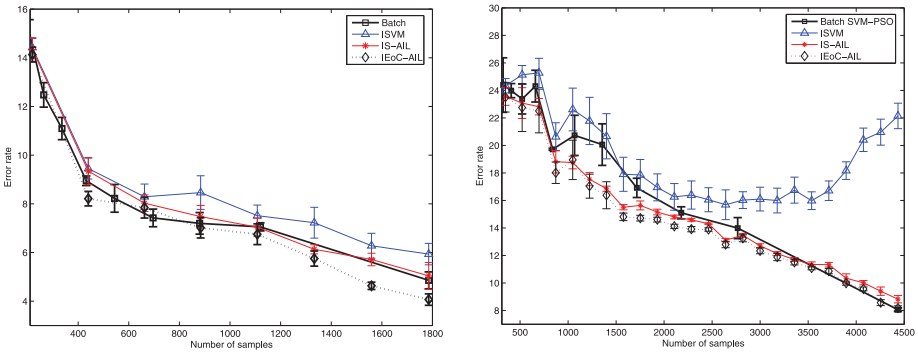
Moreover, it can be seen that the serial incremental ensemble approaches (i.e., the ORA strategies, ORA-MV, and ORA-SA) performed well, especially on noisy data (e.g., as for the Adult database), although they were not statistically superior to the proposed method in these tests.

In contrast, the need to set a maximum number of classifiers is a determining factor in the performance of these methods, since some knowledge may be lost when the oldest classifier is replaced with a new one. This is a drawback, because the results with a single incremental learner (ISVM) were better than those with the two ensemble approaches for some problems (e.g., IR-Ship, German). Our results indicate that the updating of an existing ISVM classifier might be very advantageous compared to only combining batch learners (ISVM*vs*ORAs). The results with the ORA-SA approaches (ORA-SA and ORA-MV) have shown that the simple average fusion function is superior to the majority voting rule. Finally, the classical "no less" incremental learners NB and 1-NN achieved the worst performances. The only exception was for the CiS and P2 databases, where the 1-NN classifier outperformed the other methods tested, but of course, with the inconvenience of having to store all the data.

Table II. Mean and standard deviation of error rates obtained after learning from all subsets available.

	Approaches tested								
Databases	SVM-PSO	SVM-PSO($\mathcal{D}(1)$)	1-NN	NB	ISVM	ORA-MV.	ORA-SA.	IS-AIL	IEoC-AIL
Adult	15.55 (0.06)	24.77 (0.80)	24.06	24.05	23.93 (0.50)	20.07 (1.46)	**19.29 (1.53)**	20.83 (4.34)	20.52 (1.60)
CiS	0.14 (0.03)	11.47 (0.42)	**1.02**	7.78	3.60 (0.97)	3.91 (0.63)	2.68 (0.45)	1.43 (0.80)	1.35 (0.29)
Dna	5.13 (0.18)	21.02 (0.48)	23.69	6.32	8.43 (1.48)	9.74 (0.15)	9.14 (0.20)	4.71 (0.25)	**4.61 (0.27)**
German	26.6 (0.21)	30.75 (0.77)	34.50	31.00	29.60 (1.26)	30.05 (0.15)	30.01 (0.01)	28.85 (0.22)	**28.15** (0.56)
IR-Ship	4.86 (0.35)	14.42 (0.37)	9.21	30.92	7.93 (0.44)	8.63 (0.81)	8.33 (0.73)	5.04 (0.55)	**4.03 (0.30)**
NistDig-1	2.75 (0.04)	18.10 (0.32)	3.85	6.92	3.92 (0.40)	8.39 (0.05)	7.93 (0.04)	2.71 (0.04)	**2.64 (0.01)**
NistDig-2	6.68 (0.15)	31.96 (0.37)	7.97	13.66	8.42 (0.29)	16.46 (0.08)	16.09 (0.08)	6.33 (0.10)	**6.27 (0.07)**
P2	1.64 (0.10)	29.65 (0.23)	**2.49**	42.38	5.24 (0.14)	13.26 (1.76)	10.95 (1.30)	4.80 (0.90)	3.17 (0.56)
Satimage	8.06 (0.13)	22.00 (0.52)	10.95	20.45	22.15 (0.93)	18.09 (0.50)	17.69 (0.63)	8.83 (0.27)	**8.14 (0.17)**

Shown in bold are the best results concerning the incremental strategies. Underlined values indicate when an incremental strategy was significantly better than the others. Results were computed over mean values drawn from 10 replications.



(a) IR-Ship (b) Satimage

Figure 2. Case study: Comparison among generalization error results for batch and the most promising incremental strategies.

4.3.2. Data Storage and Complexity of the Models Generated

With respect to the complexity factor of the ISVM classifiers generated, Table III summarizes some results regarding the mean number of support vectors stored up to the end of the incremental learning process. By comparing these results, we note that the dynamic adaptation of the hyperparameters during the incremental learning process (IS-AIL) seemed to converge to the results obtained in batch mode (SVM-PSO). In other words, it tends to identify about the same number of support vectors as when all the data are available for training. In contrast, the incremental single SVM classifier strategy with constant hyperparameters (ISVM) did not adjust its models as effectively as the other SVM-PSO and IS-AIL strategies. Of course, these results are related to the single-classifier strategies. However, the complexity of the ensemble version, IEoC-AIL, may be relatively higher, once the number of classifiers has been dynamically selected between 1 and P. Thus, in spite of the fact that the proposed method (IEoC-AIL) represents a remarkable improvement in terms of generalization power, it can also make the system more complex.

These experiments reveal one of the most attractive advantages of the incremental learning approaches, which is their ability to reduce the size of the training set (results shown in Table IV). The training set size reduction rate was computed as follows: the total database size minus the total number of updating samples used by the proposed method in the last incremental learning step divided by the total database size.

It can be seen that the reduction can be very expressive for some problems, especially with two classes and no overlapping, such as for the CiS problem. Training size reduction is of interest because it accelerates the updating of classifiers, mainly for multiclass problems (e.g., for NistDig with a reduction rate of 61.23%). In addition, in the same Table IV, we also report the percentage of relevant samples stored by the proposed method relative to the current total number of support vectors stored and the incoming data in the last incremental learning process. We can see that

Table III. Mean and standard deviation of number of support vectors obtained after learning from all subsets available.

	Approaches tested		
Databases	Batch - PSO	ISVM	IS-AIL
Adult	1176.50 (12.54)	1140.40 (57.85)	1178.7 (70.36)
CiS	35.40 (6.47)	24.50 (11.19)	30.10 (6.11)
Dna	628.40 (32.50)	385.90 (55.82)	640.90 (56.49)
German	306.7 (5.94)	735.80 (74.47)	426.20 (55.24)
IR-Ship	320.70 (13.34)	291.10 (5.51)	347 (13.88)
NistDig	898.40 (30.45)	729.00 (21.56)	913 (27.56)
P2	161.40 (26.12)	82.50 (10.90)	113.00 (63.44)
Satimage	1888.00 (93.51)	825.00 (66.92)	1855.30 (167.12)

Table IV. Training set size reduction (%) using incremental learning instead of batch mode calculated over the last set (first column). Proportion of relevant samples (%) inside the last incremental training set used.

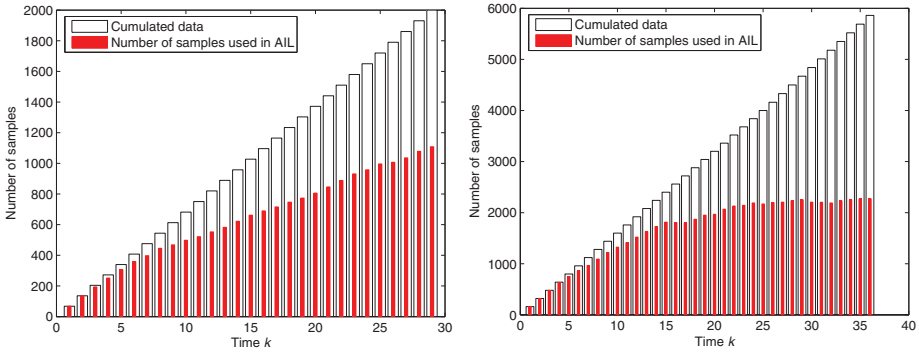
Data sets	Adult	CiS	DNA	German	IR-Ship	NistDig	P2	Satimage
Train. size reduction (%)	47.28	97.86	44.59	31.40	44.86	61.23	95.56	19.85
Proportion of relevant samples (%)	7.97	19.56	18.02	12.54	44.69	53.59	16.18	45.71

the number of relevant samples may vary depending on the problem, the number of classes, data distributions, and the density of samples in the pertinent regions defined by the incremental module.

To better illustrate this reduction effect, we show comparisons between the number of training samples used by the proposed method and what should be stored if batch mode is employed involving the two problems in Figure 3. It can be noted that the number of training samples retained during the system's updating processes may vary, depending on the problem and the number of samples. For example, in Figure 3a, the number of samples is smaller than in batch mode, but it seems that the values will always increase. However, as can be observed in Figure 3b for another problem, when more samples are learned after a longer period of time, the storage may tend to become saturated.

4.3.3. The Dynamism of the System Parameters

These experiments also empirically confirm our underlying hypothesis about the importance of considering the incremental learning process as a dynamic optimization problem. To demonstrate this, we present some results to demonstrate the shifting and tracking of optimum solutions over the search space, given sequences of data sets $\mathcal{D}(\cdot)$. Through the case study depicted in Figure 4, we show that the hyperparameter selection process is actually a type-III dynamic optimization problem. In this example, the search space covered by optimal solutions (denoted here



(a) DNA (b) NistDig

Figure 3. Comparison between the number of training samples used by the method and batch mode. The number of training samples retained during system's updating processes depends on factors such as the overlapping between classes, margin width, and density of samples in these regions.

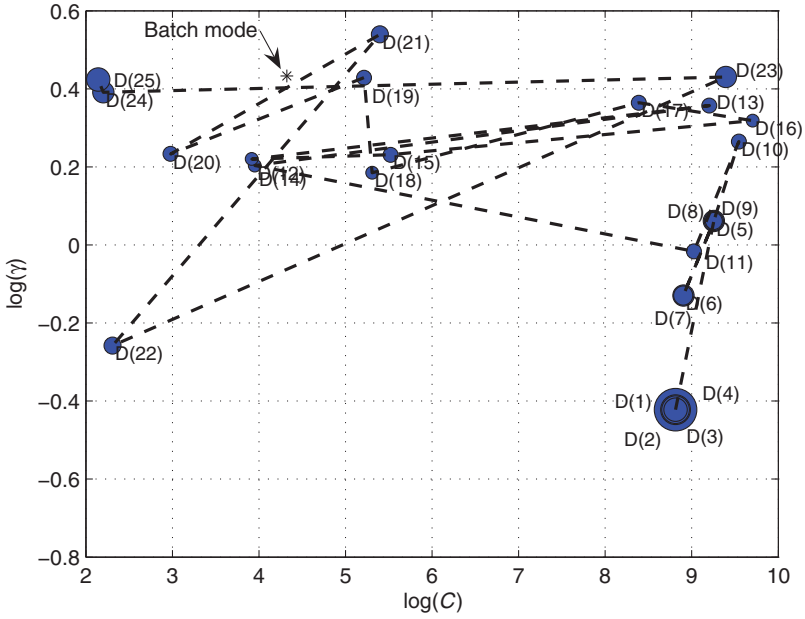
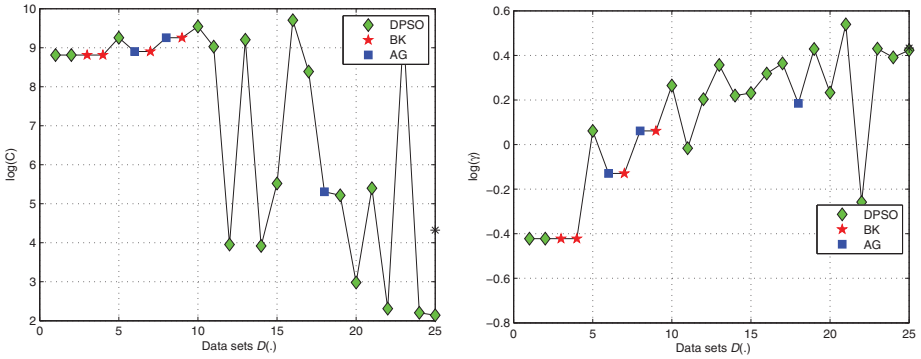


Figure 4. Trajectory covered by the best solution found (circles) from incremental steps for each new data set $\mathcal{D}(k)$. The circles' sizes illustrate how the solutions' fitness can vary. Symbol * depicts a best solution position found if the whole training data are used at once (batch mode).

as circles) is depicted for each data set $\mathcal{D}(k)$ from the Satimage database in one replication. The different circle sizes represent the way in which the fitness varies between the values 14.38% and 4.56%. The symbol * depicts a best solution position found when all the training data were used in the searching process.



(a) C (b) γ

Figure 5. Case study: example on how the solutions were pointed out for each data set $\mathcal{D}(k)$, C, and γ hyper-parameters when using IS-AIL.

We can observe that the optimum solutions $\mathbf{s}(k)^*$ can vary in both fitness and hyperparameter values, depending on the data received during the various incremental learning steps. For instance, they can be located in one region for a given interval of datachunks, e.g., between $\mathcal{D}(1)$ and $\mathcal{D}(7)$, and then move to others, e.g., for $\mathcal{D}(8)$ and $\mathcal{D}(17)$, and, finally, for $\mathcal{D}(25)$. This demonstrates that the problem must be dealt with as a dynamic optimization problem and also explains why approaches with fixed parameters (i.e., on $\mathcal{D}(1)$) might perform suboptimally, as shown in Table II, where IS-AIL is compared with ISVM. As well, Figures 5a and 5b report details on which module has identified these solutions for each data set $\mathcal{D}(k)$ and C and γ, respectively. It can be seen that, most of the time, the best values for the hyperparameters have changed and been tracked by the DPSO module.

In contrast, in more stable cases, optimized solutions stored in the system's memory could be favored for the new learning process by being kept (BK) or selected from the AG module. The frequencies of AIL module activation are listed in Table V. In these experiments, the dynamic optimization module has more often produced the final hyperparameter value solution, followed by searches over previous solutions (AG or keeping the best one).

4.3.4. Selection and Fusion of Solutions into Ensembles

We now turn our attention to dynamic ensemble selection. So far, we have seen (Table II) that combining solutions improves the system's overall performance. In

Table V. Frequencies (%) of AIL modules' activations over all the training data sets.

Data sets	Adult	CiS	DNA	German	IR-Ship	NistDig	P2	Satimage
Best Kept	28.13	7.33	16.90	12.00	24.45	6.67	9.66	17.60
Adapted Grid	15.42	11.59	8.97	18.66	10.47	10.56	15.09	10.80
DPSO	56.45	81.08	74.13	69.34	65.08	82.77	75.25	71.60

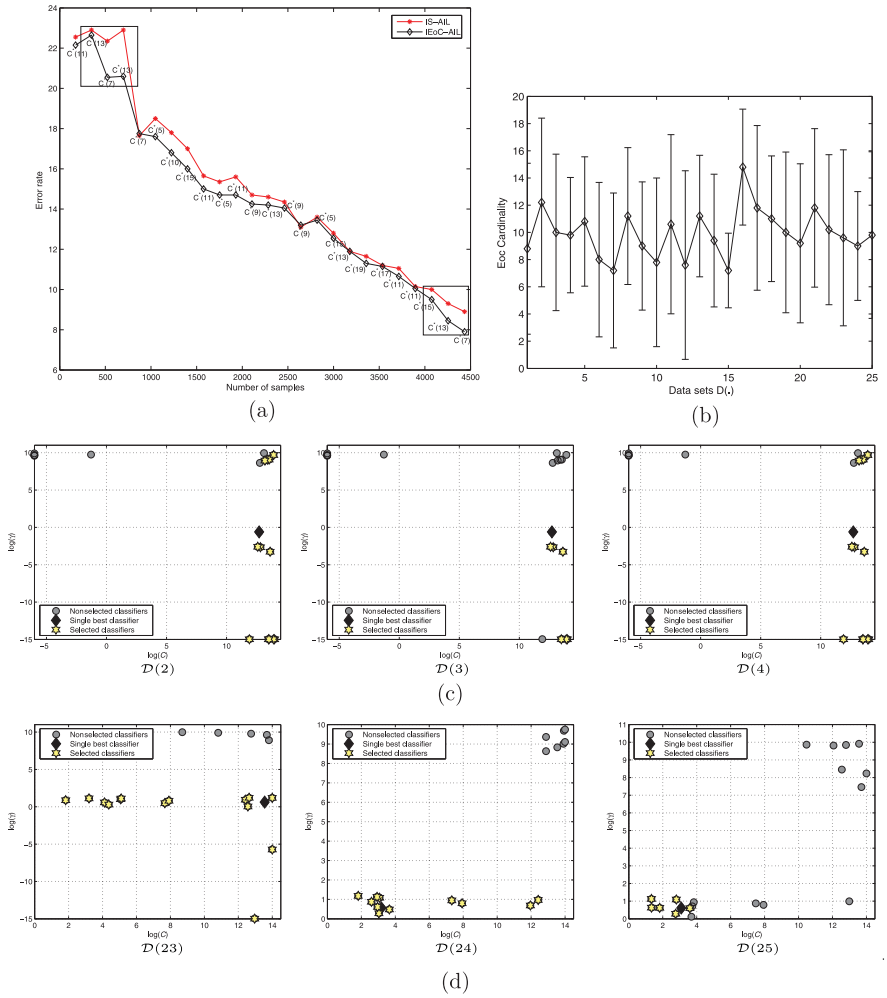


Figure 6. Examples of results involving performances and cardinalities for each data set $\mathcal{D}(k)$ for a given replication compared to AIL in single model (IS-AIL) and ensembles dynamically selected (IEoC-AIL) in (a). In (b), results of EoC cardinalities obtained for each data set $\mathcal{D}(k)$ over 10 replications on Satimage database. Eventually (c) and (d) depict examples of classifiers selected and their original pools distributed over the search space for datachunks outlined by squares in (a).

this section, we describe the effect of our decision fusion module, which is dedicated to dynamic ensemble selection. Figure 6a depicts a case study with the performances and cardinalities of the proposed method in single and ensemble mode over a sequence of datachunks $\mathcal{D}(k)$ from one replication. On the basis of these results, and others listed in this table, we first demonstrate that the dynamic selection of hyperparameters and ensembles is very advantageous in terms of providing stability during the incremental learning process, and hence for achieving better

Table VI. EoC-AIL cardinality after dynamic ensemble selection on the last learning step.

Data set	Adult	CiS	DNA	German	IR-Ship	NistDig	P2	Satimage
Mean	13.40	13.00	7.60	10.60	11.00	8.80	10.60	9.80
(Std)	(5.72)	(2.83)	(4.62)	(5.80)	(6.46)	(4.94)	(6.98)	(6.12)
Median	15	12	8	11	13	11	10	8

performances. We then plot the variations in cardinalities over all the data sets and replications with the same case study (Figure 6b). In Figures 6c and 6d, we can see some of the classifiers selected and the original pools distributed over the search space for data sets outlined by squares in Figure 6a.

We can observe that ensembles with different cardinalities were selected for each time k, when either the optimized swarm $\mathcal{S}(k)$ stays in the same position (c) or moves over the search space (d), which will depend on the problem complexity and current data. Table VI reports some results on the final cardinalities obtained following the last incremental learning process. The number of classifiers selected in the ensemble varied around the mean size of the original pool of 20 members. However, more variations among other datachunks were noted, which indicates that the dynamic selection of classifiers in incremental learning mode is an open issue, and worthy of more extensive study.

Finally, Table VII lists some results obtained for various configurations that we investigated when building our decision fusion module. Three combination functions were used (i.e., majority voting, simple average, and weighted voting), and also three selection criteria, such as none at all (all P classifiers are combined), half-best (the $P/2$ best classifiers), and the CI introduced in Section 3.1.5). By analyzing these results, as was done for the ORA strategies in Table II, the simple average combination function achieved better results than the majority voting rule, and similar to, or slightly worse than, the weighted voting applied to dynamically

Table VII. Mean errors obtained with IEoC-AIL for different combination functions and ensemble selection rules after learning from all series of datachunks available.

	Approaches					
	Majority vote		Simple Average			Weighted vote
Databases	All P	Half best	All P	Half best	CI	CI
Adult	24.03 (0.16)	24.02 (0.15)	23.62 (1.39)	21.58 (1.54)	20.52 (1.71)	20.52 (1.60)
CiS	2.76 (1.49)	2.64 (1.47)	2.36 (1.18)	2.35 (1.19)	2.26 (1.12)	1.35 (0.29)
Dna	4.87 (0.22)	4.71 (0.26)	4.72 (0.26)	4.65 (0.29)	4.61 (0.27)	4.61 (0.27)
German	30.00(0.24)	30.00 (0.24)	30.05 (0.15)	29.95 (0.49)	28.95 (0.36)	28.15 (0.56)
IR-Ship	4.17 (0.15)	4.20 (0.13)	4.12 (0.26)	4.07 (0.23)	4.03 (0.32)	4.03 (0.30)
NistDig - 1	2.65 (0.04)	2.65 (0.04)	2.65 (0.05)	2.65 (0.04)	2.64 (0.03)	2.64 (0.01)
NistDig - 2	6.28 (0.60)	6.27 (0.09)	6.27 (0.07)	6.27 (0.07)	6.27 (0.07)	6.27 (0.07)
P2	8.76 (7.43)	6.58 (5.12)	5.45 (4.32)	4.94 (3.41)	4.18 (1.56)	3.17 (0.56)
Satimage	8.34 (0.19)	8.31 (0.22)	8.18 (0.16)	8.17 (0.16)	8.14 (0.18)	8.14 (0.17)

selected ensembles. Moreover, these results illustrate the importance of the dynamic selection of ensembles, since it improved the results in relation to whole ensembles combined with majority voting. This is possible because dynamic selection ignores classifiers that could introduce some bias into the ensemble's decision and disturb their performances.

5. CONCLUSION

We have proposed a modular dynamic optimization approach to perform adaptive incremental learning. The method generates classifiers from optimum regions of the parameter search space, and then dynamically selects ensembles based on the classifiers' confidence levels to improve the overall results. We show that this process should be treated as a dynamic optimization process, unlike the classical methods, which consider incremental system parameter setting in a static way. This is because the classifiers' optimum parameter values may shift over the search space, depending on the incoming data.

Through experiments on several synthetic and real-world databases, we have empirically demonstrated that the dynamic optimization of an incremental classification system could improve its performances, surpassing those of classifiers without adaptation and of other classical methods. This shows that the performance of a classification system depends on more than merely updating existing models, but also on adapting its internal parameters. Furthermore, we have demonstrated that the application of the latter with a multiple classifier approach makes the classification system more flexible, and at the same time robust to performing incremental learning and dealing with population drifts. As our results are very encouraging, the next stage of this research might involve either (1) determining new strategies for making the system adaptable to real drifts (which demands the design of "forgetting" mechanisms) or (2) carrying on with the study of population drifts, but using semisupervised learning to overcome the dependency of labeled data. Both these research directions should lead to the creation of an even more versatile system.

Acknowledgments

This research was supported in part by Defence Research and Development Canada (DRDC-Valcartier) under contract W7701-2-4425 and by grant OGP0106456 to Robert Sabourin from NSERC Canada.

References

1. Kuncheva LI. Classifier ensembles for changing environments. In: Proc 5th Int Workshop on Multiple Classifier Systems; 2004.
2. Kelly MG, Hand DJ, Adams NM. The impact of changing populations on classifier performance. In: Proc 5th Int Conf on Knowledge Discovery and Data Mining; 1999.

3. Syed NA, Liu H, Sung KK. Handling concept drifts in incremental learning with support vector machines. In: Proc 5th Int Conf on Knowledge Discovery and Data Mining; 1999.
4. Carpenter GA, Grossberg S, Markuzon N, Reynolds JH, Rosen DB. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. IEEE Trans Neural Netw 1992;3(5):698–713.
5. Polikar R, Udpa L, Udpa SS, Honavar V. Learn++: an incremental learning algorithm for supervised neural networks. IEEE Trans Syst Man Cybernet 2001;31(4):497–508.
6. Cohen L, Avrahami-Bakish G, Last M, Kandel A, Kipersztok O. Real-time data mining of non-stationary data streams from sensor networks. Inform Fusion 2008;9(3): 344–353.
7. Parikh D, Polikar R. An ensemble-based incremental learning approach to data fusion. IEEE Trans Syst Man Cybern, Part B: Cybernet 2007;37(2):437–450.
8. Valentini G. An experimental bias-variance analysis of SVM ensembles based on resampling techniques. IEEE Trans Syst Man Cybernet Part B 2005;35(6):1252–1271.
9. Ozawa S, Pang S, Kasabov N. Incremental learning of chunk data for online pattern classification systems. IEEE Trans Neural Netw 2008;19(6):1061–1074.
10. Delany SJ, Cunningham P, Tsymbal A, Coyle L. A case-based technique for tracking concept drift in spam filtering. Knowl-Based Syst 2005;18(4–5):187–195.
11. Hulten G, Spencer L, Domingos P. Mining time-changing data streams. In: Proc 17th Int Conf on Knowledge Discovery and Data Mining; 2001.
12. Mitra P, Murthy CA, Pal SK. Data condensation in large databases by incremental learning with support vector machines. In: Proc 15th Int Conf on Pattern Recognition; 2000.
13. Domeniconi C, Gunopulos D. Incremental support vector machine construction. In: Proc Int Conf on Data Mining; 2001.
14. An J-L, Wang Z-O, Ma Z-P. Incremental learning algorithm for support vector machine. In: Proc 2nd Int Conf on Machine Learning and Cybernetics; 2003.
15. Diehl CP, Cauwenberghs G. SVM incremental learning, adaptation and optimization. In: Proc Int Joint Conf on Neural Networks; 2003.
16. Shilton A, Palaniswami M, Ralph D, Tsoi AC. Incremental training of support vector machines. IEEE Trans Neural Netw 2005;16:114–131.
17. Vapnik VN. The nature of statistical learning theory. New York: Springer Verlag; 1995.
18. Stanley KO. Learning concept drift with a committee of decision trees. Tech. Rep. AI-03-302, Department of Computer Sciences, University of Texas at Austin, Austin, TX; 2003.
19. Wang H, Fan W, Yu PS, Han J. Mining concept-drifting data streams using ensemble classifiers. In: Proc 9th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining; 2003.
20. Tsymbal A, Pechenizkiy M, Cunningham P, Puuronen S. Dynamic integration of classifiers for handling concept drift. Inform Fusion 2008;9(1):56–68.
21. Ganti V, Gehrke J, Ramakrishnan R. Mining data streams under block evolution. ACM SIGKDD Explor Newsl 2002;3(2):1–10.
22. Breiman L. Pasting small votes for classification in large databases and on-line. Mach Learn 1999;36(1–2):85–103.
23. Nickabadi A, Ebadzadeh MM, Safabakhsh R. DNPSO: a dynamic niching particle swarm optimizer for multi-modal optimization. In: Proc IEEE Congress on Computational Intelligence; 2008.
24. Kapp MN, Sabourin R, Maupin P. A PSO-based framework for dynamic svm model selection. In: Proc Genetic and Evolutionary Computation Conference; 2009.
25. Kennedy J, Eberhart RC. Particle swarm intelligence. In: Proc Int Conf on Neural Networks; 1995.
26. Clerc M, Kennedy J. The particle swarm—explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evol Comput 2002;6(1):58–73.
27. Kennedy J. Some issues and practices for particle swarms. In: Proc IEEE Swarm Intelligence Symposium; 2007.

28. Carlisle A, Dozier G. Tracking changing extrema with adaptive particle swarm optimizer. In: Proc 5th Biannual World Automation Congress, Orlando, FL, 2002.
29. Hu X, Eberhart RC. Adaptive particle swarm optimization: detection and response to dynamic systems. In: Proc Congress on Evolutionary Computation; 2002.
30. Clerc M. Particle swarm optimization. London: ISTE; 2006.
31. Chang CC, Lin CJ. Libsvm: a library for support vector machines. Available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm/. Accessed October 12, 2010.
32. Breiman L. Random forests. Mach Learn 2001;45(1):5–32.
33. Kapp MN, Sabourin R, Maupin P. An empirical study on diversity measures and margin theory for ensembles of classifiers. In: Proc 10th Int Conf on Information Fusion; 2007.
34. Ulas A, Semerci M, Yildiz OT, Alpaydin E. Incremental construction of classifier and discriminant ensembles. Inform Sci 2009;179(9):1298–1318.
35. Fan Wu T, Lin C-J, Weng RC. Probability estimates for multi-class classification by pairwise coupling. J Mach Learn Res 2003;5:975–1005.
36. Chapelle O, Vapnik V, Bousquet O, Mukherjee S. Choosing multiple parameters for support vector machines. Mach Learn 2002;46(1–3):131–159.