ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THESIS PRESENTED TO
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
Ph.D.

BY
N. KAPP, Marcelo

DYNAMIC OPTIMIZATION OF CLASSIFICATION SYSTEMS FOR ADAPTIVE INCREMENTAL LEARNING

MONTREAL, MARCH 17 2010

PRÉSENTATION DU JURY

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS :

Mr. Robert Sabourin, Thesis director
Département de génie de la production automatisée at École de technologie supérieure

Mr. Patrick Maupin, Thesis co-director
Recherche et développement pour la défense Canada (Valcartier), groupe de monitoring et d'analyse de la situation.

Ms. Sylvie Ratté, Committee president
Département de génie logiciel et des technologies de l'information at École de technologie supérieure

Mr. Éric Granger, Examiner
Département de génie de la production automatisée at École de technologie supérieure

Mr. Jean Meunier, External examiner
Département d'Informatique et Recherche Opérationnelle at Université de Montréal

THIS THESIS WAS PRESENTED AND DEFENDED

BEFORE A BOARD OF EXAMINERS AND PUBLIC

ON FEBRUARY 10 2010

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGMENTS

It is a pleasure to thank the many people who have helped and inspired me during my doctoral study.

First of all, I would like to express my deep and sincere gratitude to my supervisor, Dr. Robert Sabourin. His understanding, encouraging and personal guidance have provided a fundamental basis for the present thesis. His ideals and concepts have had a remarkable influence on my entire career since my Master degree.

Many thanks also to Patrick Maupin from Defence Research and Development Canada (DRDC Valcartier). His detailed and constructive advices helped me in the development of this work.

I would like to thank the members of my examining committee: Dr. Sylvie Ratté. Dr. Éric Granger, and Dr. Jean Meunier, who have agreed in evaluating this thesis and provided me valuable comments.

Thanks also to my many colleagues in the LIVIA (Laboratoire d'imagerie, de vision et d'intelligence artificielle) who provided a stimulating and friendly environment to learn and grow. Special thanks to Albert Ko, Bassem Guendy, Carlos Cadena, Clement Chion, César Alba, Dominique Rivard, Eduardo Vellasques, Éric Thibodeau, Eulanda Miranda dos Santos, George Eskander, Guillaume Latombe, Jean-François Connoly, Jonathan Bouchard, Jonathan Milgram, Luana Batista, Luis Da Costa, Mathias Adankon, Paulo Cavalin, Paulo Radtke, Philippe LaMontagne, Vincent Doré, and Wael Khreich. I would like to thank Marie-Adelaide Vaz, who has been a very good friend since my arrival in Montreal.

Most of all, thanks to my wife Cinthia for her understanding, endless love, and encouragement which were fundamental for me during all these five years of Ph.D. Even for helping to review my texts and offering suggestions for improvement. To you all my love and gratitude. Also, I would like to give my special thanks to my entire family and parents, Carlos and Marlene. They have always supported, inspired, and loved me. To all of them I dedicate this thesis.

I also thank the Defence Research and Development Canada (DRDC Valcartier), NSERC of Canada, and École de tecnologie supérieure for their financial support. This work would not have been possible without their support.

Last but not least, thanks to God for my life through all probations in the past five years. You have made my life more bountiful and given untiring help during my difficult moments. May your name be exalted, honored, and glorified.

DYNAMIC OPTIMIZATION OF CLASSIFICATION SYSTEMS FOR ADAPTIVE INCREMENTAL LEARNING

N. KAPP, Marcelo

ABSTRACT

An incremental learning system updates itself in response to incoming data without reexamining all the old data. Since classification systems capable of incrementally storing, filtering, and classifying data are economical, in terms of both space and time, which makes them immensely useful for industrial, military, and commercial purposes, interest in designing them is growing. However, the challenge with incremental learning is that classification tasks can no longer be seen as unvarying, since they can actually change with the evolution of the data. These changes in turn cause dynamic changes to occur in the classification system's parameters If such variations are neglected, the overall performance of these systems will be compromised in the future.

In this thesis, on the development of a system capable of incrementally accommodating new data and dynamically tracking new optimum system parameters for self-adaptation, we first address the optimum selection of classifiers over time. We propose a framework which combines the power of Swarm Intelligence Theory and the conventional grid-search method to progressively identify potential solutions for gradually updating training datasets. The key here is to consider the adjustment of classifier parameters as a dynamic optimization problem that depends on the data available. Specifically, it has been shown that, if the intention is to build efficient Support Vector Machine (SVM) classifiers from sources that provide data gradually and serially, then the best way to do this is to consider model selection as a dynamic process which can evolve and change over time. This means that a number of solutions are required, depending on the knowledge available about the problem and uncertainties in the data. We also investigate measures for evaluating and selecting classifier ensembles composed of SVM classifiers. The measures employed are based on two different theories (diversity and margin) commonly used to understand the success of ensembles. This study has given us valuable insights and helped us to establish confidence-based measures as a tool for the selection of classifier ensembles.

The main contribution of this thesis is a dynamic optimization approach that performs incremental learning in an adaptive fashion by tracking, evolving, and combining optimum hypotheses over time. The approach incorporates various theories, such as dynamic Particle Swarm Optimization, incremental Support Vector Machine classifiers, change detection, and dynamic ensemble selection based on classifier confidence levels. Experiments carried out on synthetic and real-world databases demonstrate that the proposed approach outperforms the classification methods often used in incremental learning scenarios.

OPTIMISATION DYNAMIQUE POUR L'APPRENTISSAGE INCRÉMENTAL ADAPTATIF DES SYSTÈMES DE CLASSIFICATION

N. KAPP, Marcelo

RÉSUMÉ

Lors de l'arrivée de nouvelles données, un système d'apprentissage incrémental se met à jour automatiquement sans réexaminer les anciennes données. Lors d'un apprentissage incrémental, les paramètres des systèmes de classification ne sont plus considérés comme invariants puisqu'ils peuvent évoluer en fonction des données entrantes. Ces changements causent des variations dans l'ajustement des paramètres du système de classification. Si ces variations sont négligées, la performance finale d'un tel système peut être ultérieurement compromise. De tels systèmes, adaptés au problème de classification, sont très utiles à des fins industrielles ou militaires car ceux-ci sont à la fois rapides d'exécution et peu gourmands en mémoire. On observe en conséquence un intérêt grandissant à l'élaboration de tels systèmes.

L'objectif principal de cette thèse est de développer un système capable de s'adapter de façon incrémentale à l'arrivée de nouvelles données, de suivre et d'analyser dynamiquement les paramètres du système optimal pour ainsi permettre son adaptation automatique à de nouvelles situations. Pour ce faire, nous commençons par aborder le problème de la sélection optimale des classificateurs en fonction du temps. Nous proposons une architecture qui combine la puissance de la théorie de l'intelligence des essaims avec la méthode plus conventionnelle de recherche par grilles.

Des solutions potentielles sont progressivement identifiées et mises en évidence pour des bases de données graduellement mises à jour. L'idée principale ici est de considérer l'ajustement des paramètres du classificateur comme un problème d'optimisation dynamique dépendant des données présentées au système de manière continue. En particulier, nous avons montré que si l'on cherchait à élaborer un classificateur SVM (Support Vector Machines) efficace à partir de sources de données différentes, graduelles ou en séries, mieux valait considérer le processus de sélection de modèles comme un processus dynamique qui peut évoluer et changer. Ainsi, les différentes solutions sont adaptées au fil du temps en fonction l'évolution des connaissances accessibles sur le problème de classifications et de l'incertitude sur les données.

Ensuite, nous étudions aussi des mesures pour l'évaluation et la sélection d'ensembles de classificateurs composés de SVMs. Les mesures employées sont basées sur les théories de la diversité et la marge communément utilisées pour expliquer la performance des ensembles de classificateurs. Cette étude révèle des informations précieuses pour l'élaboration de mesures de confiance pouvant servir pour la sélection des ensembles de classificateurs.

Finalement, la contribution majeure de cette thèse est une approche d'optimisation dynamique qui réalise un apprentissage incrémental et adaptatif en suivant, faisant évoluer et combinant

les hypothèses d'optima en fonction du temps. L'approche fait usage de concepts issus de différentes théories expérimentales, telles que l'optimisation dynamique de particules d'essaims, les classificateurs SVM incrémentaux, la détection de changement et la sélection dynamique d'ensembles à partir de niveaux de confiance des classificateurs. Des expériences menées sur des bases de données synthétiques et réelles montrent que l'approche proposée surpasse les autres méthodes de classification souvent utilisées dans des scénarios d'apprentissage incrémental.

CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

AIL	Adaptive Incremental Learning
AG	Adapted Grid-Search
BK	Best Solution Kept
CV	Cross-Validation
CPSO	Chained Particle Swarm Optimization
DFe	Data Feeding
DMS	Dynamic Model Selection
DPSO	Dynamic Particle Swarm Optimization
DT	Decision Tree
FPSO	Full Particle Swarm Optimization
GA	Genetic Algorithm
GS	Grid-Search
HS	Hyper-parameter selection
ISVM	Incremental Support Vector Machine
KNN	K Nearest Neighbors
LOO	Leave-One-Out
MLP	Multi-Layer Perceptron
MSS	Manipulating Sample Set Approaches
NB	Naive Bayes

NIST	National Institute of Standards and Technology
OAA	One Against All
OAO	One Against One
ORA	Optimized Random Aggregation
PKC	Preserving Karush-Kuhn-Tucker Approaches
PSO	Particle Swarm Optimization
RBF	Radial Basis Function
SMO	Sequential Minimal Optimization
SVM	Support Vector Machine
VC	Vapnik-Chernovenkis Dimension

LIST OF SYMBOLS

k	Time index, i.e. $k = 1, \ldots, \infty$
N	Dimension of feature space
$\mathbf{x}_i \in \Re^N$	Data sample in N-dimensional feature space
y	Class label
$(\mathbf{x}, y)$	Data sample with label associated
n	Number of training samples
$\mathcal{D}_k$	A particular training data chunk received at time k, i.e. $\mathcal{D}_k = (\mathbf{x}_i, y_i)_{i=1}^n$
$\mathcal{D}(k)$	A training dataset state at time k, i.e.: $\mathcal{D}(k) = \{\mathcal{D}_1, \cdots, \mathcal{D}_k\}_{k=1}^{\infty}$
$\mathcal{D}(k-1, k)$	Dataset transition state in interval of time
$\mathcal{G}$	Test set
$W(.)$	Training dataset size, e.g. $W(\mathcal{D}(k))$
w	Weight vector orthogonal to optimal hyperplane
b	Bias term
ξ_i	Slack variable associated to $\mathbf{x}_i$
α_i	Lagrangian multiplier associated to $\mathbf{x}_i$
l	Number of support vectors
$(\mathbf{x}_i.\mathbf{x}_j)$	Dot product
H	Eucledian hyperspace
Φ	Mapping function $\Phi : \Re^N \mapsto H$

$K(\cdot)$	A kernel function
C	Penalty parameter of the error term
γ	Radial Basis Function kernel parameter
r, u	Polynomial kernel parameters
ρ	Sigmoid kernel parameter
c	Number of classes in problem
Θ	Set of SVM hyper-parameters, e.g. $\Theta = \{C, \gamma\}$
$f(.)$	SVM decision function
$\mathcal{F}(\cdot)$	Objective function
P	Number of PSO particles employed to solve an optimization problem
$\mathcal{S}$	Set of solutions, i.e $\mathcal{S} = \{s_i, \dot{s}_i, s'_i\}_{i=1}^{P}$
q	DPSO current iteration index
s_i	Particle i's current position (i.e. a solution j)
$\dot{s}_i$	Particle i's velocity
s'_i	Particle i's individual best position
$s_{id}(k, q)$	Particle i's current position at dimension d, iteration q, time k
$\dot{s}_{id}(k, q)$	Particle i's velocity at dimension d, iteration q, time k
$s'_{id}(k, q)$	Particle i's individual best position at dimension d, iteration q, time k
$net(\cdot)$	Index of particle's best informant in its neighborhood.
$s'_{net(i,\lambda)}(k, q)$	Best solution from particle i's informant neighborhood, i.e. $net(\lambda)$, at time k, iteration q

$s^*(k,q)$	Best solution at time k, iteration q in $\mathcal{S}(k)$
$s^*(k)$	Best solution at time k in $\mathcal{S}(k)$
$s^*(k-1)$	Best previous solution found, i.e. at time $k-1$
$\mathcal{S}(k)$	Set of solutions at time k
$\mathcal{S}(k-1)$	Set of solutions at time $k-1$
β	Dynamic combiners methods' user pre-defined parameter
u	Weight assigned to the classifier.
χ	Clerc's constriction coefficient
ϕ	Clerc's constant
λ	Maximum number of connections in swarm communication topology
g	Index of best particle informant
$r1, r2$	Random values in [0,1]
$\Psi(\cdot)$	Training process
$\mathcal{M}$	SVM classifier, i.e. $\mathcal{M} = \Psi(s^*(k), \mathcal{D}(k))$
$\epsilon(\cdot)$	Generalization error estimation, e.g. $\epsilon((M), \mathcal{G})$
ν	Number of folds used in the cross-validation procedure
$\epsilon_\nu(\cdot)$	ν-Cross validation generalization error estimation, e.g. $\epsilon_\nu(s^*(\cdot), \mathcal{D}(k))$
δ_{max}	Max. difference between the objective functions values
σ	Variance between two errors by using a Normal approximation to the Binomial distribution
$z_{0.9}$	Standard value at the 90% confidence level

$EL(\mathbf{x})$	Expected loss for a sample $\mathbf{x}$
$B(\mathbf{x})$	Bias of an ensemble for a sample $\mathbf{x}$
$V(\mathbf{x})$	Variance of an ensemble for a sample $\mathbf{x}$
$V_n(\mathbf{x})$	Net variance of an ensemble for a sample $\mathbf{x}$
$V_u(\mathbf{x})$	Unbiased variance of an ensemble for a sample $\mathbf{x}$
$V_b(\mathbf{x})$	Biased variance of an ensemble for a sample $\mathbf{x}$
y_m	Main ensemble's prediction
y^*	Optimal prediction
$y_{\mathcal{M}_i}$	The prediction provided by the model $\mathcal{M}_i$
$\bar{d}$	Pairwise diversity measure average
$\bar{d}$	Average diversity
d_{ij}	Diversity between a pair of classifiers
Q	Q Average
DS	Disagreement measure
DF	Double-Fault measure
DF	Difficulty measure
KW	Kohavi-Wolpert (KW) variance
GD	Generalized diversity
A	Ambiguity
DY	Difficulty

CI	Margin-based measure
v_y	Number of votes for the true class
v_j	Number of votes for for any other class
$\tau(\mathbf{x}, y)$	Measure of margin computed for a sample $\mathbf{x}$ with label y
$\mu(\cdot)$	Mean function
sv	Set of support vectors of a classifier
rs	Set of relevant samples
Δ	Set of samples composed of support vectors and relevant samples
SV	Set of support vectors of all classifiers into an ensemble
L	Maximum number of ensemble members
$\mathcal{C}$	A classifier ensemble
$\mathcal{M}_i$	A classifier ensemble member
$\mathcal{C}^*$	Best ensemble selected

INTRODUCTION

Pattern classification systems have been devised for many applications and in many fields in the past. Intended for different purposes but sharing the same principles, these systems are designed to teach computers to solve problems based on past experiences. To build a pattern classification system, a considerable amount of data is processed and compared with patterns already stored in memory. In the last four decades, remarkable advances have been made in a number of recognition fields, e.g. recognition, speech, handwriting, etc. In fact, nowadays, if sufficient data are provided, it is possible to make an almost perfect classifier for any pattern classification problem.

However, despite the advances, most of these systems have been built using real-world data that are considered to be stationary. In other words, their development is based on the assumption that the available training data are always adequate, representative, and available in sufficient quantity. Consequently, once the classification system has been trained in a laboratory phase, the assumption is that it will be capable of classifying new, future instances indefinitely in the real world, i.e. in its operational phase. However, the incompleteness of training data is a common problem when developing many real-world applications. For instance, in face recognition applications, due to the large variation in facial expressions, lighting conditions, makeup, and hairstyles, it is very difficult to collect data on all the possibilities in advance. Likewise, there are unlimited ways of writing and speaking when developing handwriting or speech recognition systems. Thus, even with the knowledge that the performances of classification systems are highly dependent on data, to wait until the entire acquisition and storage process has been complete would be impractical, uneconomical, or even impossible. An alternative would be to implement systems capable of learning incrementally.

Incremental learning systems update trained models in response to incoming data during their operational phase, without reexamining all the old data. As a result, they are economical, in terms of both space and time, which makes them immensely useful for industrial, military, and commercial purposes. Because of this, interest in designing classification systems capable

of incrementally storing, filtering, and classifying data is growing. At the same time, there is a challenge with incremental learning, which is that classification tasks can be no longer seen as unvarying, since they can actually change according to the evolution of data. These changes make the adjustment of a classification system's parameters a dynamic process. If such variations are neglected, the overall performance of these systems will be compromised in the future, resulting in the defeat of even the most successful conventional machine learning techniques, because they are not capable of adapting.

In light of this, a classification system must be able to incrementally accommodate new data and dynamically adapt itself in order to better maintain its optimality with respect to internal parameters, computational cost, and generalization performance. This brings us to the central topic of this thesis, which is to contribute, with new solutions and breakthroughs, to the implementation of an adaptive incremental system based on dynamic optimization techniques. In particular, experiments are carried out using Support Vector Machine (SVM) classifiers as base classifiers, and synthetic and real-world databases involving different types of applications, such as: handwritten digits, multisensor remote-sensing images, forward-looking infrared ship images, etc. Therefore, databases with different numbers of classes, features, and training samples are used when testing approaches with different learning strategies (i.e. gradual and incremental) in a supervised learning context.

Problem Statement

A fundamental problem with incremental learning in static environments is that the best set of a classification system's parameters can vary over time, owing to changes in the incoming data. Such changes can, for example, be minor fluctuations (random or systematic [70]) in the underlying probability distributions. These usually result from either sample shifting or the natural evolution of classification problems, considering that new knowledge comes in part from new observations at different times. Therefore, the sample distributions of training data chunks may change and affect the system in several ways, since its decision boundaries are estimated according to those distributions. In the literature, these possible data changes are

defined as *population drifts* [58, 109]. The problem in incremental learning scenarios is that they are unavoidable, even though the application environment seems to be static (i.e. where the numbers of classes, features, etc. remain constant).

Consequently, the incremental updating of a classification system might require not only reviewing its existing models in terms of knowledge acquired and new data, but also in terms of its internal parameters set with respect to such data variations. Otherwise, the whole system may become obsolete and so fail to achieve a better adaptation in the future. This assumption might explain why, even though significant research has been conducted to design incremental learners [13, 109, 93, 26, 88], the results are not often as satisfactory as those for batch mode learners (i.e. when all data are considered). Taking this into account, we propose to optimize the traditional incremental learning approaches that consider the adjustment of parameters as a static process (i.e. constant parameter values are employed infinitely) over time, to increase the system's power of generalization and decrease its complexity.

In addition, as we use the SVM classifier here, because of its robustness against the well known *curse of dimensionality* [38], the task of searching for optimum hyper parameter values is a primary problem that must be faced, the so-called SVM model selection problem. Solving this problem is important because, although SVMs are very powerful classifiers in theory, their efficiency in practice relies on the optimal selection of hyper parameters. This is because a naïve or *ad hoc* choice of values for its hyper parameters can lead to poor performance in terms of generalization error, as well as high complexity in terms of the number of support vectors identified. In recent years, many model selection approaches have been proposed in the literature. They differ basically in two aspects: (1) the selection criterion; and (2) the searching methods used. The selection criterion, i.e. the objective function, is a measure that guides the search. Some of these criteria are specifically related to the SVM formulation, such as radius margin bound [118], span bound [19], and support vector count [117]. Others are classical, such as the well-known cross validation and hold-out estimations. The most common searching methods applied are the gradient descent techniques [27, 20, 3], the grid-search techniques [18, 47, 49], and the evolutionary techniques, such as genetic algorithms

(GA) [22, 25, 107, 21], the covariance matrix adaptation evolution strategy (CMA-ES) [40], and, more recently, Particle Swarm Optimization (PSO) [29, 52].

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. Similarly, multiple local minima in objective functions are a nightmare for gradient descent-based methods. To overcome this, the application of grid-search or evolutionary techniques is a very attractive option. Unfortunately, in the case of the grid-search method, a good discretization of the search space in fixed values is crucial for achieving high performances. So, the main challenges in the SVM model selection research field are considered to be: (1) the choice of objective function, (2) the presence of local minima in the search space, and (3) the computational time required for model selection task. In addition to these typical parameter estimation difficulties, the estimation of parameters over time from incoming data at different times aggravates the model selection problem. This is because, when knowledge of the problem is limited, or the data are noisy or arrive in batches over time, the model selection task and its performance can progressively degrade. So, we consider a gradual learning scenario (i.e. when historical data are not discarded) in order to study the dynamism of the parameter search space with respect to different levels of uncertainty.

An interesting alternative for improving the performance of single classifiers is the fusion of classifier decisions into ensembles, especially when the level of uncertainty is high, i.e. when only small sample sets are available [116]. However, despite all these efforts, our understanding of the effectiveness of the ensemble methods is still lacking, and is driving new research on classifier fusion. As a result, several works on ensembles of classifiers (EoC) have been conducted to find measures that could be well correlated with ensemble accuracy and so used to evaluate and select the best classifier ensembles [67, 99, 36, 28, 125, 96, 69, 73, 122, 9, 116, 110]. Nevertheless, there is a consensus in the literature indicating that some diversity exists between ensemble members, and that this diversity is the main source of possible improve-

ment in overall performance [28, 69, 73, 122, 9]. Although it is well accepted that diversity is, as far as we know, a necessary condition for improving overall accuracy, there is no general agreement on how to quantify it or deal with it. Thus, even though the application of EoC is clearly advantageous, the search for an efficient objective function for selecting the best ensemble from a pool of classifiers is still a persistent problem. This is a particularly important issue with respect to the development of an incremental learning system, as considered in this thesis.

Research Goals and Contributions

In our effort to implement an adaptive classification system, we accomplish three major goals. The first is to develop a method for searching for optimum values for SVM hyper parameters over time. We face two main challenges in this endeavor: (1) overcoming common difficulties involving optimization processes, such as the presence of multimodality or discontinuities in the parameter search space, and (2) quickly identifying optimum solutions that fit both historical data and new, incoming data. If we do not meet these challenges, the processes for searching hyper parameters over sequences of datasets could perform poorly or be very time-consuming.

To tackle these two issues, we first study the SVM model selection task as a dynamic optimization problem considering a gradual learning context in which the system can be tested with respect to different levels of uncertainty. In particular, we introduce a Particle Swarm Optimization-based framework which combines the power of Swarm Intelligence Theory with the conventional grid-search method to progressively identify and evaluate potential solutions for gradually updated training datasets. The key idea is to obtain optimal solutions via re-evaluations of previous solutions (adapted grid-search) or via new dynamic re-optimization processes (dynamic Particle Swarm Optimization, or DPSO). Experimental results demonstrate that the proposed method outperforms the traditional approaches, while saving considerable computational time. This framework was presented in [57, 55].

The second goal is to experimentally investigate several objective functions for the evaluation and selection of EoC. This is an important step in improving the applicability of SVM en-

sembles in the classification system proposed here. In this study, we analyze classifier fusion empirically through the relationship between two theories related to an ensemble's success, i.e. diversity measures and margin theory, with ensemble accuracy. In order to achieve this, we first survey some classical diversity measures and some measures related to margin theory. Then, an experimental protocol similar to that introduced in [116] for characterizing SVM ensembles is employed to evaluate the measures and draw results. Then, from a discussion on those results, we try to answer some questions currently arising from the literature, such as the following: Which measure offers the best guidance in classifier fusion evaluation? How are the diversity measures related to each other? Is there a relationship among diversity, margins, and ensemble accuracy? What are the best measures for observing such a relationship? Finally, we conclude this study with valuable insights on methods for fusion evaluation and selection of EoC. These investigations are very important, since it has been demonstrated in the literature that the fusion of classifier decisions into ensembles can actually improve the performance of single classifiers, even SVMs [116]. However, despite these efforts, our understanding of the effectiveness of ensemble methods continues to perplex, and this is driving new research on classifier fusion [67, 99, 36, 28, 125, 96, 69, 73, 122, 9, 116, 110]. Most importantly, this study provides valuable insights on how these two theories can influence each other and shows us how confidence-based measures can be of greater interest than diversity measures for the selection of EoC. A study of this nature was presented in [54].

The final goal is to propose a classification system that performs adaptive incremental learning. The method is implemented based on the following two principles: (1) the incremental accommodation of new data by updating models, and (2) the dynamic tracking of new, optimum system parameters for self-adaptation. Our aim is to overcome a problem that arises with incremental learning, which is the obsolescence of the best set of classification system parameters as a result of incoming data. In particular, the proposed method relies on a new framework incorporating various techniques, such as single incremental SVM (ISVM) classifiers, change detection, DPSO, and, finally, dynamic selection of EoC. The goal of our method is to update, evolve, and combine multiple heterogeneous hypotheses (i.e. models with different parame-

ters and knowledge) over time, and hence to maintain the system's optimality with respect to internal parameters, computational cost, and generalization performance. As a result, adaptations are realized in two levels, beyond what is achieved by the incremental learning aspect alone and into the levels of base mode parameters and decision fusion. Thus, unlike the traditional incremental learning approaches, which consider classifier parameter adjustment as a static process (i.e. constant parameter values are employed to update the system infinitely), we are suggesting that they be optimized over time to increase their power of generalization and decrease their complexity. In order to achieve this, our underlying hypothesis, set out here, is to consider the incremental learning process as a dynamic optimization process, in which optimum hypotheses are dynamically tracked, evolved, and combined over time.

The proposed method is validated and demonstrates 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: SVM optimized with PSO in batch mode, ISVM with parameter values fixed beforehand, and two increment-capable classifiers (1-NN and Naïve Bayes), which are widely applied in incremental learning studies. The performances of these classifiers are considered "no less" than those of their batch versions [87]. An incremental ensemble strategy with optimized parameters and different combination rules is also employed for comparison. As additional objectives of this study, we try to verify whether or not: (1) incremental learning with SVM can achieve similar performances to those obtained in batch mode; (2) adaptation of the system's parameters over time is actually a dynamic optimization problem, and, if so, it is important to achieve high performances; and (3) the dynamic selection of EoC can lead to better results than simply combining all the pools of classifiers available. We introduce this method and results in [56].

The additional contributions of this work are to provide insights on strategies for optimizing and selecting classifiers, on the use of memory-based mechanisms, and on dynamic optimization methods.

Organization of the thesis

The thesis consists of four chapters. Chapter 1 and chapter 2 present a brief literature review of the main research topics and works related to the development of classification systems capable of performing incremental learning. The notion of data changes is also described. Then, general approaches and classifiers that have been proposed to build classification systems capable of learning incrementally are surveyed. The research directives adopted in this thesis are also discussed.

In chapter 3, we empirically demonstrate that the SVM model selection problem performed over time can, in fact, be treated as a dynamic optimization problem. Based on this assumption, a PSO-based framework, which combines the power of Swarm Intelligence Theory with the conventional grid-search method is introduced. Experimental results with this method and with traditional approaches are presented.

In chapter 4, we investigate nine measures from two different theories (diversity measures and margin theory) to be employed in the evaluation and selection of SVM ensembles. From empirical results, discussions on how these two theories can influence each other and on the application of margin-based measures are described.

In chapter 5, the proposed adaptive incremental learning method is presented. We describe each additional module composing the framework, and explain the various strategies for adaptation and performance improvement, such as dynamic parameter optimization and the selection of ensembles based on their respective confidence levels . Experiments and results obtained are reported. Finally, we outline our conclusions and suggest guidelines for future work.

CHAPTER 1

PATTERN CLASSIFICATION IN IMPRECISE ENVIRONMENTS

The development of classification systems capable of performing adaptive incremental learning requires an understanding of the challenges inherent to classification in imprecise environments, i.e. environments where the uncertainty level in the incoming data is usually high and where different types of data change can be involved. In this chapter, the two main concepts regarding pattern classification in such environments are introduced: (1) the capability of incremental learning; and (2) the various changes that can occur in the data.

1.1 Incremental Learning Definition

Incremental learning means learning new data over time without keeping all the old data for subsequent processing, thereby reducing training time and computational effort. However, an incremental learner should be able to adapt to new information without corrupting or forgetting previously learned information. In other words, it must deal with the so-called stability-plasticity dilemma, which describes the state where a stable classifier will preserve existing knowledge, but will not accommodate new information, while a completely plastic classifier will learn new information, but will not conserve prior knowledge [93].

Incremental learning approaches are very attractive for solving several real world classification problems, especially those where: (1) the data acquisition process is expensive, and so only a few samples become available over time; (2) the data generation process is itself time-dependent, as in time series data; or (3) the training data available are too large to be loaded into computer memory [109]. Basically, in agreement with Polikar et al. [93], an incremental learning algorithm must meet the following criteria:

a. It should be able to learn additional information from new data;

b. It should not require access to the original data used to train the existing classifier;

c. It should preserve previously acquired knowledge, i.e. it should not suffer from catastrophic forgetting;

d. It should be able to accommodate new classes that may be introduced with new data.

It is important to note that incremental learning as referred to here is a process of updating a classification system with suitably sized samples of datasets at a time, i.e. block by block, and not one sample at a time, which is called online learning or instance-by-instance learning [109]. In the literature, a generic algorithm for incremental learning may be defined in five steps [77]:

1) Learn rules from examples;

2) Store rules, discard examples;

3) Use rules to predict, navigate, etc.;

4) When new examples arrive, learn new rules using old rules and new instances;

5) Go to step 2.

To summarize, the general idea behind incremental learning is that the knowledge base is increased incrementally as each new piece of information is obtained. For this reason, classification systems with incremental learning capabilities can more accurately represent the manner in which humans learn.

Unfortunately, as new small pieces of information arrive at different times during incremental learning, the whole learning/classification process can suffer disturbances, depending on the changes occurring in the data. We explain the possible changes to the data in the next section.

1.2 Concept Drift Issues

In many real world problems, a huge quantity of new information is created dynamically moment by moment; for example, applications involving data streaming: spam filtering, financial prediction, credit card fraud protection, network intrusion and surveillance video streams, stock market trend analysis, etc. Most of the time, these data must be stored, filtered, or organized in some way. Such tasks demand powerful computers and systems capable of dealing with huge volumes of data and data distributions that may change over time.

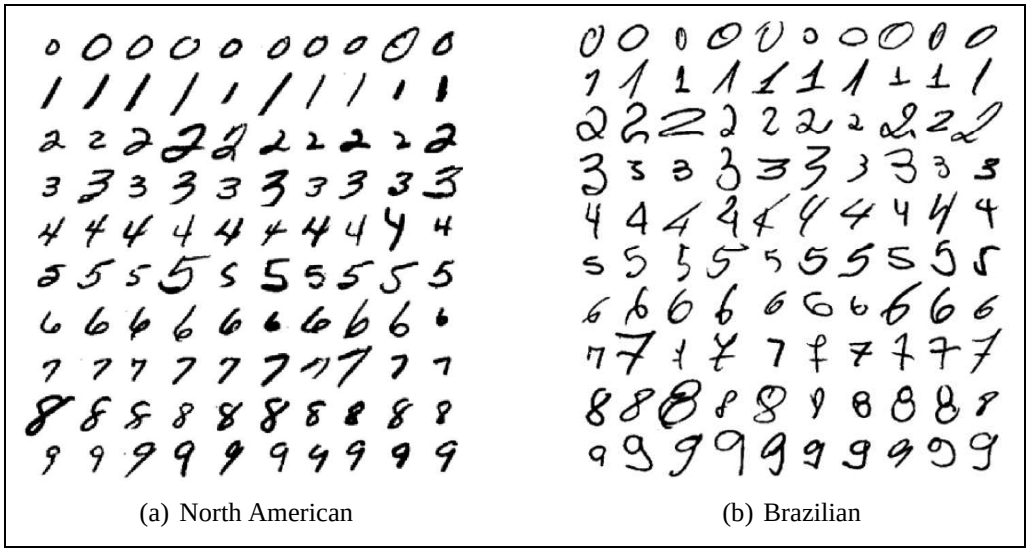
A persistent challenge with incremental updating is that possible variations in problem data distributions can affect system performance. In the literature, these changes in the data are called *concept drifts*, specifically *population drifts* or *real drifts*, depending on the type of change. We explain these changes below:

- *Real drifts*: Real drifts refer to changes in the target concepts (e.g. class labels) [58]. This kind of data drifting occurs for a category of real-world problems. For example, in object tracking or user-interest-guided applications, the class of interest varies over time. This means that, in order to efficiently predict data, the system might incrementally learn data about the current concept, and, at the same time, remove old, conflicting concepts. Thus, in real-drift situations, the incremental learning process must cope with population drifts resulting from updating phases, and also with changes inherent to the nature of the problem, which can sometimes even invalidate the knowledge already acquired by the system.

- *Population drifts*: Population drifts refer to hidden changes in the underlying data distributions intrinsically related to the incremental learning process. This is because they result, for example, from sampling shifting, which depends on the order and the representativity of samples present in the incoming data. In such cases, the concepts (classes) are usually predefined, but their distributions can evolve when new data arrive. For example, the frequency of new types of spam mails and their features may change drastically over time, which causes variations in the data distributions and decision boundaries that

distinguish whether or not a message is spam [31]. This means that population drifts are unavoidable in the incremental learning process, even when the application environment seems to be static or when real drifts are involved.

Several examples involving real-world applications can be provided to better illustrate population drifts. For example, handwriting recognition systems are usually trained from a fairly large amount of data. Nevertheless, there are unlimited ways of writing a character, and it would be impractical, if not impossible, to collect and store every possibility. In this connection, a problem arises when systems implemented from specific user styles are exposed to other styles, e.g. different populations and regions. The systems would certainly not achieve the same success for both styles. To illustrate, in Figure 1.1, we show some isolated digits handwritten in the North American and Brazilian styles. It is easy to see that variations in the two styles, e.g. for the numbers 1, 2, 7, etc., could be reflected in changes to the data distribution classes, which would require updating of the system in order to prevent compromising future classifications.



(a) North American (b) Brazilian

Figure 1.1 Examples of variations between handwritting styles. 1.1(a) Handwritten digits from North American (NIST SD-19 database) and 1.1(b) handwritten digits from a Brazilian database of checks [84].

In probabilistic terms, the changes that may occur in a classification problem are related to [58, 70]:

- Prior probabilities for the c classes, $P(\omega_1), \ldots, P(\omega_c)$;
- Class-conditional probability distributions, $p(x|\omega_i)$,, $i = 1, \ldots, c$; or
- Posterior probabilities $P(\omega_i|x)$, $i = 1, \ldots, c$.

Population drifts result from changes that occur in the $P(\omega_i)$ and $p(x|\omega_i)$ of classes, while real drifts are related to changes in the $P(\omega_i|x)$ of classes. Figure 1.2 depicts these types of concept drifts in probabilistic terms. Consider the class densities for two classes: ω_1 and ω_2, and the optimal decision boundary of separation regarding one input variable x, as illustrated in Figure 1.2(a). The effect on the decision boundary of the various kinds of drifts mentioned previously are subsequently depicted in Figures 1.2(b), 1.2(c), and 1.2(d). Figure 1.2(b) illustrates a drift caused by the priors. Then, Figure 1.2(c) depicts a drifting in class density resulting from a sampling shift for the class ω_1 between the x values 0.65 and 0.8, for example. Finally, a drifting in the posterior probability of the class ω_1, and also between the x values 0.65 and 0.8, is shown in Figure 1.2(d).

Now that the definition of incremental learning has been presented and the difficulties it commonly encounters explained, in the next section we survey the main approaches introduced in the literature to deal with these data changes and for incremental learning to occur.

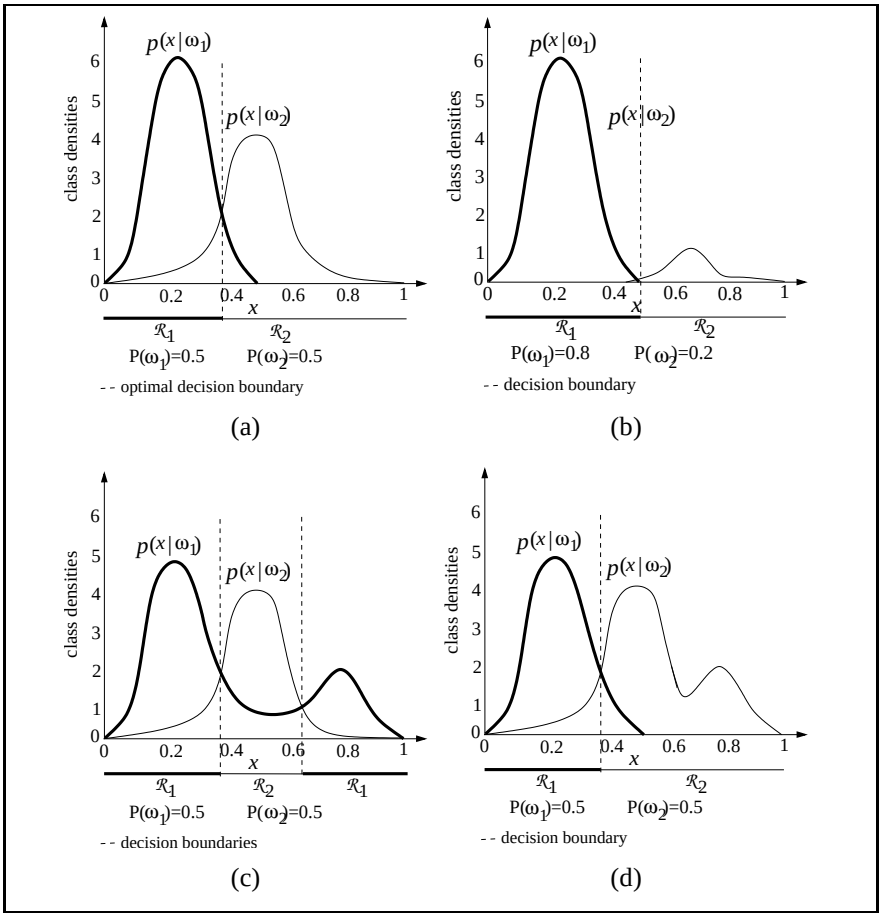


Figure 1.2 Illustrations of different kinds of data drifting. (a) Initial. Data drifting in (b) Priors, (c) Class densities, (d) Posterior probability.

CHAPTER 2

RELATED APPROACHES

In the previous chapter, the definition of incremental learning was presented, along with the challenges involved in keeping a classification system up to date. In addition, we have seen that population drift is a common difficulty which needs to be faced by incremental learning processes with the presence, or not, of real drifts. Taking this into account and for the sake of clarity, a literature review is provided in this chapter on the main approaches and techniques that have been employed for dealing with these situations.

First, the main approaches applied in this research area are surveyed: (1) instance selection, (2) instance weighting, (3) incremental classifiers, and (4) ensemble of classifiers. We start by giving a general overview of these approaches, as illustrated in Figure 2.1. They are then summarized, with reference to their respective related works. Finally, we present a discussion on the research directives adopted in our thesis for the implementation of an adaptive incremental learning method, which is the research domain at issue here.

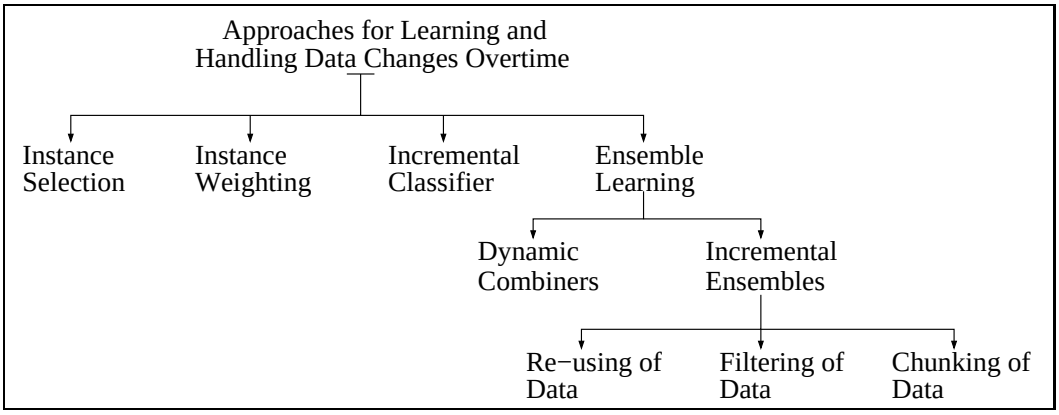


Figure 2.1 General overview of techniques for developing adaptive classification systems.

2.1 Instance Selection

Instance selection-based methods handle data changes, especially those related to real drifts, by learning from a “time-window" of relevant examples. The relevance of the examples is usually measured by their “age" or correctness, depending on the strategy. A generic example of this method is depicted in Figure 2.2. Consider an existing learner and a selection criterion; the instance selector carefully selects relevant examples from the data stream and stores them in a *time-window*. When the number of examples in that window reaches a maximum of $\mathcal{T}$ items, the learner is entirely recreated or incrementally updated from the window. Older examples, or examples that are no longer relevant, are discarded.

Related works we can cite are those of Schlimmer and Granger [100], Widmer and Kubat [121], Maloof and Michalski [78], and Lazarescu et al. [75]. Schlimmer and Granger [100] introduced the STAGGER system, which maintains a set of concept descriptions (sets of symbols numerically weighted by Bayesian weighting measures). When the system fails to predict a membership class for a new instance, a new, more complex concept description is built by the iterative use of feature construction, where the most relevant concept is selected.

In [121], Widmer and Kubat introduced the FLORA algorithm, which learns current concepts by implementing a rule system from a window of recent examples. The algorithm learns new instances incrementally, while “forgetting" the oldest ones. Algorithm variants (FLORA 2, 3, and 4) have been also been implemented with different characteristics, such as: the use of an adaptive window size, a store of “stable" concepts, etc. In this same vein, Maloof and Michalski [78] have introduced a partial memory system, called AQ-PM, which tests training instances and selects only misclassified examples to store in the window for future learning phases. A user-defined threshold controls a forgetting mechanism. Klinkenberg and Joachims [64] suggest dynamically adjusting the window size by monitoring the system’s performance on the last chunk of data. The authors train the Support Vector Machine (SVM) classifier with different window sizes from previous data and select the window size that maximizes the accuracy on the last chunk of data. Another variation of the original instance selection

approach has been introduced by Lazarescu et al. [75], which uses an unsupervised algorithm and not one, but three multiple competing windows of different sizes to give the method more flexibility.

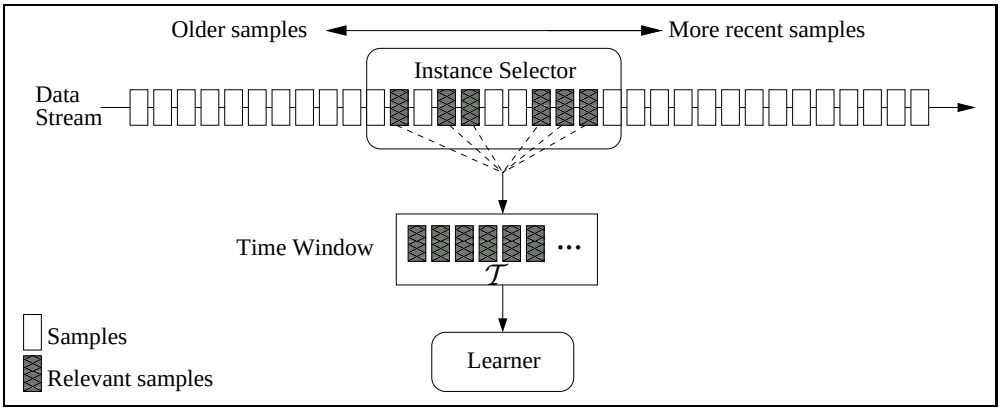


Figure 2.2 Illustration of the instance selection approach.

2.2 Instance Weighting

Instance weighting methods assign weights to instances according to their age and/or their influence in the current concept. Unlike instance selection methods, where examples are considered equally relevant in the window, these methods try to create different degrees of relevance for each example by computing weights for all instances, even the relevant ones. This approach has not often been applied, probably because it calls for learning algorithms capable of processing weighted instances. For example, in [62, 65], the authors implemented instance weighting by employing an SVM classifier. Furthermore, they have a tendency to overfit the data, as observed in [63].

2.3 Incremental Classifier

The incremental classifier approach refers to incremental model maintenance. In other words, approaches in this group employ a classifier algorithm capable of being continually updated. The incremental learning process with a single classifier can be summarized as illustrated in Figure 2.3. Let $\mathcal{D}(1), \mathcal{D}(2), \ldots, \mathcal{D}(n)$ be datasets available to the learning algorithm at instants

$k = 1, 2, \ldots, n$. The learning algorithm starts with an initial classifier (hypothesis) $\mathcal{M}(1)$ trained from $\mathcal{D}(1)$. Then, $\mathcal{M}(1)$ is updated to $\mathcal{M}(2)$ on the basis of $\mathcal{D}(2)$, and $\mathcal{M}(2)$ is updated to $\mathcal{M}(3)$ on the basis of $\mathcal{D}(3)$, and so on for future iterations.

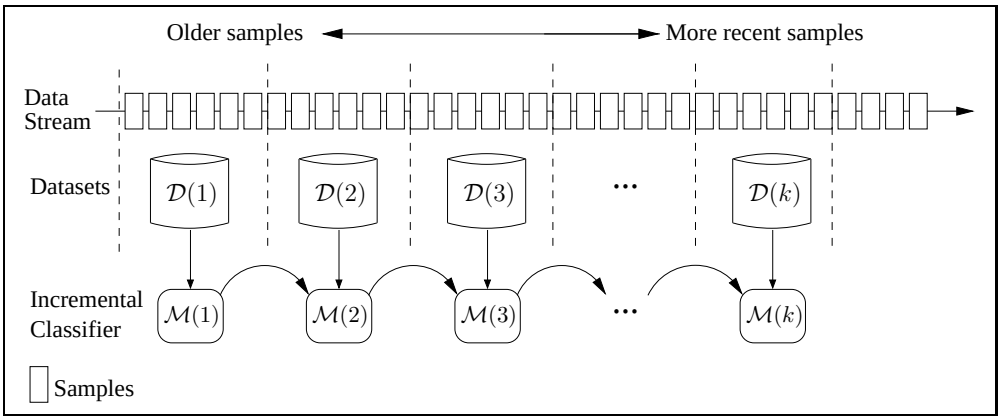


Figure 2.3 Illustration of the incremental learning process.

Below, we summarize some single classifiers capable of incremental learning:

- Learning vector quantization (LVQ): a simple and successful online learning algorithm also originating from the neural network literature [70];
- Naive Bayes Classifier: a very suitable method for updating an existing classifier, since the sample frequencies required for calculating the prior probabilities can simply be increased as new examples arrive;
- Nearest Neighbor: a classifier that is both intuitive and accurate. The training set can be built by storing each labeled sample **x** as it arrives;
- Neural Networks: while generally suffering from catastrophic forgetting [93], a particular neural network family is capable of incremental learning: ARTMAPs [13, 16, 15];
- Decision Tree: a classical batch classifier that has been modified to accommodate new data over time. An interesting version can be found in [50];

- Support Vector Machine (SVM): a classifier that tries to find a good representation of the boundary between classes. It has the advantage of being readily suited to incremental learning tasks. More details about incremental versions of the SVM are provided in the next section.

2.4 Ensemble Learning

An ensemble is a set of classifiers (e.g. decision trees (DT), artificial networks neural (ANN), support vector machines (SVM), etc.) organized in such a way that their individual decisions are combined to obtain the ensemble prediction when a new example is to be classified. The goal is to combine different classifier decisions to decrease the variance and the error among single solutions obtained by training from a dataset $\mathcal{D}$.

Ensembles of Classifiers (EoC) have become very popular, as they often outperform single models. Consequently, the literature on EoC has grown extensively with the objective of understanding them better and improving their results [67, 99, 36, 28, 125, 96, 69, 73, 122, 9, 116, 110, 54]. Because of this interest, EoC are now widely applied in diverse pattern recognition applications.

The construction of an EoC involves the design of classifier members and choosing a fusion function to combine their decisions. Classifier members can be designed in different ways [33], such as the following:

- *Manipulating training examples*: These methods vary the training samples in order to generate different datasets for training the ensemble members. Some examples are: Bagging (*B*ootstrap *Agg*regat*ing* [5]) and Boosting ([98]).

- *Manipulating input features*: Methods in this group manipulate the features to obtain diversity among members. Their goal is to provide a partial view of the training dataset to each ensemble member, so that they become different from one another. Examples are the Random Subspace method [46] and the feature subset selection strategies [111].

- *Manipulating output targets*: In this strategy, the labels of training samples are manipulated to produce different classifiers. For instance, using the Error Correcting Code method [34]), a multi-class problem is transformed into a set of binary problems. At each new iteration, a different binary division of the training dataset is used to train a new classifier.

Along with these three categories, there are also some methods that manipulate ensemble members (i.e. producing heterogeneous ensembles, the members of which can actually be different classifiers [97] or represent variations of some aspects of a given classifier, such as the topology for neural networks [102] or hyperparameters for SVMs [115], etc).

At the same time, the choice of fusion function depends on what kind of information is obtained (e.g. labels, probability estimation, etc.) from the individual models. Among the most common options found in the literature are: majority voting, simple average, sum, product, maximum, minimum, weighted average, Naive-Bayes combination, Decision Templates (DT), etc. [61, 72, 71]. For more information about combination functions and classical methods for the creation of ensembles, a comprehensive survey with examples can be found in [71].

In addition, different combination architectures can be defined, according to classifier arrangement. There are several related topologies or structures in the literature, such as conditional, serial, parallel, etc. Lam [74] proposed a classification of these topologies as follows:

- *Conditional*: This topology is based on confidence level, and it works in two ways. A base structure is used to measure the confidence level. If there is a rejection, or if the classification is made with a low level of confidence, a secondary structure is used which is more specialized in the particular problem. This secondary structure, which is usually more complex than the first, is only used for more difficult patterns.

- *Serial*: The classifiers are arranged in series. Each classifier produces a reduced set of possible classes or values that are used by posterior classifiers.

- *Parallel*: This topology consists of a set of classifiers consulted in parallel. First, an EoC operates in parallel to produce classifications of a pattern, and then their decisions are combined by a fusion function.

- *Multistage*: In this topology, the classifiers are arranged in different stages, such as hybrid combinations of parallel-serial or serial-parallel architectures.

In the same way, inspired by the success of the conventional EoC methods introduced above, which are traditionally applied over a single dataset, similar techniques have been proposed to perform incremental learning. Based on the original idea, EoC methods for incremental learning also generate and combine sets of classifiers. However, the creation of base classifiers is slightly different, i.e. rather than fixed datasets, now new datachunks can arrive over time.

A comprehensive survey on the various ensemble techniques for dynamic environments is presented in Kuncheva [70]. Based on that study and [31], the next two sections present a compilation of the proposed strategies found in the literature in two groups: dynamic combiners and incremental ensemble approaches.

2.4.1 Dynamic Combiners

Dynamic combiners train ensemble members in advance and then changes (i.e. concept drifts) are tracked by updating the combination rule with respect to new data. Therefore, the adaptation is performed only at the decision fusion level, since existing classifiers are never retrained. Methods in this group are commonly called "horse racing" algorithms. We outline some of these algorithms below:

a. *Weighted Majority*: This algorithm is composed of four steps [70]: 1 - Initialize all weights $\{u_j\}_{j=1}^{L} = 1$, assigning to each base classifier a classifier ensemble $\mathcal{C}$. 2 - For each new training sample $\mathbf{x}$, compute the support for each class as the sum of the weights of all classifiers that suggest its respective class labels for $\mathbf{x}$. Label as $\mathbf{x}$ the class with the largest support. 3 - Check the true label of $\mathbf{x}$ and update the weights of all experts

with an incorrect prediction as $u_i = \beta u_i$, where $\beta \in [0, 1]$ is a user predefined parameter. 4 - Continue from Step 2.

b. *Hedge* β: The same updating rule as in the Weighted Majority algorithm is employed. However, instead of taking decisions based on the weighted majority, this method uses the prediction from a selected classifier as the ensemble decision. The selection process is based on a probability distribution defined by the normalized weights.

c. *Winnow*: This method is similar to the Weighted Majority algorithm, but has a different updating rule. In this algorithm, the weights are recomputed only if the ensemble provides an incorrect prediction for the current input $\mathbf{x}$. In addition, the weight u of each classifier is updated, as follows: If the correct label for $\mathbf{x}$ is obtained by a given classifier, its weight is increased, becoming $u_j = \beta u_j$ (*promotion step*), otherwise it is decreased, becoming $u_j = u_j/\beta$ (*demotion step*). In this way, base classifiers are promoted or punished according to the ensemble errors.

d. *Mixture of Experts*: Unlike the previous dynamic combiner methods, this strategy represents a special case in which the fusion decision rule and a selected classifier are updated from each new example. Therefore, it is important to note that the base classifiers must support incremental learning.

Although the dynamic combiner methods are attractive from an implementation point of view, the main problem with such an approach is their failure to adapt to new data at the base classifier level, since they must be trained in advance. This is a disadvantage, because it may compromise the performance of the whole system when exposed to an environment where no adequate classifier has been previously trained.

2.4.2 Incremental Ensemble

Unlike dynamic combiners, the Incremental Ensemble methods are flexible, since they consider the updating of ensemble size, member knowledge, and a combination rule. The key here is

how to assign the data to subsets in order to train the base classifiers. This decision also determines how new examples are learned by the ensemble. Basically, they can be categorized into three groups [70]:

a. ***Updated training data***: The methods in this group use fresh data to make online updates of the ensemble members, where the combination rule may or may not change.

   - *Reusing data points*: As described by Oza [86], an online bagging algorithm is used to converge to batch bagging as the number of training examples and the number of classifiers tend to infinity. The training samples for the classifiers in the ensemble are created incrementally. The base classifiers are trained using online classifier models.

   - *Filtering*: Training sets are formed for the consecutive classifiers as the data flows through the system. The basic idea is to build the ensemble members progressively using portions of a training set. Examples of this kind of approach are variants of the traditional Boosting method [98], e.g. in [82] or the Pasting-small-votes [7] method.

   - *Using data blocks or chunks*: The ensemble is updated using batch mode training on a "chunk" of data. That chunk can be treated as a single item of data, because the ensemble is trained on the most recent block, on a set of past blocks, or on the whole set of blocks.

b. ***Updating ensemble members***: The classifiers in the incremental ensemble can be updated online or retrained in batch mode when blocks of data are available.

c. ***Structural changes to the ensemble*** This strategy creates an individual classifier from each new data chunk available. Then, whenever a change in the environment is detected, they are re-evaluated and the worst or oldest classifier is replaced by a new classifier trained on the most recent data. A general overview of this approach is depicted in

Figure 2.4. The idea here is to divide data streams into chunks of data for learning. For each chunk of data, a new base classifier is trained and combined with preceding ones for future predictions. This scheme is also called "block evolution" in [41]. The decision's fusion of classifiers is usually realized by weighted voting, where the weights are computed from the most recent data. Figure 2.4 depicts the key idea and the variants that can be suggested naturally based on this idea.

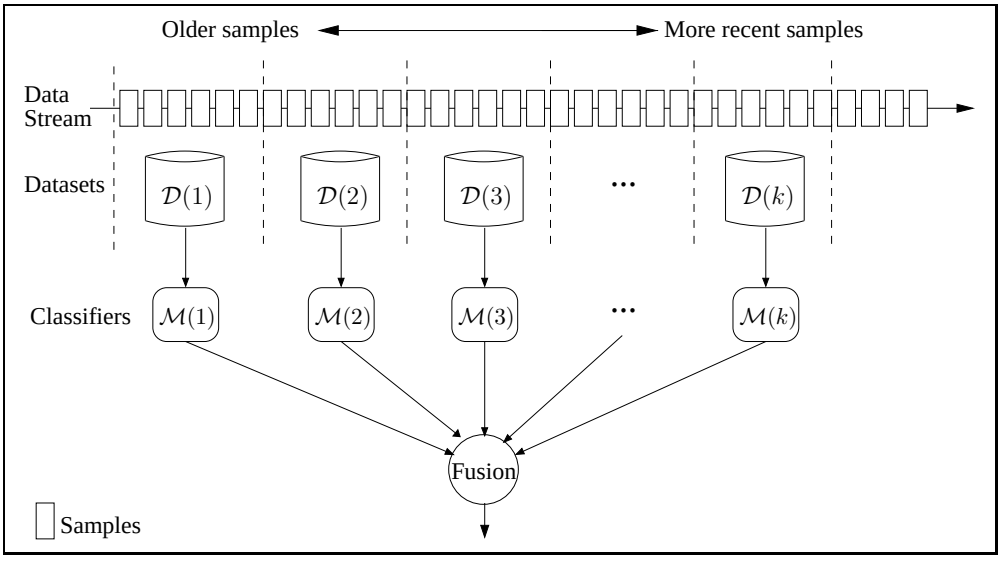


Figure 2.4 Illustration of the incremental learning process based on ensemble learning.

Despite this categorization of the approaches, it is important to note that combinations between them are possible. In order to provide a better overview of the approaches and related works that we have cited in this chapter, we summarize this information in Table 2.1.

In this table, we can see the base classifier, the type of data change studied when learning over time, and the approach employed. We provide more details on these aspects and works in section 2.7. We now turn our focus to the base classifier employed in this paper, which is described in the next section.

Table 2.1 Compilation of some related works reported in the literature by outlining the base classifiers, the type of concept drift involved, and approach adopted.

Related works	Base classifier	Drift Type	Approach applied
Schlimmer and Granger [100]	Rule-based learning	Real	Instance selection
Widmer and Kubat [121]	Rule-based learning	Real	Instance selection
Maloof and Michalski [78]	Rule-based learning	Both	Instance selection
Klinkenberg and Joachims [62]	Support Vector Machine	Real	Instance selection
Klinkenberg and Rüping [65]	Support Vector Machine	Real	Instance weighting
Syed et al. [109, 108]	Support Vector Machine (SVM)	Population	Incremental classifier
Ruping Androutsopoulos et al. [2]	Naïve Bayes (NB)	Population	Instance selection
Street and Kim [106]	Decision Trees (DT)	Both	Ensemble learning
Hulten et al. [50]	Decision Tree	Real	Incremental classifier
Kolter and Maloof [68]	Decision Tree, Naïve Bayes	Real	Ensemble learning
Stanley [105]	Decision Trees	Real	Ensemble learning
Wang et al. [119]	Decision Trees, Naïve Bayes	Both	Ensemble learning
Delany et al. [31]	Instance-based (K-NN)	Population	Instance selection
Wang et al. [120]	Decision Tree	Real	Incremental classifier
Cohen et al. [26]	Decision Tree	Population	Incremental classifier
Tsymbal et al. [112]	Decision Trees	Population	Ensemble learning
Mohammed et al. [81]	Multi-Layer Perceptron (MLP)	Population	Ensemble learning
Muhlbaier and Polikar [82]	MLP, NB, SVM	Real	Ensemble learning
Parikh and Polikar [88]	Multi-Layer Perceptron	Population	Ensemble learning
Tsymbal et al. [113]	Decision Trees	Both	Ensemble learning

2.5 Support Vector Machines

The SVM classifier is a machine learning approach based on the structural risk theory introduced by Vapnik in [117]. In particular, an SVM classifier is capable of finding the optimal hyperplane that separates two classes. This optimal hyperplane is a linear decision boundary separating the two classes and leaving the largest possible margin between the samples of the two classes. 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 datasets, since classification success does not depend on the dimensions of the input space. Because of this, it can be very promising for incremental learning situations, and so we employ it here.

In particular, the training of an SVM classifier can be summarized as follows. Consider a set of labeled training samples represented by $\mathcal{D}$=$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, where $\mathbf{x}_i \in \Re^d$ 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:

$$\begin{aligned} &\min \quad \tfrac{1}{2} \parallel w \parallel^2 + C \textstyle\sum_{i=1}^{n} \xi_i, \\ &\text{subject to } y_i((w^T \mathbf{x}_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \text{i=1,}\ldots, n \end{aligned} \tag{2.1}$$

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, and ξ_i is a non negative slack variable for $\mathbf{x}_i$. The optimization problem in Equation 2.1 is usually solved by obtaining the Lagrange dual, which can be reformulated as:

$$\begin{aligned} &\max \quad \tfrac{1}{2} \textstyle\sum_i^n \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j, \\ &\text{subject to } 0 \leq \alpha_i \leq C, \ \textstyle\sum_i^n \alpha_i y_i = 0 \end{aligned} \tag{2.2}$$

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 cases, the number of support vectors will be the same as the number of samples contained in the training set. As a result, the w vector can be denoted as $\sum_i^n \alpha_i y_i x_i$. Figure 2.5 illustrates the general idea of the decision boundary computed by the SVM, where there are two classes (circles and squares) separated by an optimal hyperplane. The training samples that were selected as support vectors are located under and between the dashed lines (margin).

However, this SVM formulation only works for linearly separable classes, and, since real-world classification problems are almost never solved with a linear classifier, an extension is needed for nonlinear decision surfaces. To solve this problem, the dot products $(\mathbf{x}_i.\mathbf{x}_j)$ in the linear algorithm are replaced by a nonlinear kernel function $K(.)$, where $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i).\Phi(\mathbf{x}_j)$,

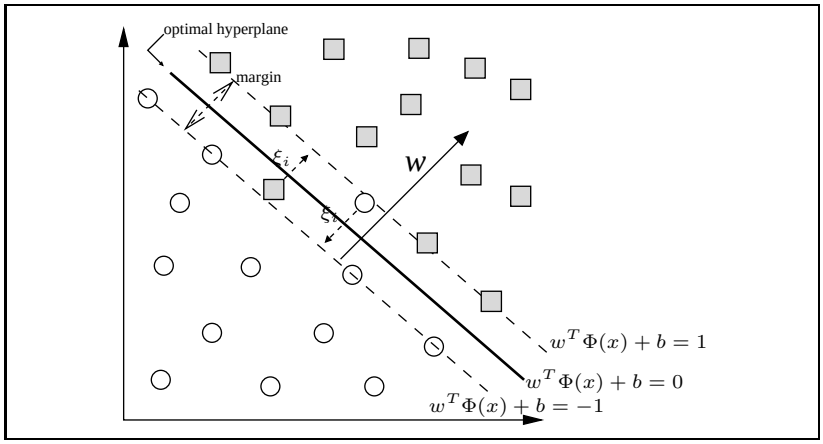


Figure 2.5 Illustration of SVM optimal hyperplane separating two classes.

and Φ is a mapping function $\Phi : \Re^d \mapsto H$. Such a replacement constitutes the so-called "kernel trick" [10]. In order to work, the kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ must satisfy the Mercer condition [117]. The kernel trick enables the linear algorithm to map the data from the original input space $\Re^d$ to some different space H (possibly infinitely dimensional), called the feature space. In this space, nonlinear SVMs can be generated, since linear operations in that space are equivalent to nonlinear 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 2.2. The decision 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:

$$sign(f(\mathbf{x})) \quad \text{with} \quad f(\mathbf{x}) = \sum_{i}^{l} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \tag{2.3}$$

Table 2.2 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)	$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$,$\gamma > 0$
Sigmoid	$K(\mathbf{x}_i, \mathbf{x}_j) = tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + \tau)$

In the same way that this extension deals with nonlinear problems, the primary SVM formulation requires additional modification to solve multiclass problems ($c > 2$). There are two approaches for handling this:

- One-Against-One (OAO): This strategy arranges pairs of classifiers into separate classes, and is also called a 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 the class with the maximum number of votes to it.

- One-Against-All (OAA): In contrast, the one-against-all 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.

In this work, we use the OAO strategy, since it has been demonstrated to be faster to train and uses fewer support vectors than the OAA approach [47]. Overall, the SVM is a powerful classifier with strong theoretical foundations and good generalization performance. However, even though it occurs in most machine learning algorithms, training it requires fine-tuning of its hyperparameter set (i.e. kernel parameters and the 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 overfit the training data or underfit them. Kernel parameters that are not well tuned can also lead to underfitting or overfitting 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 C setting can cause an explosion in the number of support vectors identified, thereby increasing the complexity of the classifiers obtained. So, tuning the SVM hyperparameters controls the classifier's power of generalization. The problem now is to find their best values, which is a non trivial task (the so-called "model selection" problem). In the next chapter, we explain this problem and relate it to dynamic optimization problems.

2.6 Incremental Support Vector Machines

The SVM classifier has robust theoretical fundamentals, and often demonstrates good empirical results in the literature. Unfortunately, its training process is very time-consuming when dealing with large or noisy datasets. This is mainly because the original SVM formulation involves solving a quadratic programming problem, which requires that all training samples be loaded into computer memory at once. For this reason, many incremental support vector methods (ISVM) have been proposed to provide options for updating an existing model that will minimize the computational cost in terms of memory and processing time. Without ISVMs, the application of SVMs could be unviable in these situations. Here, we have grouped the most incremental SVM approaches found in the literature into two categories, according to the way in which they conduct the incremental process:

- *Manipulating Sample Sets (MSS):* These ISVM methods update a classifier by merging new data, old support vectors, and, optionally, additional samples considered relevant in an iterative training procedure. Other non important samples are discarded after training, or used for recursively testing the models generated [108, 109, 80, 123, 35, 1]. Other approaches even filter samples before retraining the model [91].

- *Preserving Karush-Kuhn-Tucker Conditions (PKC):* These ISVM training algorithms attempt to incrementally approximate an optimal decision boundary by adding a new sample to the solution and "adiabatically" updating Lagrange coefficients (α_i), and retaining the Karush-Kuhn-Tucker conditions on all previously seen data [32, 103]. A sample-discarding procedure is implemented based on a kind of 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.

We have compiled some related works in Table 2.3. Most of the ISVM methods were introduced in order to reduce training time over a dataset, and so no special techniques, like

Sequential Minimal Optimization (SMO) or chunking-based optimization, were used in the process.

Table 2.3 Compilation of incremental SVM methods

Ref.	Grp.	Description	Requirements/data feeding (DFe)/Hyper-parameters selection (HS)/general comments
[108]	MSS	Current support vectors are merged with new data for classifier updating.	No. / **DFe:** Chunks. / **HS:** No.
[80]	MSS	Based on condensed nearest neighbor classification technique, it exchanges samples between two extra sets w.r.t. their correctness and distance to separating hyperplane. The process stops whether a user specified level of accuracy is reached on a extra test set or one of the sets become exhausted.	Validation set, pre-setting of accuracy level intended, number of nearest neighbors, multiple incremental sessions. / **DFe:** One training set, i.e. no chunks. / **HS:** No.
[123]	MSS	This approach uses three sample sets during the process, samples are classified, selected and exchanged between extra sets.	$\alpha, \beta, \gamma, \Delta d, \delta$ controls storage and performance rates; multiple incremental sessions. / **DFe:** Chunks. / **HS:** No
[11]	MSS	The idea is to train an ISVM from samples that represent vertices from convex hulls for each class. It is impractical, since the complexity of convex hull computation can be huge depending on the input space dimension.	Impractical, the complexity of convex hull computation can be huge w.r.t. sample space dimension. No experiments were performed. / **DFe:** It could be from chunks. / **HS:** No.
[17]	PKC	It tries to retain the Karush-Kuhn-Tucker (KKT) conditions on all previously seen data, while “adiabatically" adds a new sample to the solution. Leave-one-out is performed for “unlearning" samples.	Leave-one-out estimation for discarding samples. / **DFe:** One training set, i.e. no chunks. / **HS:** No. No experiments were performed.
[35]	MSS	Four strategies are compared in this work. Error-driven technique, which keeps only the misclassified data. Fixed-partition that is similar to [108]. Exceeding-margin that keeps new samples that exceed the margin defined by the current SVM model. And finally, a combination of exceeding-margin+errors-driven technique.	**DFe:** Chunks / **HS:** Cross-validation to set parameters over the first chunk, but used values were not mentioned. / Reported results indicate that the fixed partition [108] overcomes the other strategies.
[94]	MSS	This method attempts to re-learn only a neighborhood from new data and update weights of old data.	Two extra parameters: number of neighbors and a well suited error estimate. Multiple incremental sessions. **DFe:** One dataset, no chunks. **HS:** No.
[95]	MSS	This method works similarly to [108]. However, it changes the SVM formula to compute the loss function by adding a weight to punish errors on previous support vectors.	Extra parameter to weight previous support vectors. **DFe:** Chunks / **HS:** No.
[32]	PKC	The authors expand the work introduced in [17] to enable hyper-parameters updating during incremental learning sessions.	Leave-one-out estimation for discarding samples. **DFe:** One training set, i.e. no chunks. / **HS:** Yes, through gradient-based search. / One database is used (PIMA from UCI), no performance results were reported.
[124]	MSS	The idea is to reduce SVM training time by filtering a large training dataset and training a SVM only from filtered samples. In order to achieve this, the authors use a clustering algorithm.	There are extra parameters for the clustering algorithm and a linear SVM is used (i.e. no kernel parameters, just linear decision function) / **DFe:** Chunks / **HS:** No.
[101]	MSS	The authors propose a modification to adapt the SMO algorithm for online learning.	The approach has serious limitations, since it works only for binary features and linear SVMs. Besides, often with little degradations of performance can be observed. **DFe:** Chunks. / **HS:** No.
[1]	MSS	It pre-extracts support vectors candidates from new data to reduce computational training time. The pre-extraction is done based on a relative distance between samples to optimal hyperplane and a correctness rate of test over all previous data.	Relative distance, multiple incremental sessions for one chunk. / **DFe:** Chunks. / **HS:** No.
[91]	MSS	Like in [108], the previous SVs replaces all historical samples in the retraining process. In contrast, new chunks are “filtered" and only some samples are considered for training. The importance of a sample is measured w.r.t. an adaptive distance to the hyperplane.	No mention on how to implement, set, or measure such a distance adaptively. Some experiments with user graphics. **DFe:** Chunks. / **HS:** No.
[103]	PKC	As in [17], this method also works based on updates at the level of samples coefficients.	While updating hyper-parameters, previous data must be used. / **DFe:** One training set, i.e. no chunks. / **HS:** Yes, gradient-based search.

The term “incremental" is used to describe the process of building models incrementally based on recursive procedures (training and testing) applied to a selection of samples from different subsets and the large original training set. Their goal is to generate a final model more quickly than by solving a larger quadratic optimization problem. Also, it is important to note that SVM hyperparameters are usually set beforehand, without using more sophisticated methods, such as evolutionary computation, for example.

In addition, the terms *exact* or *approximate* SVM usually appear in the literature in connection with ISVM methods. These terms are specifically related to the resolution of the quadratic problem for building the final SVM classifier. If, for example, the final SVM solutions are found using all the training samples during the resolution of the quadratic optimization problem, the SVM is described as exact. The chunking decomposition method and SMO are examples of other methods that provide an exact solution.

By contrast, when the algorithm considers finding the SVM solution by employing one sample at a time (single pass), it is called an approximate ISVM. This is because they do not check other samples, and so the final solution is not optimal. Taking this into account, we suggest in our proposed method a modified version of Syed et al.’s method [108, 109], which we will introduce in section 5.1.2.1. In addition, the SVM implementation used in this thesis already provides mechanisms to accelerate SVM training through SMO. Such a technique is very efficient and demands less computational effort than traditional quadratic programming solvers, as shown in [92].

2.7 Discussion

This chapter surveyed the main approaches for developing mechanisms to learn from imprecise environments. From this literature review, we note that the former methods were proposed based on the instance selection and weighting approaches. In recent years, though, more sophisticated methods have been developed using incremental classifier and ensemble learning approaches. Furthermore, when dealing with real-drift scenarios, most works use only linearly separable synthetic classification problems (e.g. SEA concepts, rotating hyperplane, etc.)

[68, 119, 120]. By contrast, population drifts have been studied in more realistic scenarios, i.e. with real-world data [26, 112, 88].

Most importantly, we have seen that the main incremental learning approaches introduced in the literature are based on different techniques: (1) single incremental learning classifiers; or (2) ensemble of classifiers. We have noted that some authors have adapted traditional machine learning algorithms when using single classifiers, e.g. DTs [50] and SVMs [109], to support incremental learning. In the former approach, the methods update a learner from blocks of data, while the second approach usually uses the serial combination of several individual classifiers. In this thesis, in order to make our system more robust and capable of achieving high performances, we adopt an ensemble of incremental learners in a parallel structure that combines optimized members over time. Our system also employs the concept of relevant samples, inspired by the idea underlying the instance selection approach.

In addition, the choice of a base learner for a classification system is very important. Through this literature review, for example, it can be seen that several incremental versions of classical classifiers have been implemented. So, in order to select a specific classifier, characteristics such as power of generalization, computational complexity, and storage space required by the learner must be analyzed. LVQ and Nearest Neighbors may be easily employed as incremental algorithms for this purpose. However, they demand a great deal of storage space if problems with large databases are considered. The Naive Bayes classifier is very suitable for updating an existing classifier as it learns quickly, but it usually produces more generalization errors. By contrast, although the SVM classifier is relatively more computationally complex, it is asymptotically much better than the Naive Bayes and other classifiers. As for neural networks with incremental capabilities, they often have several parameters to fit and are very sensitive to the order in which examples are presented. For these reasons, we have selected the SVM classifier as part of the core of the system proposed in this thesis.

From our literature review, it can also be noted that, no matter what the incremental learning approach, no consideration has been given to tuning the system parameters over time. In other

words, system updating is always performed based on the same fixed parameter values, or at the classifier combination levels in the ensemble approaches. Thus, updating classifiers with the adaptation of their parameters has not yet been investigated. Moreover, recent results indicate that using well-tuned incremental learners could achieve better performances than just moderate ones [88].

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

In order to overcome this challenge, an incremental learning system must be able to accommodate new data at no detriment to knowledge already learned [93], but it must also better adapt its parameters. The approach we propose for adaptive incremental learning takes this into account by regarding incremental learning as a dynamic optimization process. In particular, it employs knowledge acquired from previous optimization processes to decrease the computational cost of frequent reoptimization.

From an optimization point of view, our assumption is that the natural data changes mentioned above are sources of uncertainty reflected in dynamic changes to the parameter search space. Such uncertainties become even more intense when the search for optimum parameter values must be performed over time. In the literature, dynamic optimization problems are categorized into three types: (I) the location of the optimum changes over time and the amount of shift is quantified by a severity parameter; (II) the location remains fixed, but the value of the objective function changes; and (III) both the location and the value change [83]. In this thesis, we demonstrate empirically that the reoptimization of classifiers over time can be seen as a type III problem.

Moreover, from the analysis of SVM ensembles presented in [116], which shows that the performances of a single SVM classifier can be improved over small datasets 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 to try to achieve better performances than with a single incremental learner.

However, instead of picking up parameter values from an arbitrary grid of options, as in [116], the proposed method explores the self-organization power of the swarm intelligence theory and dynamic optimization techniques. In this way, the proposed approach is able to dynamically move a population of solutions towards optimum regions in the system parameter search space. Finally, the aim is to combine an optimized population of hypotheses that are well placed over the search space and that can even be multimodal, and so become a more robust system than when only single models are used. In the next section, we introduce the first steps in the development of the system, which are to study the SVM model selection problem performed over time as a dynamic optimization problem, and to propose a solution to it.

CHAPTER 3

A PSO-BASED FRAMEWORK FOR THE DYNAMIC SVM MODEL SELECTION (DMS)

In the previous chapter we outlined that the Support Vector Machine (SVM) is a very powerful classifier. However, we also mentioned that its efficiency in practice relies on the optimal selection of hyper-parameters. The search process for optimal values for its hyper-parameters is the so-called SVM model selection problem.

In this chapter 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.

Despite of some search methods have practical implementations, e.g. gradient descent, they usually are limited by difficulties related to the model selection process. For example, the gradient descent methods require a differentiable objective function with respect to the hyper-parameters and the kernel. In this case, the kernel is also required to be differentiable. Likewise, multiple local minima in objective functions also represent a hard challenge for gradient descent based methods. To tackle this, the use of grid-search and evolutionary techniques are interesting alternatives. However, the grid-search method needs a good discretization of the search space in fixed values, which is crucial to reach high performances. Thus, the determination of objective function to be employed, 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.

Additionally to these difficulties, 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 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 issue 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 chapter, we first 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.

The proposed method is a Particle Swarm Optimization (PSO) based framework to select optimal models in a dynamic fashion over incoming data. 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?

This chapter is organized as follows. In section 3.1 we explain the relation between the model selection problem and dynamic optimization problems. Our proposed method is introduced in section 3.2. Finally, the experimental protocol and results are described in section 3.3. Discussions and conclusions are presented in section 3.4.

3.1 SVM Model Selection as a Dynamic Optimization Problem

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:

a. Collect training data;

b. Start the search for solutions;

c. Find the hyper-parameters that perform best;

d. Train the final model with the best hyper-parameters;

e. Assess the performance of the final model using the test set.

In Table 3.1 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 [115]. Because of this, the RBF kernel is considered in this study.

Table 3.1 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[1]	Search method	Objective function
[35]	RBF	Grid-search (GS)	ν-Cross-validation
[20]	RBF	Gradient descent (GD)	Radius-margin, Span bounds, Leave-one-out
[18, 47]	RBF	Grid-search (GS)	ν-Cross-validation error (CV)
[22]	RBF	Genetic algorithm (GA)	Radius-margin bound
[25]	RBF	Genetic algorithm (GA)	ν-Cross-validation error (CV)
[3]	RBF	Gradient descent (GD)	Hold out error, radius-margin, Generalized Approximate CV error (GACV)
[32]	RBF	Gradient descent (GD)	Leave-one-out (LOO), span bound
[116]	RBF,POL	Grid-search (GS)	ν-Cross-validation
[103]	POL	Gradient descent (GD)	Generalization error estimation bound
[107]	RBF	Multi-objective GA (MOGA)	Modified radius-margin bounds
[29]	RBF	Particle Swarm Optimization (PSO), Grid-search (GS)	Hold out error, $\xi\alpha$-estimator
[49]	RBF	Uniform design (UD), Grid-search (GS)	ν-Cross-validation
[52]	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.

Most of effort associated with the approaches listed in Table 3.1 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.

By contrast, data from real-world applications are usually far from perfect, which 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 which the optimal solution can change overtime in different ways [53]. The changes can result from variations in the objective function, which implies in fitness dynamism. Figure 3.1 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. By way of illustration, we depict in Figure 3.2 an overview of the SVM model selection task seen as an optimization process and the possible uncertainties involved.

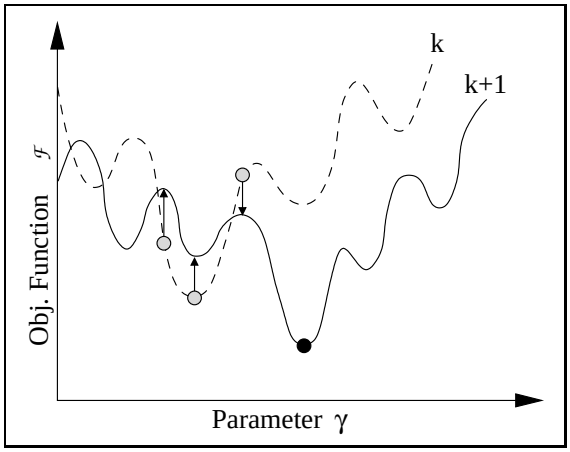


Figure 3.1 Illustration of changes in the objective function. In a first moment (k), solutions are approximated for a parameter γ. Next, due to some unexpected change, e.g. new data, noise, etc.,the objective function changes and the solutions (gray circles) stay trapped in a local minimum, what requires some kind of reaction to adapt the solutions to the new function and optimal (dark point) in $k + 1$.

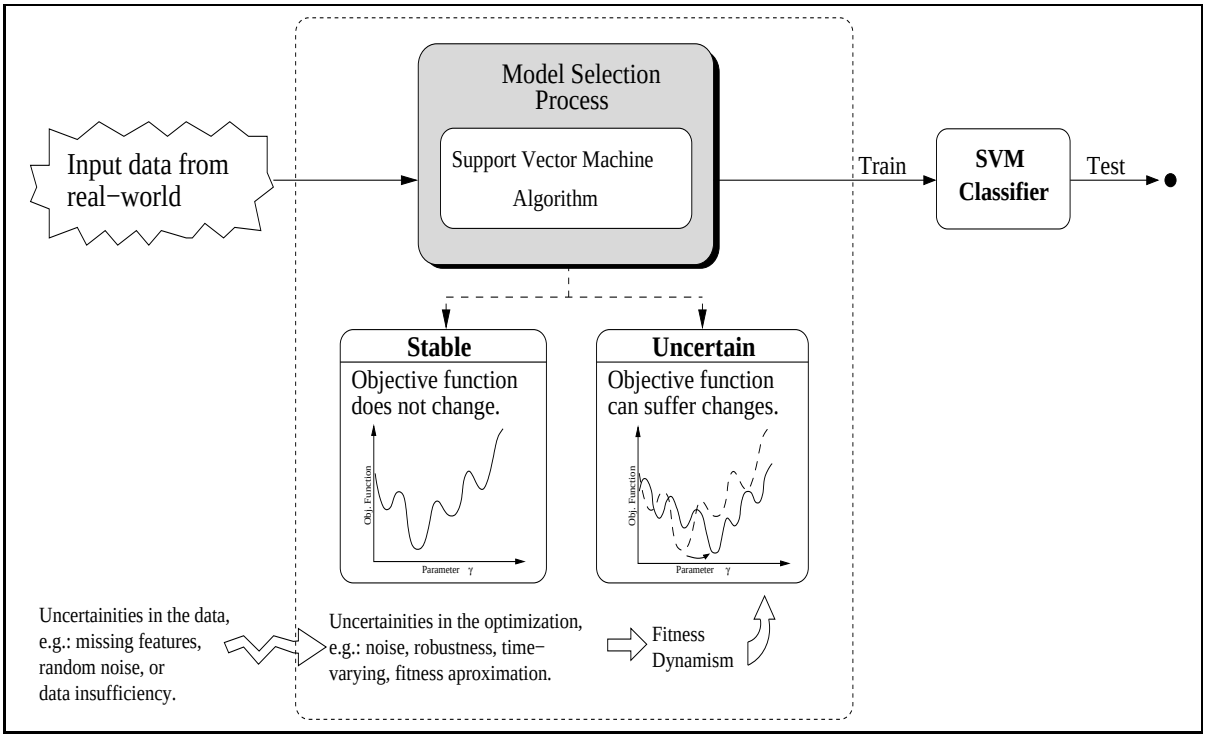


Figure 3.2 Overview of the SVM model selection task seen as an optimization process and the possible uncertainties involved.

To demonstrate this fact, we depict a case study in Figure 3.4 regarding the $min(\epsilon(\mathbf{s}, \mathcal{D}(k)))$ mentioned above for a two-class (I and II) classification problem called P2, which is depicted in Figure 3.3. More details about the construction of this synthetic classification problem can be found in the appendix I.

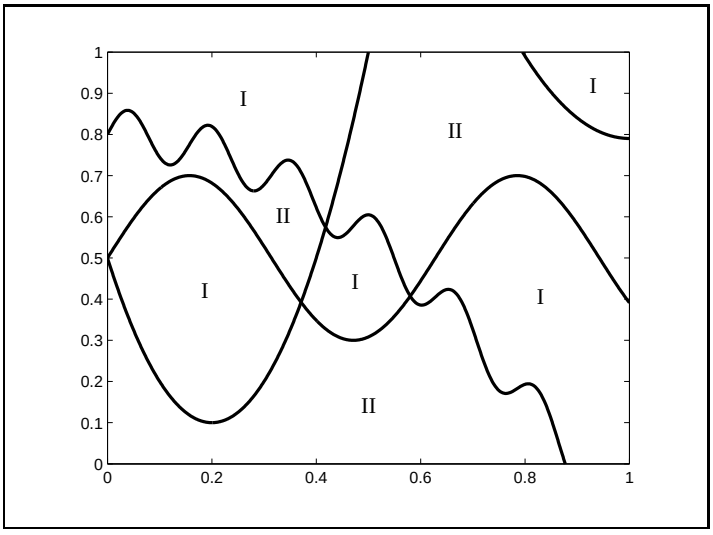
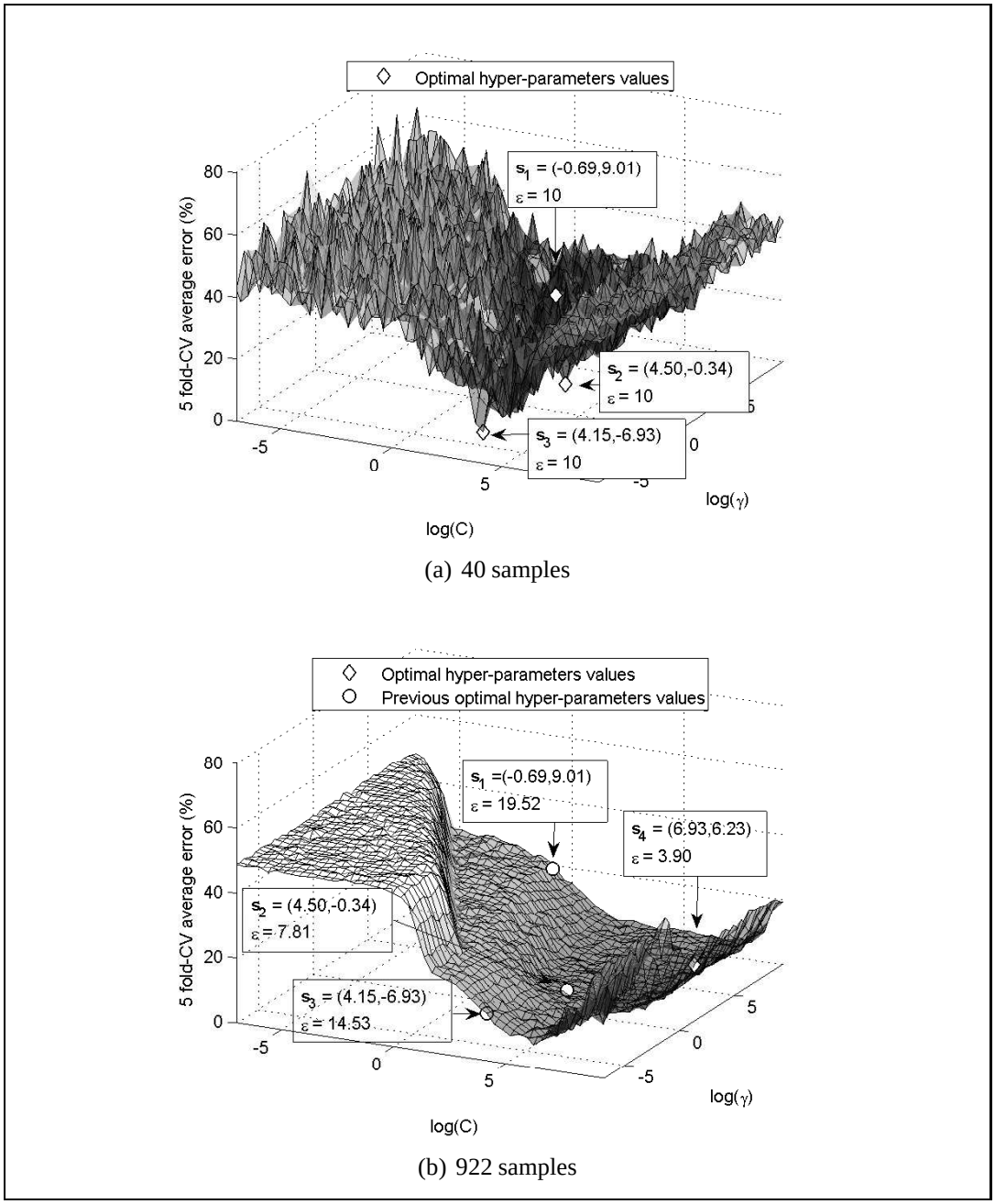


Figure 3.3 Illustration of the P2 classification problem.

So first, in Figure 3.4(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 Figure 3.4(b). It can be seen that the search space and the optimal solutions 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). 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.



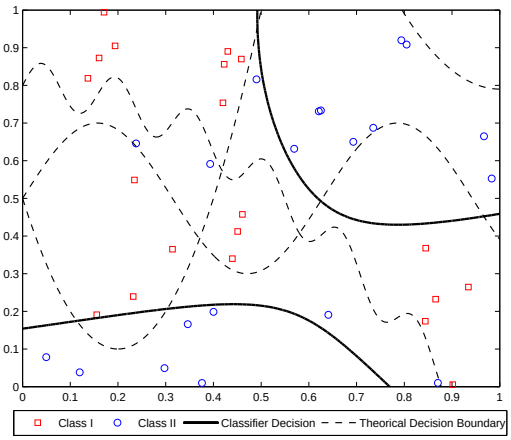
(a) 40 samples

(b) 922 samples

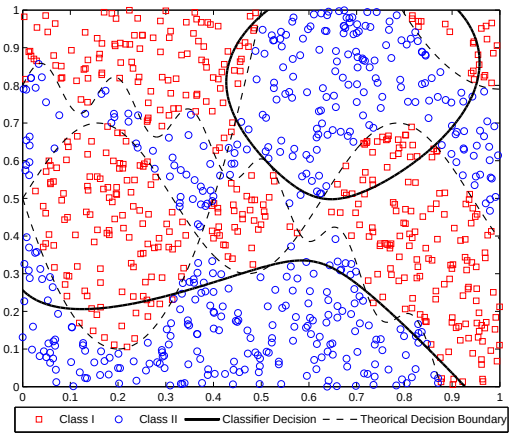
Figure 3.4 Hyper-parameter search space for P2 problem with different number of samples.

In order to show the effect that these hyperparameters changes produce in obtaining a final SVM model, we depict in Figure 3.5, 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.

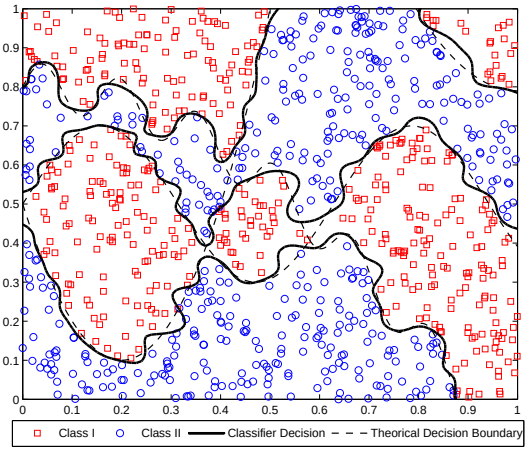
From these results, we can see that despite of $\mathbf{s}_2$ adequately separates the classes given a certain knowledge about the problem (Figure 3.5(a)), it is not capable of producing the same satisfatory results (Figure 3.5(b)) that a new best evaluated solution (i.e. $\mathbf{s}_4$) can achieve (Figure 3.5(c)) if more samples are considered.



(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, $\epsilon = 3.90\%$

Figure 3.5 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. (b) Decision boundaries obtained for the same optimum solution $\mathbf{s}_2$ for 40 samples, but now training over 922 samples. (c) Final result achieved for the best solution $\mathbf{s}_4$ regarding 922 samples.

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 seach processes from scratch every time.

3.2 The Proposed Dynamic SVM Model Selection Method (DMS)

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

An overview of the general concept proposed is depicted in Figure 3.6. 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. On the other hand, when it is not possible to identify a satisfactory solution even after an adapted grid search, the method starts a dynamic optimization process, as 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.

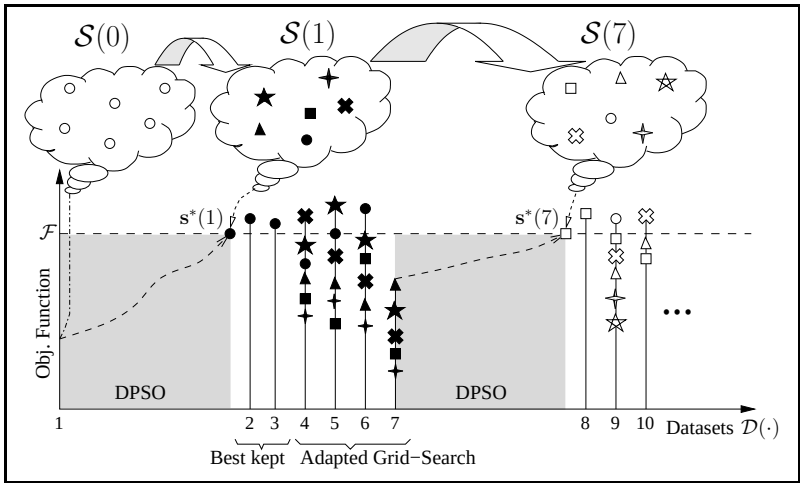


Figure 3.6 Overview of the proposed model selection strategy (conceptual idea). Optimum solutions for a current dataset $\mathcal{D}(k)$ are pointed out by switching among three search strategies: 1) use the best solution $\mathbf{s}^*(k-1)$ found so far, 2) search for a new solution over an adapted 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.

3.2.1 Framework for the Dynamic Selection of SVM models

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 Figure 3.6 and also by concepts of dynamic optimization problems introduced in section 3.1, we propose a framework for the dynamic selection of SVM models over time.

In particular, our general framework for the dynamic selection of SVM models is composed of three main modules: change detection, adapted grid-search, and dynamic particle swarm optimization (DPSO). Figure 3.7 depicts its general idea. In addition, we summarize its working in Algorithm 1.

Details on each one of these modules are described in the next sections. The *upgrade_stm* and *recall_stm* functions are respectively responsible for storing and retrieving optimized solutions from the system's *Short Term Memory* (STM).

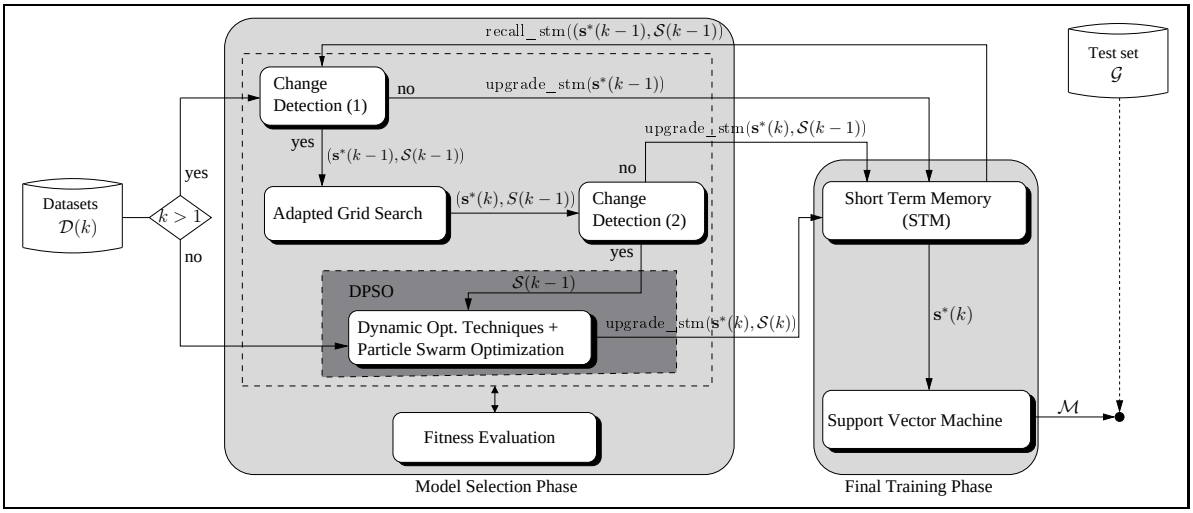


Figure 3.7 General framework for the dynamic SVM model selection.

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.

3.2.1.1 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 Equations 3.1 and 3.2) [26]. 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. $\mid \epsilon(\mathbf{s}^*, \mathcal{D}(k-1)) - \epsilon(\mathbf{s}^*, \mathcal{D}(k)) \mid \geq \delta_{max}$), then other levels are activated for additional searches.

Figure 3.8 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.

$$\delta_{max} = z_{0.9} \times \sqrt{\sigma} = 1.282 \times \sqrt{\sigma} \tag{3.1}$$

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

$$\sigma = \frac{\epsilon(\mathbf{s}^*, \mathcal{D}(k-1)) \times (1 - \epsilon(\mathbf{s}^*, \mathcal{D}(k-1)))}{W(\mathcal{D}(k-1))} + \frac{\epsilon(\mathbf{s}^*, \mathcal{D}(k)) \times (1 - \epsilon(\mathbf{s}^*, \mathcal{D}(k)))}{W(\mathcal{D}(k))} \tag{3.2}$$

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.

3.2.1.2 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* (STM). 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 Figure 3.9 for two hyper-parameters (C and γ)), the

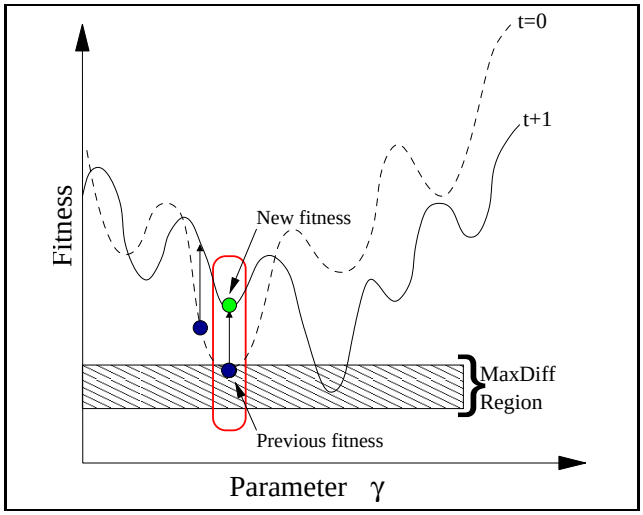


Figure 3.8 Illustration of the change detection mechanism. In this case, as the new fitness is situated outside the expected region, a new optimization is carried out in order to find a new better solution.

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, for example. 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 Figure 3.7. If the current solution is still not considered satisfactory, the dynamic optimization module is activated.

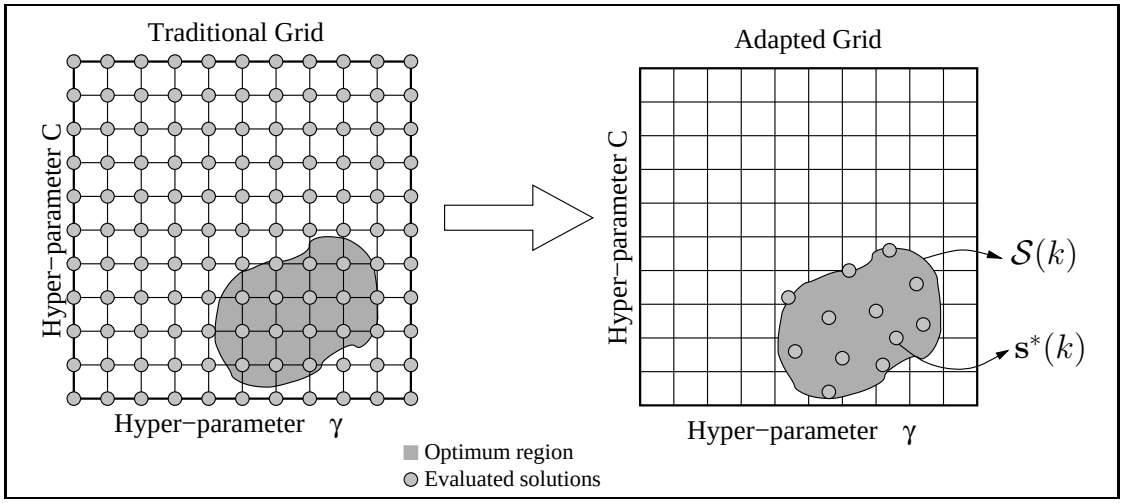


Figure 3.9 The traditional grid must try a higher number of combinations than the adapted grid, which profits from the already optimized solutions $\mathcal{S}(k)$ provided by DPSO. $\mathrm{s}^*(k)$ denotes the best solution.

3.2.1.3 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 [59]. 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 [60]. 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 $\mathcal{S} = \{\mathbf{s}_i, \dot{\mathbf{s}}_i, \mathbf{s}'_i\}_{i=1}^{P}$[1] 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 $\mathbf{s}'_i = (s_i'^1, s_i'^2, \ldots, s_i'^d)$ 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 hyper-parameter 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 (i.e. $s'_{net(i,\lambda)}(q)$, where λ is the number of neighbors which communicate with particle index i) and the particle's own experience $\mathbf{s}'_i$. For a new iteration $q+1$ and dimension d, the update is computed as follows:

$$\dot{s}_i^d(q+1) = \chi(\dot{s}_i^d(q) + \phi r_1(s_i'^d(q) - s_i^d(q)) + \phi r_2(s'^d_{net(i,\lambda)}(q) - s_i^d(q))) \tag{3.3}$$

where χ is the constriction coefficient introduced by Clerc [24], and r_1 and r_2 are random values. Constriction coefficient values of $\chi = 0.7298$ and $\phi = 2.05$ are recommended [60].

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

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 Equation 3.4. Figure 3.10 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.

$$s_i^d(q+1) = s_i^d(q) + \dot{s}_i^d(q+1) \tag{3.4}$$

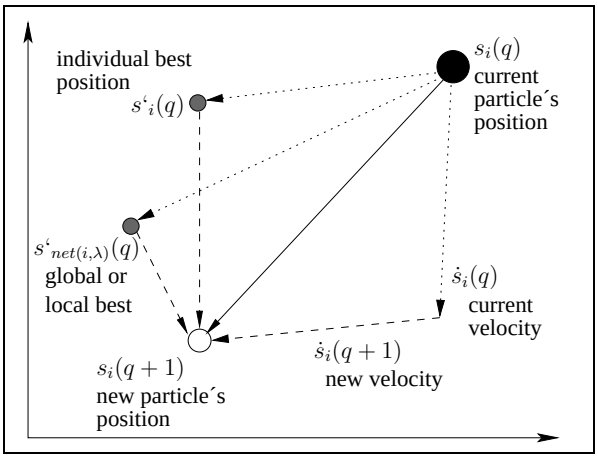


Figure 3.10 Example of a particle's trajectory during position updating.

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 $1 \leq i \leq P$ **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 $\mathbf{s}'_i(q)$
  for all particles i such that $1 \leq i \leq P$ **do**
    Update velocity $\dot{s}_i(q)$ (Equation 3.3) and current position $\mathbf{s}_i(q)$ (Equation 3.4)
  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 [60]. Because of this, we implement this module based on PSO with a more sophisticated topology called local best (*lbest*) [60]. 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 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 Figure 3.4). 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 [12, 48] 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 Figure 3.7 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 [12]. 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 [48]. 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 [23] 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 $1 \leq i \leq P$ **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 $\rho\%$ 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 $1 \leq i \leq P$ **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 $1 \leq i \leq P$ **do**
17: Update velocity $\dot{s}_i(q)$ (Equation 3.3) and current position $\mathbf{s}_i(q)$ (Equation 3.4)
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 Figure 3.11. This case study represents an empirical reference to the general concept illustrated

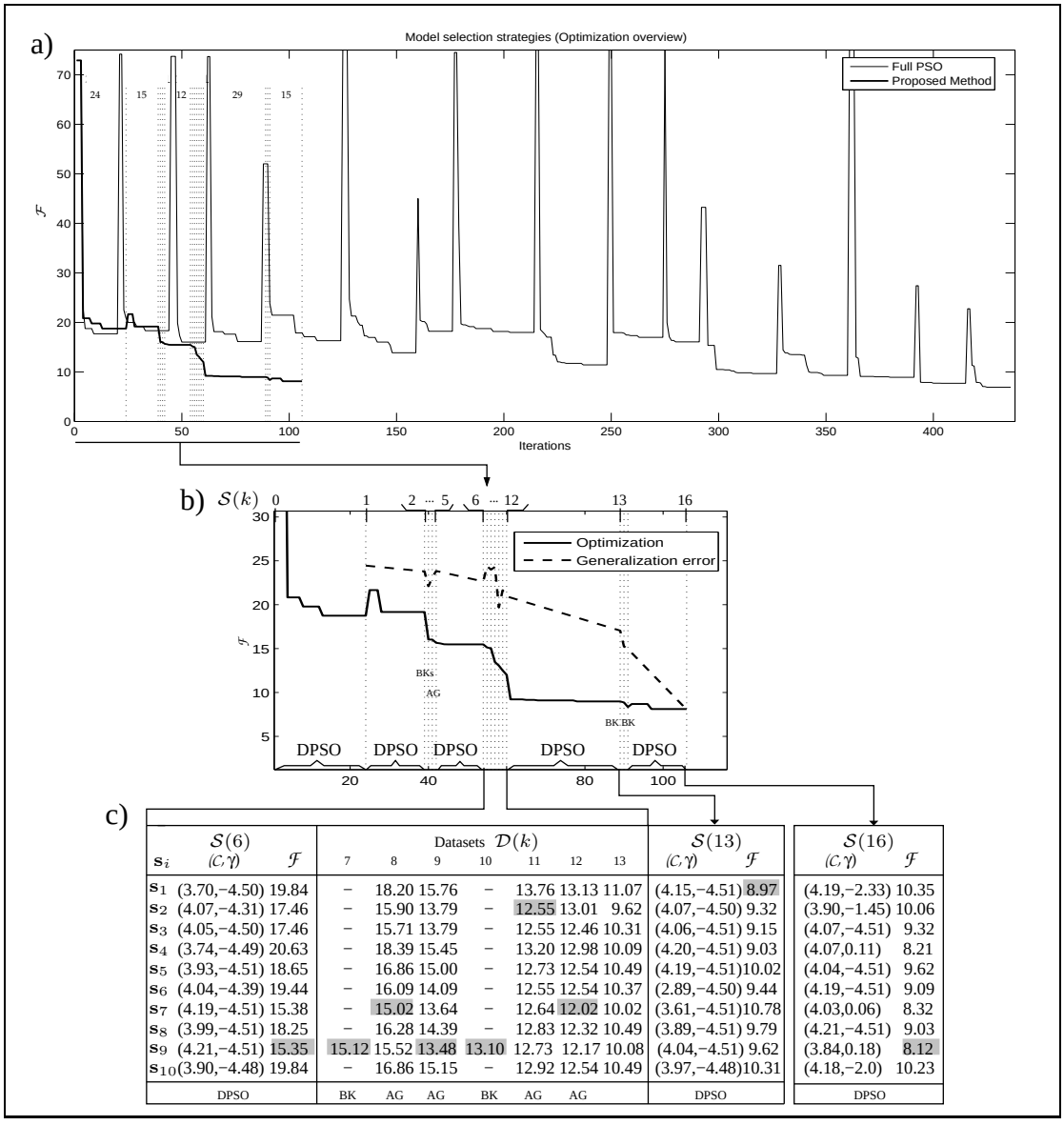
in Figure 3.6. In particular, it depicts overviews of searching processes carried out by the proposed method and full optimization processes over cumulative sequences of data increased logarithmically from the Satimage database. Based on these results, it is shown in Figure 3.11 (a) that the proposed method can achieve similar results to those obtained by full optimization processes with PSO (Full PSO), but more quickly and in fewer iterations 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 Figure 3.11 (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 Figure 3.11 (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 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



	$\mathcal{S}(6)$		Datasets $\mathcal{D}(k)$							$\mathcal{S}(13)$		$\mathcal{S}(16)$	
$\mathbf{s}_i$	(C,γ)	$\mathcal{F}$	7	8	9	10	11	12	13	(C,γ)	$\mathcal{F}$	(C,γ)	$\mathcal{F}$
$\mathbf{s}_1$	(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	

Figure 3.11 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 Figure 3.6.

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.

3.3 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 [45], 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 has 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.

3.3.1 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 (7x10) 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 3.2.

3.3.2 Experiments Parameters Setting

The following parameters setting was 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 [60]. 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 [118], span bound [19], support vector count [117], etc. More information about them can be found in [20].

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 [40]. 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 [18].

3.3.3 Datasets

We have used nine synthetic and real-world datasets in the experiments. They are listed in Table 3.2 along with more details. The synthetic problems used were the well-known Circle-in-Square (CiS) [14] and P2 [116] 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 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 [45], 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 [4]. The task is to predict whether or not income exceeds $50K/yr based on census data. The DNA, German Credit, and Satimage datasets are from the Statlog Project [79]. 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 3x3 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*. [85]. 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.

Finally, 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 [89] was used in this work. More details on the synthetic, Nist-Digits, and IR-Ship databases can be found in the appendix I.

Table 3.2 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	3,185	19	29,376
Circle-in-Square	2	2	3,856	21	10,000
DNA	3	180	1,400	15	1,186
German Credit	2	24	800	13	200
IR-Ship	8	11	1,785	10	760
Nist-Digits	10	132	5,860	16	60,089/58,646
P2	2	2	3,856	21	10,000
Satimage	6	36	4,435	15	2,000

3.3.4 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 1GB of PC-2700 DDR RAM (333MHz 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.

3.3.5 Obtained Results

The results are reported in Tables 3.3, 3.4, and 3.5, 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.

From 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 3.3 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.

Table 3.3 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 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)**
German Credit	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)**

Table 3.4 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)

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 3.3 and 3.4. 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.

Table 3.5 Mean computational time spent (hh:mm:ss) for model selection processes over all sequences of datasets. Results with FPSO over the whole databases (FPSO-all data) are also reported

Database	FPSO-all data	FPSO	CPSO	DMS
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)**
German Credit	**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)**

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, 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. To better visualize the performance of the methods, we also report the mean error rates across all the subsets and over the 10 replications for two case studies in Figure 3.12.

For a deeper analysis of the proposed method, we have depicted in Figure 3.13 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.

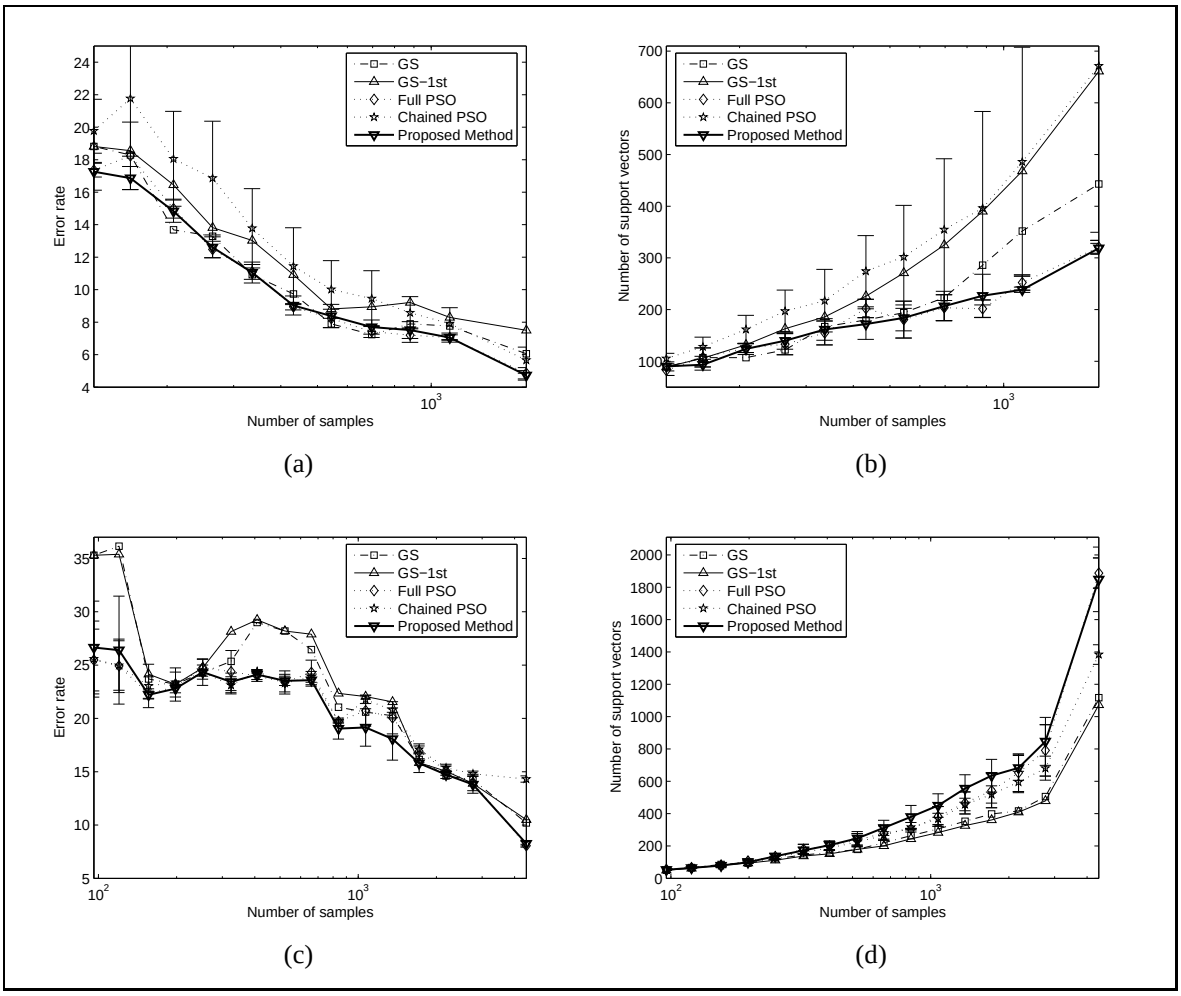


Figure 3.12 Error and support vectors rates. For the databases, Ship ((a) and (b)) and Satimage ((c) and (d)). The results were obtained over 10 replications.

By comparing the optimization approaches directly, we can see that the results reported in Table 3.6 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, increasing

diversity, etc. 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.

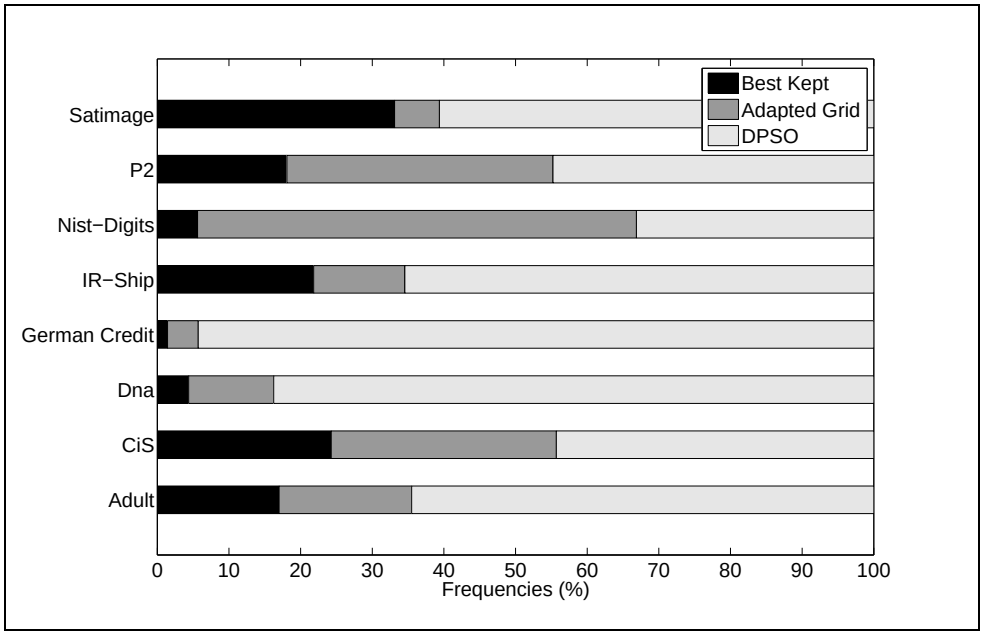


Figure 3.13 Average of frequencies which indicates how many times each module was responsible for pointing out the final solution.

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

tion of DMS gradually over subsets of data can be even faster than realizing a full optimization process over the entire original training set.

Table 3.6 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)
German Credit	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)

Thus, 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).

3.4 Discussion

In this chapter 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 nine 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. By contrast, (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.

Above all, 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, signature verification, etc. The proposed method is also particularly useful for real-world applications requiring the generation or updating of dynamically in a serial way (e.g. those involving streaming data). We present some more additional results that restate our conclusions concerning the strategies tested in the appendix II.

Nevertheless, even considering that the optimization of a single classifier is important to increase its performances, we know that the use of an ensemble of classifiers can improve the overall performance of a classification system. Especially when the members composing the ensemble are especially selected, which makes them still more accurate.

Taking this into account, the evaluation and selection of such classifiers depend on the choice of an adequate objective function. Therefore, in order to better understand and apply classifier ensembles to compose our adaptive incremental system in the context of this thesis, we investigate a series of measures, based on different theories, to achieve such tasks in the next chapter.

CHAPTER 4

TOWARDS TO THE EVALUATION AND SELECTION OF ENSEMBLE OF CLASSIFIERS

The fusion of classifier decisions into ensembles has been widely applied to improve the performance of single classifiers. Over the last years, several efforts on ensembles of classifiers have been conducted to find measures that could be well correlated with ensembles' accuracy [67, 99, 36, 28, 125, 96, 69, 73, 122, 9, 116, 110]. However, despite of the efforts, the understanding of the effectiveness of ensembles methods has still intrigued many authors.

A consensus in the literature indicates the presence of some diversity between the ensembles members as the main factor for improving the overall performance [28, 69, 73, 122, 9]. Even though it is well accepted that diversity is a necessary condition for improving the majority vote accuracy, there is no general agreement on how to quantify or to deal with it. On the other hand, bias-variance and margin theory has also allured some attention in the literature, since it may cast the study of ensembles of classifiers into a large margin classifiers context. In particular, the margin theory was first applied by Schapire et al. [99] to provide an explanation on how the boosting method works. After that, other authors have used this theory to create new ensemble methods [6, 8].

The main goal of this chapter is, through an empirical study, to investigate measures for the evaluation and selection of ensemble members. This is important because sometimes, mainly for those situations in which only small datasets are available, the use of ensemble accuracy over such data may not provide sufficient information to select the best ensemble. The conclusions obtained in this chapter help us on the choice of the best objective function to be used in our adaptive incremental learning system.

In order to achieve this, we start our study by surveying measures from some classical theories: bias-variance, diversity measures, and margin theory for ensembles. Afterwards, an experimental protocol similar to one introduced by Valentini [116] for characterizing ensembles of

Support Vector Machines is employed to evaluate the measures. In addition, a discussion on the obtained results is also offered, in which we try to answer some questions currently found in the literature, such as:

- Which measure could offer the best guidance to evaluate the classifiers fusion?
- How are the diversity measures related to each other?
- Is there a relationship between diversity, margins, and ensemble accuracy?
- Which are the best measures for observing such relationship?

This chapter is organized as follows. In Section 4.1 we summarize the bias-variance theory for ensemble according to Domingos' theoretical framework [36]. Section 4.2 surveys classical measures to estimate diversity for classification fusion. Section 4.3 introduces the margin theory for ensemble of classifiers and measures related to it. Section 4.4 describes the experimental protocol applied and the obtained results. Finally, we discuss results and outline the conclusions in Section 4.5.

4.1 Bias-Variance Decomposition of Error

In general, zero-one loss functions are the only option to be applied to classification problems. In order to analyze bias-variance in this context, an alternative is to use the unified bias-variance decomposition of the error proposed by Domingos [36]. In this theory, regarding a free noise case, the expected loss $EL(\cdot)$ for a sample $\mathbf{x}$ is basically decomposed into two terms: the bias $B(\cdot)$ and the variance $V(\cdot)$. Therefore, following the same notation introduced by Valentini in [116], the expected loss is computed as:

$$EL(x) = B(\mathbf{x}) + V(\mathbf{x}) \tag{4.1}$$

$B(\mathbf{x}) = E(y_m, y_*)$ represents the bias of an ensemble of L classifiers on an example $\mathbf{x}$. The bias is the loss $E(.)$ incurred by the main prediction y_m with respect to the optimal prediction y_*. Therefore, for the 0/1 loss, the bias is always 0 or 1 and computed by:

$$B(\mathbf{x}) = \begin{cases} 1 & \text{if } y_m \neq y_* \\ 0 & \text{if } y_m = y_* \end{cases} \tag{4.2}$$

For an ensemble composed of $\{\mathcal{M}_i\}_i^L$ classifiers, the variance of errors is considered according to two opposite aspects: the unbiased and biased variance. The unbiased variance $V_u(\mathbf{x})$ is the variance when $B(\mathbf{x}) = 0$, it is responsible for increasing the error. On the other hand, biased variance $V_b(\mathbf{x})$ represents the variance when $B(\mathbf{x}) = 1$, hence it is responsible for decreasing the error. These variances are calculated as:

$$V_u(\mathbf{x}) = \frac{1}{L}\sum_{i=1}^{L}[(y_m = y_*) \wedge (y_m \neq y_{\mathcal{M}_i})] \tag{4.3}$$

$$V_b(\mathbf{x}) = \frac{1}{L}\sum_{i=1}^{L}[(y_m \neq y_*) \wedge (y_m \neq y_{\mathcal{M}_i})] \tag{4.4}$$

where $y_{\mathcal{M}_i}$ is the prediction provided by a classifier $\mathcal{M}_i$.

Finally, the net variance $V_n(\mathbf{x})$ is defined in order to combine the effects of unbiased and biased variance $V_n(\mathbf{x}) = V_u(\mathbf{x}) - V_b(\mathbf{x})$. From this point of view, variance can be seen as a measure of diversity, where its effects on error are related to the type of the variance [115, 116].

This decomposition for a sample $\mathbf{x}$ can be generalized to a whole dataset by defining $E_{\mathbf{x}}[.]$. This way the average bias $E_{\mathbf{x}}[B(\mathbf{x})]$, the average unbiased variance $E_{\mathbf{x}}[V_u(\mathbf{x})]$, and the average biased variance $E_{\mathbf{x}}[V_b(\mathbf{x})]$ compose the expected loss of generalization over all dataset is redefined to:

$$E_{\mathbf{x}}[EL(\mathbf{x})] = E_{\mathbf{x}}[B(\mathbf{x})] + E_{\mathbf{x}}[V_u(\mathbf{x})] - E_{\mathbf{x}}[V_b(\mathbf{x})] \tag{4.5}$$

Overall, the bias-variance decomposition of error theory allows to understand the working of ensembles. Unfortunately, as it is defined by a set of metrics, the use of the bias-variance theory for the selection of ensembles is more complex.

By contrast, the margin theory can express the same concepts, but in a more compact way, since the increasing of margins denotes the decreasing of the bias and variance terms together [36]. In light of this, we focus on the margin theory and diversity measures. However, for sake of clarity, we also present results based on bias-variance analysis with two problems in appendix III.

4.2 Diversity Measures

Diversity has been quantified in several ways for classification fusion. As a result, different measures have been proposed in the literature. In this section, we describe seven well-known diversity measures which are usually grouped into two types: pairwise and non-pairwise [73]. Their values vary in a range of 0 and 1. Moreover, in here each diversity measure name is accompanied with a downward arrow $\downarrow$ or upward arrow $\uparrow$ indicating if the diversity obtained is decreasing or increasing with its value.

4.2.1 Pairwise Measures

In pairwise measures, firstly the diversity between all pairs of classifiers is calculated. Thereafter, the overall diversity measure values are computed as the mean of the pairwise values. For instance, given L classifiers, $\frac{L\times(L-1)}{2}$ pairwise diversities d_{ij} are measured between pairs of classifiers, and then the final diversity $\bar{d}$ is defined by an average:

$$\bar{d} = \frac{2}{L(L-1)} \sum_{\substack{i,j=1,\ldots,L \\ i\neq j}} d_{i,j} \tag{4.6}$$

In general for a pairwise measure, n is the total number of samples, n^{11} is the number of times that both classifiers are correct, n^{00} represents the number of times that both classifiers are

incorrect, and n^{10} and n^{01} denote the number of times when just the first or second classifier is correct, respectively. Below, we describe some pairwise measures applied in this work.

4.2.1.1 Q average (↓)

With this measure, classifiers that tend to recognize the same samples correctly will have positive values of Q. This measure is computed for pairs of classifiers i and j as:

$$Q_{i,j} = \frac{n^{11}n^{00} - n^{01}n^{10}}{n^{11}n^{00} + n^{01}n^{10}} \tag{4.7}$$

4.2.1.2 Disagreement measure (↑)

This measure denotes the ratio between the number of observations where one classifier is correct and the other is incorrect with respect to the total number of observations [104]. For a pair of classifiers i and j, it is computed by:

$$DS_{i,j} = \frac{n^{10} + n^{01}}{n} \tag{4.8}$$

4.2.1.3 Double-fault measure (↓)

The double-fault measure estimates the probability of coincident errors for a pair of classifiers. It is defined for a pair of classifiers i and j as [104, 42]:

$$DF_{i,j} = \frac{n^{00}}{n} \tag{4.9}$$

4.2.2 Non-pairwise Measures

Unlike pairwise measures, non-pairwise measures are not calculated by comparing pairs of classifiers, but by comparing all L classifiers as a whole. Below there are some examples of these types of measures:

4.2.2.1 Kohavi-Wolpert (KW) variance (↑)

Let $l(\mathbf{x}_j)$ be the number of classifiers that correctly recognize $\mathbf{x}_j$. From the formula for the variance [67], the diversity measure becomes:

$$KW = \frac{1}{nL^2}\sum_{j=1}^{n} l(\mathbf{x}_j)(L - l(\mathbf{x}_j)) \tag{4.10}$$

4.2.2.2 Generalized diversity (↑)

Let Z be a random variable to represent the proportion of classifiers that are incorrect on a randomly drawn sample $\mathbf{x}$, p_i is the probability that $Z = i/L$, and $p(i)$ is the probability that i randomly chosen members will be wrong on a randomly chosen $\mathbf{x}$. The generalized diversity is defined as [90]:

$$p(1) = \sum_{i=1}^{L} \frac{i}{L} p_i, \quad p(2) = \sum_{i=1}^{L} \frac{i(i-1)}{L(L-1)} p_i \tag{4.11}$$

$$GD = 1 - \frac{p(2)}{p(1)} \tag{4.12}$$

4.2.2.3 Ambiguity (↑)

The ambiguity measure was proposed by Zenobi and Cunningham [125]. Basically, it measures the disagreement among the classifiers predictions $\hat{y}_j$ with respect to the majority prediction y_m, where the factor correctness is not important. The ambiguity measure can be defined as:

$$A = \frac{1}{nL}\sum_{i=1}^{n}\sum_{j=1}^{L}[y_{im} \neq \hat{y}_{ij}] \tag{4.13}$$

4.2.2.4 Difficulty (↓)

Unlike the ambiguity measure, the difficulty measure [44] like most of the measures is calculated taking into account the base classifiers' correctness. The goal is to measure the degree of

classification difficulty of samples. Basically, this measure is defined to be the variance of a $\mathbf{X}$ random variable which denotes the proportion of classifiers that correctly classify a sample $\mathbf{x}$: $DY = \sigma^2(\mathbf{X})$.

4.3 Margin Theory

The margin theory was originally applied to develop the Support Vector Machines theory [117] and to explain the success of Boosting [99]. In the former, Vapnik [117] has introduced the idea that the generalization error of a classifier can be decreased by maximizing the separation margin between classes. Basically, the margin of a sample $\mathbf{x}$ represents a degree of confidence in its classification. Here, in order to provide a global understanding of this theory, we summarize the different ways to compute the margin regarding a sample.

First, the margin of a single classifier based on some discriminate function $f(.)$ over a sample $(\mathbf{x}, y)$ with $y \in [-1, 1]$ and $f(\mathbf{x}) \mapsto [-1, 1]$ can be computed by:

$$\tau(\mathbf{x}, y) = y \cdot f(x) \tag{4.14}$$

Second, if the classifier is based on some probabilistic model, so the margin can be defined as:

$$\tau(\mathbf{x}, y) = P(y|\mathbf{x}) - max(P(y \neq j|\mathbf{x})) \tag{4.15}$$

where j is any other class related to the classification problem. Next, for ensembles of classifiers, the concept of the margin follows the same idea introduced by Schapire et al. [99]. In general, the margin of a sample $\mathbf{x}$ can be computed by Equation 4.16 or by Equation 4.17 [43, 110], where v_y is the number of votes for the true class, v_j is the number of votes for any other class, and c is the maximum number of classes in the problem:

$$\tau(\mathbf{x}, y) = \frac{1}{L}\left(v_y - \sum_{\substack{j=1,\ldots,c \\ j \neq y}} v_j\right) \tag{4.16}$$

$$\tau(\mathbf{x}, y) = \frac{1}{L}\left(v_y - \underset{\substack{j=1,\ldots,c \\ j \neq y}}{\arg\max}\ v_j\right) \tag{4.17}$$

The main difference between these two definitions for ensembles is that, while the first one applies a sum operation, the second one computes a max operation. Based on the first margin definition, when dealing with multi-class problems the margins can even assume negative values for correct ensemble decisions, i.e. when there is a plurality but not a majority [43]. By contrast, following the second definition, which is a special case of the first one, the margins are always positive when the ensemble is correct and negative otherwise. Thus, for the sake of clarity, in this chapter, we employ both definitions and show that in fact they perform similarly and converge to the same regions.

4.3.1 Margin-Related Measures

Naturally, the definition of margin for a sample $\mathbf{x}$ can also be generalized and employed to other measures applied over a dataset $\mathcal{D} = (\mathbf{x}_i, y_i)_{i=1}^{n}$. In particular, there are two main measures related to this theory:

- *Minimum Margin*($\uparrow$): The minimum margin of an ensemble of classifiers on a dataset $\mathcal{D}$ is defined as the smallest value of margin obtained to any correct label [43]. Therefore, the minimum margin is governed by:

$$\varrho(D) = \underset{1 \leq i \leq n}{arg\,min}(\tau(\mathbf{x}_i, y_i)), \tag{4.18}$$

- *Average Margin*($\uparrow$): the average margin denotes the mean of all margins obtained over samples of a given dataset $\mathcal{D}$. This measure can be calculated as:

$$\mu(D) = \frac{1}{n}\sum_{i=1}^{n}\tau(\mathbf{x}_i, y_i) \tag{4.19}$$

In addition to these two typical margin-based measures, another measure has been proposed from the margin theory. In here, we denote this measure as CI measure, since it was derivate from the Chebishev's Inequality. This measure represents a generalization bound suggested in [8]. In particular, assuming an average margin $\mu(\mathcal{D}) \geq 0$, this measure is defined as:

$$CI(\downarrow) = \frac{\sigma(\tau(D))}{(\mu(\mathcal{D}))^2} \tag{4.20}$$

This measure establishes a relation between the strength of the base classifiers (average margin) and the dependence between them for predicting the generalization error. This is because, it has been proven that the variance of the margins is lower or equal to the average of the correlation coefficients of pairs of classifiers times an average of variance between them [8].

Finally, the use of cumulative margin distribution graphics is also an efficient tool to observe the ensembles' behaviors. They can be computed by two simple steps. First, the set of margin values from a dataset is sorted. Next, for each possible value of margin is calculated the percentage of the samples whose margins are lower or equal to the current value. Graphics of cumulative distribution of margins were firstly introduced by Schapire et al. [99] to demonstrate that Boosting maximizes margins. Once that the definitions of diversity and margin theories were already presented, we describe the experimental protocol adopted and the results obtained in the next section.

4.4 Experimental Protocol

In order to investigate the measures previously introduced as objective functions, we have carried out an experimental protocol similar to one realized by Valentini [115, 116] for characterizing ensembles of Support Vector Machines (SVM).

The experimental setup has been organized into two steps. First of all, we have selected the complex synthetic problem denoted P2 and two other multi-classes real-world problems, Satimage and Letter, from Satlog collection [79]. P2 [115] is a classification problem that

consists of two classes (I and II), where the decision region for each class is delimited by one, two, or even more than four equations and without overlapping between the distributions. More details about this synthetic problem can be found in the appendix I.

We summarize some information about the three classification problems used in Table 4.1. For the P2 problem, a large dataset was generated and splited into a small training set and a large testing set composed of 100 and 10,000 samples respectively. For the real-world problems, the same original distributions of samples for training and test sets were used.

Table 4.1 Information on the databases

Database	Number of Classes	Number of Features	Number of Training Samples	Number of Test Samples
P2	2	2	100	10,000
Satimage	6	36	3,104	1,331
Letter	26	16	10,500	4,500

Thereafter, ensembles of SVMs with RBF-kernel varying the C and γ parameters were built based on the Bagging method [5]. Therefore, ensemble members were created by taking random samples with replacement from a given original training set D, and by building them on different bootstrapped subsets. The total number of 50 classifiers was generated for each problem.

For each test sample $\mathbf{x}$, the final classification decision was made by taking the majority vote over the class labels produced by each ensemble member. A SVM one-against-one strategy was employed when dealing with the multi-class problems. Moreover, a RBF kernel was used because it nonlinearly maps samples into a higher dimensional space. Furthermore, this kernel has also obtained superior power of generalization and lower complexity than the Polynomial kernel [115], for example. The variations of the C and γ parameters were done based on these

values:

$$\begin{cases} \gamma \in & \{10000, 2500, 100, 25, 4, 1, 0.25, 0.04, 0.01, 25e03, \\ & 4e04, 1e04, 25e05, 11e05, 6e06, 4e06, 1e06\} \\ C \in & \{0.01, 0.1, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000\} \end{cases}$$

Therefore, 204 different combinations of models were trained and evaluated on each subset of data totalizing more than 30,600 different RBF-SVMs for all databases. Finally, the measures introduced in Sections 4.2 and 4.3 were evaluated over the ensembles generated and compared with their average loss and generalization errors. The average loss of predictions is computed between base classifiers outputs $\hat{y_j}$ and a true class y_i^*. In particular, it represents the mean error rate between the ensemble members as defined in Equation 4.21.

$$A.Loss = \frac{1}{nL} \sum_{i=1}^{n} \sum_{j=1}^{L} [\hat{y}_{ij} \neq y_i^*] \tag{4.21}$$

While the generalization error of the ensembles is computed according to Equation 4.22, where y_m denotes the majority vote. It corresponds to the actual error of the ensemble after combining the base classifiers.

$$G.Error = \frac{1}{N} \sum_{i=1}^{N} [y_{im} \neq y_i^*] \tag{4.22}$$

4.4.1 Obtained Results

From the obtained results we could observe very interesting relationships among diversity measures, margin theory, and majority vote accuracy. In order to better examine them, we start by analyzing the best results for each measure previously mentioned regarding each theory and classification problem tested. The results are reported in Tables 4.2, 4.3, and 4.4. In these tables, we can see the optimum value reached for each measure, their corresponding ensemble configuration, and generalization power yielded, i.e. in terms of individual errors (i.e. average loss) and generalization error after combination.

In Figures 4.1, 4.2, and 4.4, we can also observe the behavior of all measures concerning two different perspectives of the ensembles with the best performances, i.e. with the lowest generalization error. In the first one, results of different ensembles are plotted by fixing the value of the parameter C and varying the parameter γ. By contrast, in a second perspective, the parameter γ is fixed while C is varied. Based on all this information, we examine each experimental result with respect to the majority vote accuracy, and finally discuss details on their use as objective function for ensemble selection. This analysis is described in the next sections.

4.4.1.1 Diversity results

The results have shown that diversity is very important for accuracy of EoCs, since ensembles with the lowest average loss of predictions between their members have not reached the lowest generalization error. This can be seen in all Tables 4.2-4.4 and Figures 4.1-4.5.

For example, in Table 4.3, the ensemble composed of the highest performing classifiers, i.e. with parameters $C = 5$ and $\gamma = 1$, did not produce the most performing combination, which was obtained when $C = 20$ and same γ value. It means that individual performances of members are one factor that contributes to the overall ensemble performances, but they are not sufficient. Thus, some diversity is requested to get the highest majority vote performances, as also pointed out in [66].

However, as we have mentioned before, the relationship between diversity and ensemble accuracy may be very complex. In fact, we could see that the results for some diversity measures were more ambiguous in relation to the ensemble accuracy. This is because, for several ensembles, they have assumed the same values, even if the ensembles had different average loss (i.e. mean error rates) or generalization error (i.e. majority vote error). As examples, we can relate mainly those focused on the increasing of the variance between the base classifiers outputs, such as: Q average, Disagreement, Ambiguity, and Kohavi-Wolpert variance measures, as shown in Figures 4.1 - 4.5. Above all, these results revealed that the diversity measures can be categorize into two groups according to their relationship with ensemble accuracy. In

Table 4.2 Best results obtained for each measure evaluated on the P2 database

Measures	C	γ	**Value**	**Average Loss(%)**	**Generalization Error (%)**
Average Loss (↓)	2	100	0.1719	17.19	12.78
Generalization Error (↓)	1	100	0.1274	17.59	12.74
Difficulty (↓)	0.1	25	0	35.06	28.91
Ambiguity (↑)	0.1	25	0.2563	35.06	28.91
Double Fault (↓)	2	100	0.1006	17.19	12.78
Disagreement (↑)	0.1	25	0.3508	35.06	28.91
Kohavi-Wolpert (↑)	0.1	25	0.1719	35.06	28.91
Generalized Diversity (↑)	0.1	25	0.5003	35.06	28.91
Q Average (↓)	0.1	25	0.3100	35.06	28.91
Minimum Margin (↑)	0.1	25	0	35.06	28.91
Average Margin (↑)	2	100	0.6561	17.19	12.78
CI (↓)	2	100	0.6734	17.19	12.78

Table 4.3 Best results obtained for each measure evaluated on the Satimage database

Measures	C	γ	**Value**	**Average Loss(%)**	**Generalization Error (%)**
Average Loss	5	1	0.1091	10.91	9.92
Generalization Error	20	1	0.0969	11.06	9.69
Difficulty (↓)	0.1	0.25	0.1586	15.37	15.10
Ambiguity (↑)	200	$6e06$	0.0800	32.94	29.30
Double Fault (↓)	50	1	0.0816	11.08	9.77
Disagreement (↑)	1000	0.25	0.0787	12.37	10.59
Kohavi-Wolpert (↑)	1000	0.25	0.0386	12.37	10.59
Generalized Diversity (↑)	1000	0.25	0.3181	12.37	10.59
Q Average (↓)	1000	0.25	0.9568	12.37	10.59
Minimum Margin (sum rule) (↑)	50	$1e04$	0.2000	26.59	26.52
Average Margin (sum rule) (↑)	5	1	0.7818	10.91	9.92
CI (sum rule) (↓)	50	1	0.4622	11.08	9.77
Minimum Margin (max. rule) (↑)	50	0.01	0.2000	26.59	26.52
Average Margin (max. rule) (↑)	5	1	0.7853	10.91	9.92
CI (max. rule) (↓)	50	1	0.4442	11.08	9.77

the first one, we can group diversity measures that were “weakly" related, such as: Qaverage, Disagreement, Ambiguity, and Kohavi-Wolpert variance measures. On the other hand, Gen-

Table 4.4 Best results obtained for each measure evaluated on the Letter database

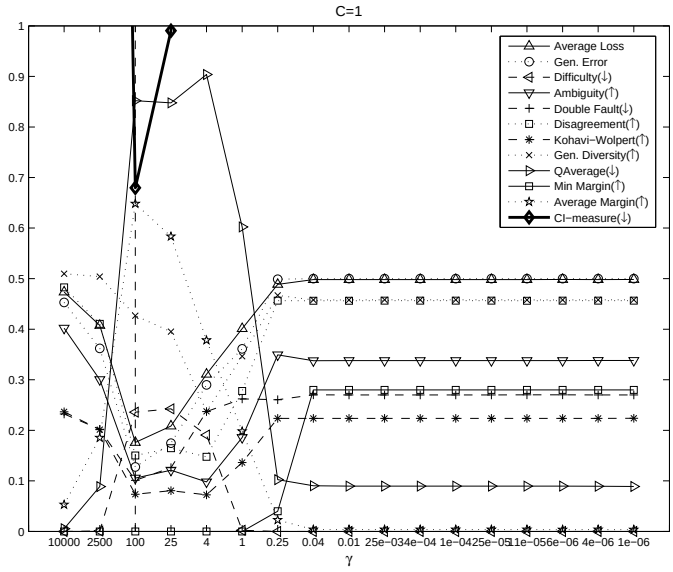
Measures	C	γ	**Value**	**Average Loss(%)**	**Generalization Error (%)**
Average Loss	10	1	0.0456	4.56	3.44
Generalization Error	20	1	0.0336	4.58	3.36
Difficulty (↓)	5	1	0.1298	4.71	3.80
Ambiguity (↑)	1	25	0.1685	29.94	24.93
Double Fault (↓)	20	1	0.0275	4.58	3.36
Disagreement (↑)	1	25	0.1247	29.94	24.93
Kohavi-Wolpert (↑)	1	25	0.0611	29.94	24.93
Generalized Diversity (↑)	500	0.25	0.4063	6.02	4.31
Q Average (↓)	2	25	0.9526	28.79	23.78
Minimum Margin (sum rule) (↑)	10	4	-0.2000	4.77	3.47
Average Margin (sum rule)(↑)	10	1	0.9088	4.56	3.44
CI (sum rule) (↓)	20	1	0.1247	4.58	3.36
Minimum Margin (max. rule)(↑)	20	0.25	0	4.69	3.45
Average Margin (max. rule)(↑)	10	1	0.9142	4.56	3.44
CI (max. rule)(↓)	20	1	0.1086	4.58	3.36

eralized Diversity, Difficulty, and Double-Fault measures belong to the second group denoted as “strongly" related. Yet concerning this last group, Double-Fault measure was more related to the ensemble accuracy, followed by the Difficulty and Generalize Diversity measures which were slightly less correlated to the ensemble errors. Similar conclusions about the behaviors of the Double-Fault and Difficulty measures have been also outlined in [69].

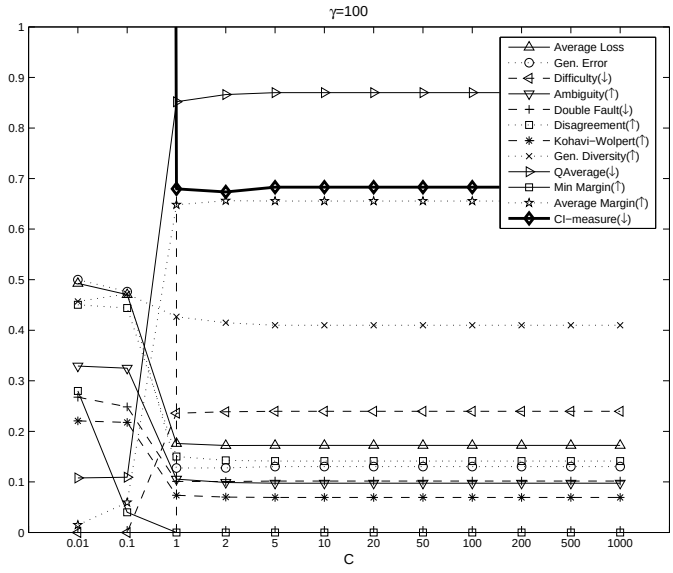
4.4.1.2 Margin results

In particular, we have evaluated the main measures provided by the margin theory: minimum margin, cumulative margins distributions, average margin, and CI. Based on the results, we could observe some interesting insights on this theory and the majority vote accuracy. For instance, in the literature, the maximization of margins on training data is commonly pointed out as responsible for decreasing the generalization error on future test sets [99].

So, in a first moment, we would expect that maximizing the minimum margin for ensembles should be accompanied with the minimum generalization error. However, the fact is that the



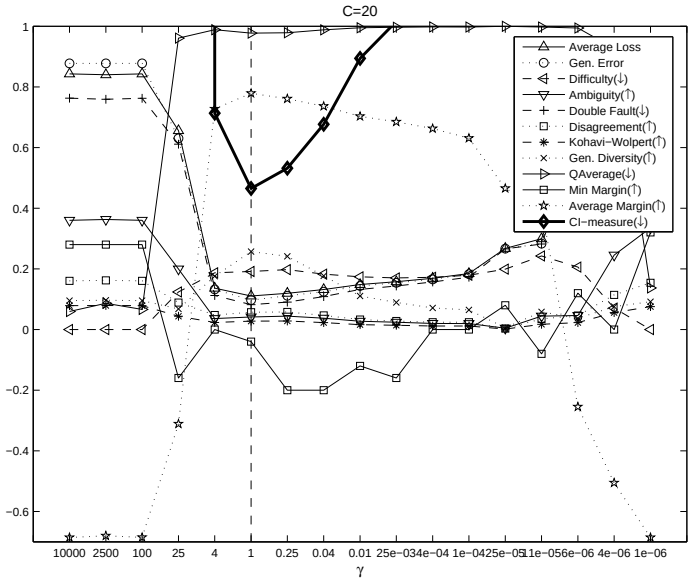
(a) P2 - fixed C



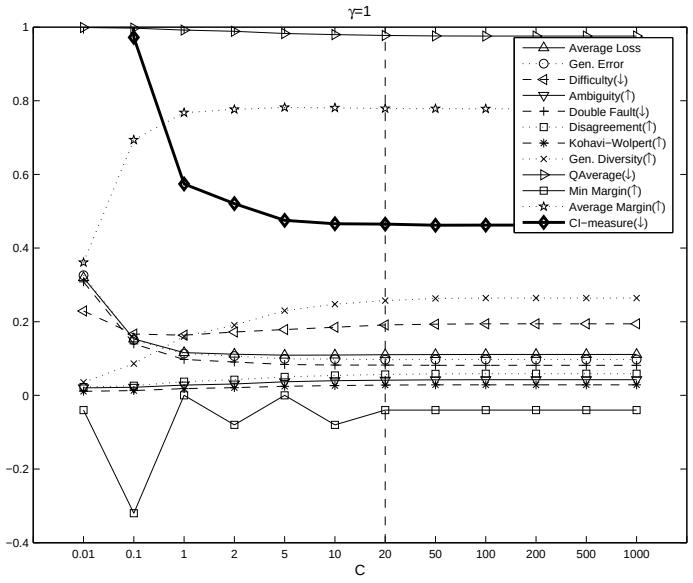
(b) P2 - fixed γ

Figure 4.1 Results for ensembles with the best combinations of C and γ parameters on two different perspectives over the P2 database. (a) Ensembles with the best C value fixed and varying γ, and vice-versa in (b). The vertical dashed lines indicate where the minimal generalization error was attained.

minimum margin measure have shown great instability. This is because, as it can be seen in Figure 4.2 (a), the tracking of the maximum minimum margin can be quite difficult, since

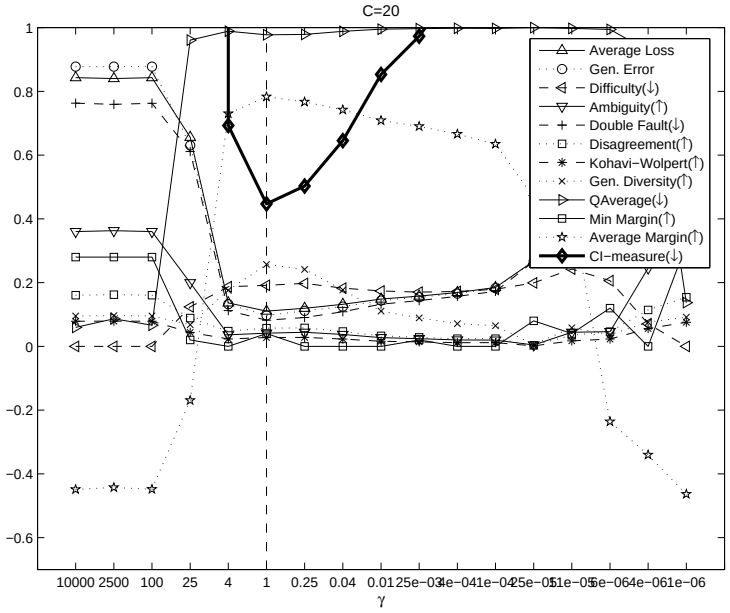


(a) Satimage - fixed C

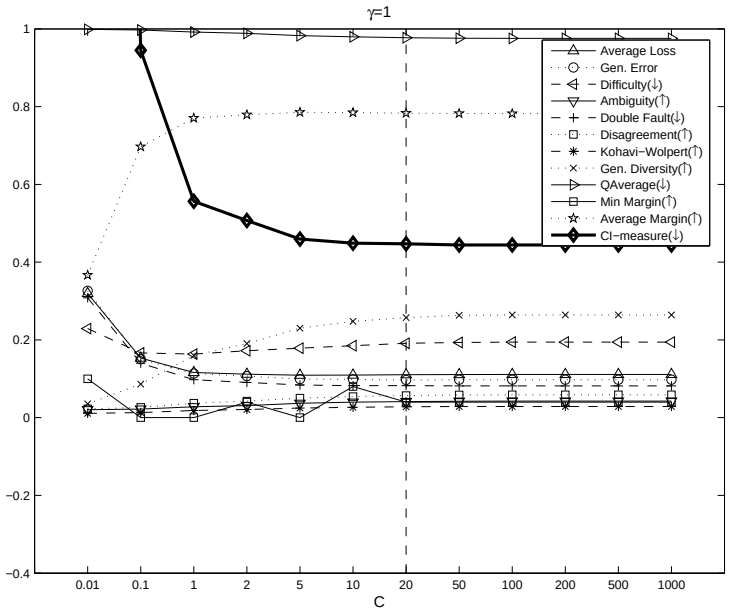


(b) Satimage - fixed γ

Figure 4.2 Some results obtained for ensembles with the best combinations of C and γ parameters on two different perspectives over the Satimage database. (a) results for ensembles with the best combination by fixing and varying the C and γ parameters, respectively. (b) results obtained by fixing the best γ parameter found and varying C. The vertical dashed lines indicate where the minimal generalization error was attained.

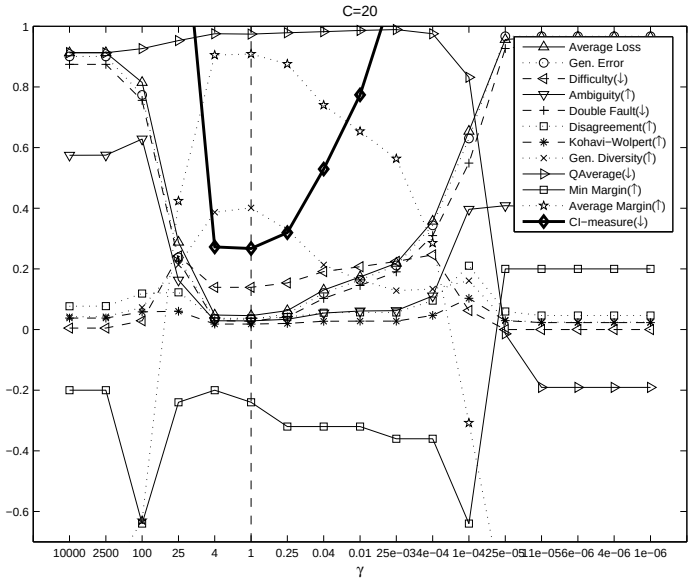


(a) Satimage - fixed C

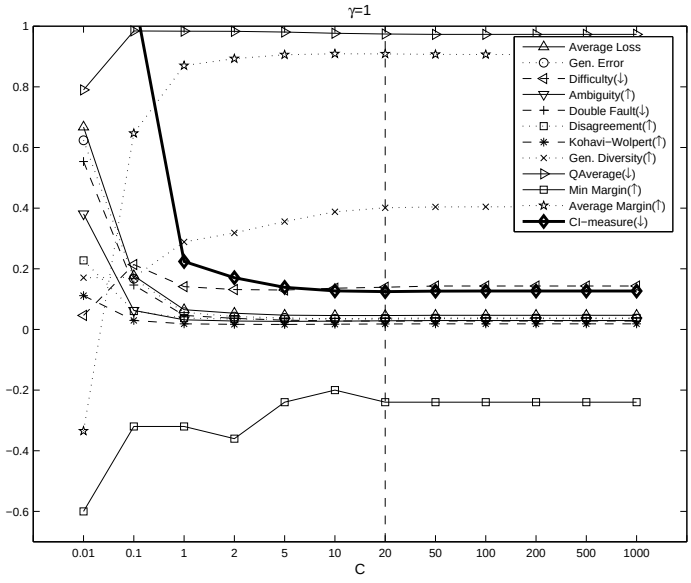


(b) Satimage - fixed γ

Figure 4.3 Similar results to that depicted in Figure 4.2 but with the second margin definition (Equation 4.17), i.e. with the max. rule. Vertical dashed lines point out the region in which the optimum generalization error was achieved.

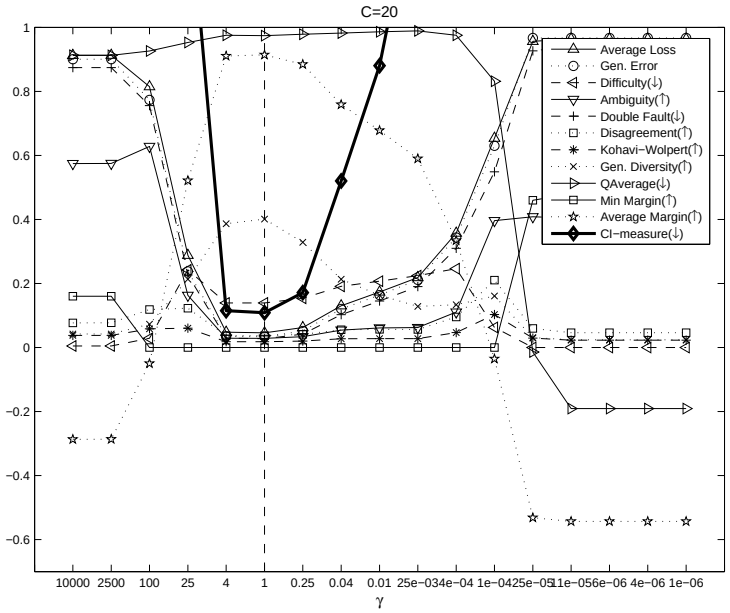


(a) Letter - fixed C

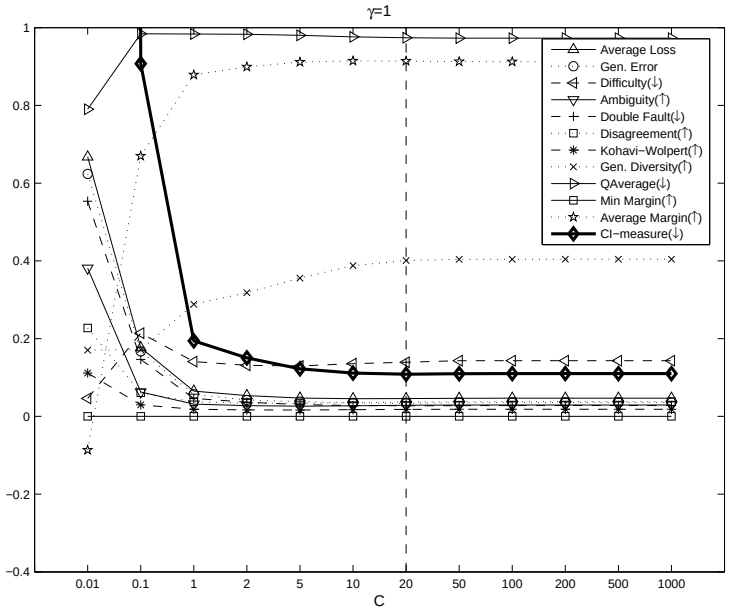


(b) Letter - fixed γ

Figure 4.4 Some results obtained for ensembles with the best combinations of C and γ parameters on two different perspectives over the Letter database. (a) results for ensembles with the best combination by fixing and varying the C and γ parameters, respectively. (b) results obtained by fixing the best γ parameter found and varying C. The vertical dashed lines outline where the minimal generalization error was achieved.



(a) Letter - fixed C



(b) Letter - fixed γ

Figure 4.5 Similar results to those shown in Figure 4.4 but now with the second margin definition denoted by Equation 4.17. The vertical dashed lines indicate where the minimal generalization error was attained.

many different values can be achieved even around the best ensemble. In light of this, the greed maximization of the minimum margin may not be satisfactory for searching the best ensembles.

On the other hand, the results reported with cumulative margins distributions have shown us such relation. As examples, we have plotted some results involving high and less performing ensembles over the Satimage problem in Figure 4.6. From these resuts, it can be seen that the ensembles with the best performances (i.e. composed of the parameters $C = 20$ and $\gamma = 1$ and $C = 50$ and $\gamma = 1$), have actually reached larger margins than ensembles with lower performances (e.g with $C = 1000$ and $\gamma = 0.25$ pointed out by some diversity measures), since their margin values are more concentrated at the maximum value (i.e. around 1, which produces the lowest curves).

In addition, the results with the average margin measure have also demonstrated that classifier ensembles with large margin values are very performing. In fact, we have observed that this measure is very stable. Thus, we can particularly assert that is more relevant to concentrate on the average margin than only on the minimum one.

However, although the average margin over test instances represents an estimate of expected margin for a classification problem [111], after an analysis of the results, it is clear that this measure is strictly related to the average loss (mean error rate) of the base classifiers composing an ensemble and not exactly to its generalization error. To illustrate this, we can see that the maximum values of average margin correspond to the minimum values of average loss in Tables 4.2, 4.3, and 4.4, and also in Figures 4.1, 4.2, and 4.4.

Therefore, maximizing the average margin points out the ensembles composed of the strongest individual members in a given pool. In general, this fact is not much interesting because there is a great tendency that in a limit of the highest possible individual performances, the base classifiers will be very similar, with so low diversity that their team may not reach the maximum majority vote accuracy. As a consequence, despite of most of the times the maximum values of average margins accompany the minimum values of generalization error for some ensembles, usually those with the maximum average margin and minimum generalization error (majority

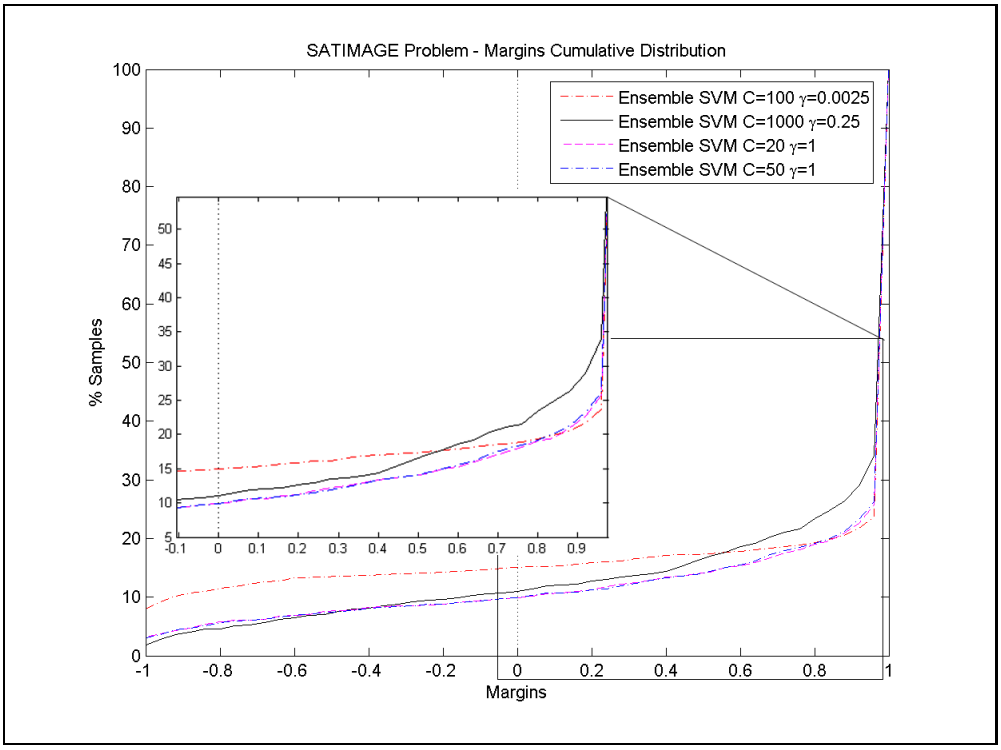
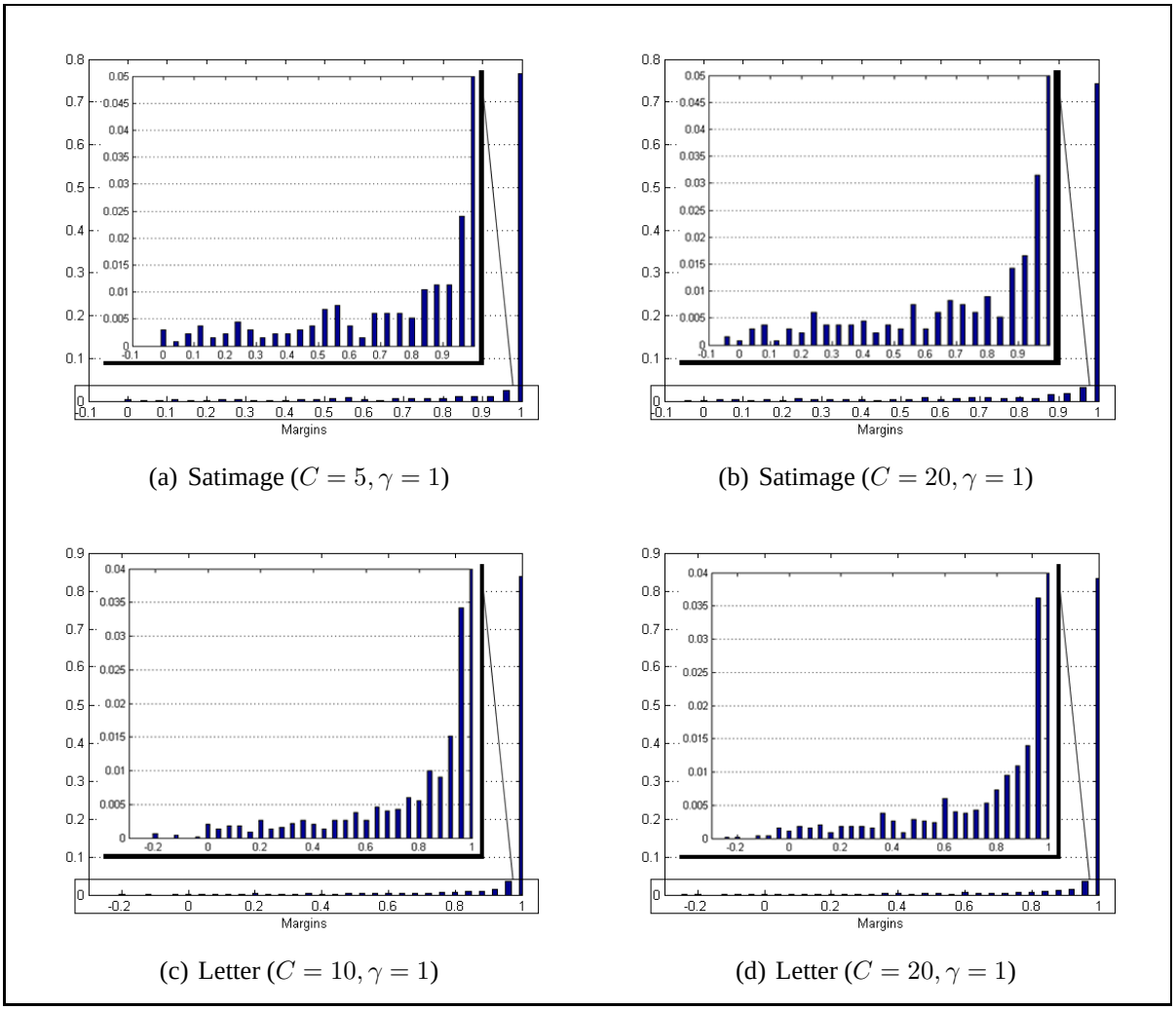


Figure 4.6 Some cumulative margins distributions computed on the Satimage problem.

vote error) in the extreme cases may slight diverge. It can be seen in all results listed in Tables 4.2, 4.3, and 4.4).

Taking this into account, based on two case studies, we have examined more carefully the relationship between the expected generalization error rate and the margins of the ensembles with the lowest average loss and generalization error, respectively. In order to achieve this analysis, we compare their histograms formed by frequencies of margins defined by Equation 4.16 computed for all samples in the test set. They are depicted in Figures 4.7 (a)-(d) for the Satimage and Letter problems.

Based on these results, it is possible to observe that ensembles with the lowest generalization error (Figures 4.7 (b) and (d)) have obtained margins with more plurality of values than those ensembles with the lowest average loss (Figures 4.7 (a) and (c)). Thus, it is clear that while ensembles with very performing members reach high values of margins, ensembles with the lowest generalization error obtain margins relatively high, but also tend to produce values more



(a) Satimage ($C = 5, \gamma = 1$)

(b) Satimage ($C = 20, \gamma = 1$)

(c) Letter ($C = 10, \gamma = 1$)

(d) Letter ($C = 20, \gamma = 1$)

Figure 4.7 Histograms of the margins frequencies from ensembles with the largest average margin ((a) and (c)), and with the lowest generalization error ((b) and (d)) from Tables 4.3 and 4.4 for the Satimage and Letter problems, respectively.

varied. These results have demonstrated how important is a balance between the increasing of the margins accompanied of some variance between the ensemble members. This explains why the results obtained with the CI-measure were the most correlated with the ensemble accuracy regarding all measures tested. Now, after reporting the evaluation of all measures and relating their results to the ensemble accuracies, we present a discussion on the relationship between these two theories and their application as objective function to ensemble selection processes.

4.5 Discussion

In this chapter we have tested various measures for the evaluation of classifier ensembles. In particular, we could observe that the most appropriate diversity measures to evaluate or select ensembles are: Generalized Diversity, Difficulty, and Double-Fault measures. The other measures that regard only the variance of the outputs and not the individual members performances, such as Disagreement, Ambiguity, Kohavi-Wolpert, etc have been proved to be inadequate for such tasks. Thus, we can assert that the relationship between most of diversity measures and accuracy is not so strong. This fact explains why seeking diversity explicitly may be ineffective to point out ensembles with optimal generalization performance. Besides, it confirms the Accuracy-Diversity dilemma, which states that highly accurate classifiers cannot be very diverse [69]. In other words, it means that the base classifiers are strong, but also with some variance among them.

On the other hand, we could observe that only the increasing of the margins over a dataset may be an interesting option for selecting classifier ensembles. In contrast, the minimum margin measure seems not to be stable, and average margins indicated just ensembles composed of the strongest individual classifiers, but not with the best answers combined.

By analyzing the results we have also seen that the diversity measure Double-Fault and the margin-based measure CI-measure were the two measures more related to the generalization error over all the problems. From this point of view, the relationship between the diversity and margin theories becomes strong. This is because, the generalization error can be well estimated by the combination of high performing base classifiers (i.e. with high average margins) and a relative diversity between them.

Taking this into account, both Double-Fault and CI-measure seem promising to be used as objective functions for the selection of classifier ensembles. This is probably because strong classifiers were available and both measures tries to decrease the probability of identical errors. However, as Double-Fault is a pair-wise measure, the cardinality of the final ensemble selected must be specified in advance. Otherwise, the resulting ensemble will always have the

minimum number of classifiers, i.e. 2. On the other hand, the CI-measure does not share the same problem. The boundary provided by the CI-measure seems to be a good measure for the selection of ensembles. Besides, it has the advantage that the balance between accuracy and diversity is explicit: while the average margin is related to the strength of the base classifiers, the variance of the margins can be seen as diversity represented by the variance between the base classifiers. In light of all these results, and based on experimental results presented in the appendix IV, we have decided to employ this measure as part of our decision module responsible for the selection of ensembles, which is described with our framework presented in next chapter.

CHAPTER 5

A DYNAMIC OPTIMIZATION APPROACH FOR ADAPTIVE INCREMENTAL LEARNING IN STATIC ENVIRONMENTS

In the previous chapters we have studied important aspects in order to develop an adaptive classification system. Regarding the former, we have seen the importance of well tuning and updating the parameters of classifiers overtime, since they can vary depending on the data available. So, the aim was at developing a method able to search for optimum parameters values, and at the same time efficient to adapt new solution if needed. Then, considering that the use of ensemble of classifiers can overperform single models, especially when its members are selected and the level of uncertanity is high, we have investigated several measures to evaluate and select ensemble. The results showed that measures based on the margin theory are promising to deal and select ensembles, once they regard directly the degrees of confidence of classifiers.

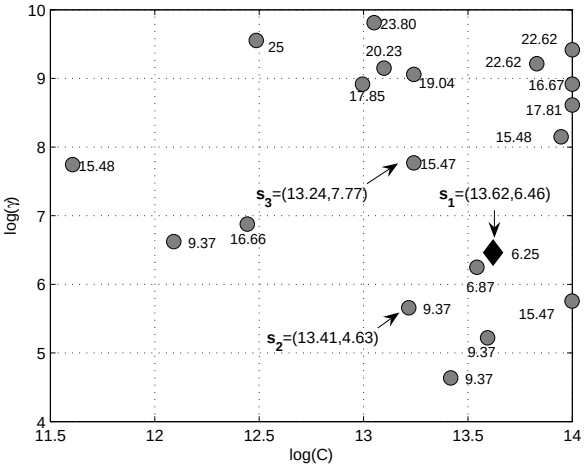
From these standing points, in this chapter we propose a method to perform adaptive incremental learning based on these two principles: (1) to incrementally accommodate new data by updating models, and (2) to dynamically track new optimum system's parameters for self-adaptation. Thus, the underlying hypothesis herein is also to consider the incremental learning process as being a dynamic optimization process, in which optimum hypotheses are dynamically tracked, evolved, and combined overtime. Likewise, we have achieved with the SVM model selection processes carried out overtime in a gradual learning scenario.

In particular, the proposed method relies on a new framework incorporating different 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) overtime to maintain the system's optimality w.r.t. internal parameters, computational cost, and generalization perfor-

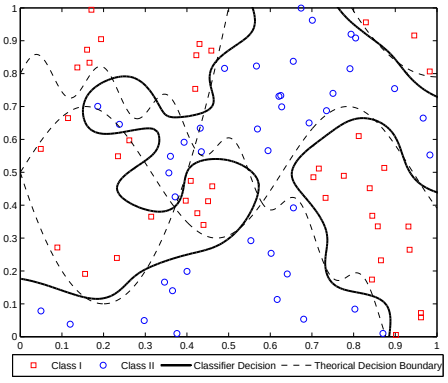
mance. As mentioned before, the use of ISVM ensembles in this study is justified based on two main evidences found in literature. First, as the classification success of SVMs does not depend on the dimensions of the input space, SVM is a robust classifier against the well known *curse of dimensionality* mainly involving small data sets. Therefore, it is very advantageous for incremental learning situations. Second, SVMs ensembles are employed because they can often overcome single models' performances, especially when heterogeneous (in terms of hyperparameters values) base classifiers are used and the level of uncertainty is high, i.e. when only small sample sets are available [116]. We illustrate this concept with an example in Figure 5.1, which shows that three different optimized solutions, i.e. $\mathbf{s}_1$, $\mathbf{s}_2$, and $\mathbf{s}_3$ can produce different classifiers' decision boundaries in (b), (c), and (d) based on a same small training set of 84 samples. Because of this, the use of multiple solutions is very interesting, since each optimized solution may represent the same problem in different ways. Eventually, the proposed framework provides contributions on strategies to optimize and overproduce classifiers, as well as the application of memory-based mechanisms for solving dynamic optimization processes. This latter is a promising and ongoing research area [37].

In addition, we validate the proposed method and show its efficiency through experiments with synthetic and real-world databases. Results in single and multiple classifiers configurations are compared with those obtained with these strategies: SVM optimized with PSO in batch mode, incremental SVM with parameter values beforehand fixed, two incremental capable classifiers (1-NN and Naïve Bayes) widely applied in incremental learning studies. These classifiers are tested because their performances are considered "no-less" with respect to their batch versions [87]. An incremental ensemble strategy with optimized parameters and different combination rules is also employed for comparisons.

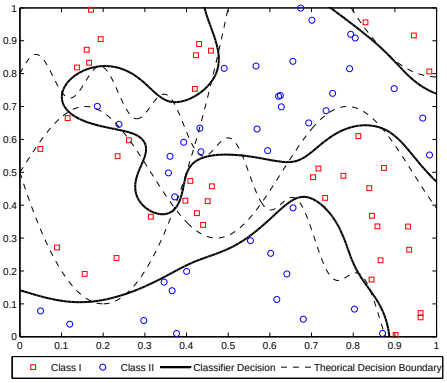
As additional purposes, we try to verify if (1) incremental learning with SVM can achieve similar performances to those obtained in batch mode, (2) the adaptation of system's parameters over time is actually a dynamic optimization problem and hence important to achieve high performances, (3) the dynamic selection of ensemble can lead to better results than by simply combining all pool of classifiers available.



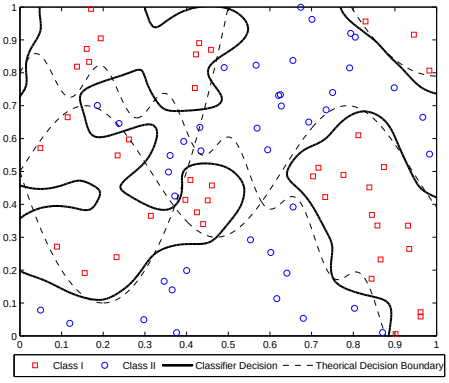
(a) Swarm



(b) $\mathbf{s_1}$



(c) $\mathbf{s_2}$



(d) $\mathbf{s_3}$

Figure 5.1 Examples of different classifiers' decision boundaries in (b), (c), and (d) trained from three optimized solutions, i.e. s_1, s_2, and s_3 in (a) on the same small training set of 84 samples.

The remaining of this chapter is organized as follows. In section 5.1 we introduce the proposed method for adaptive incremental learning. Experimental results and discussions are reported in sections 5.2 and 5.3, respectively.

5.1 The Proposed Approach

So far, we have seen that, traditionally, researches on incremental learning regard the classifiers' parameters setting as a static process, i.e., parameters values are initially set (e.g. based on standard values or estimated over the first datachunk available), and kept infinitely fixed. However, optimum hyper-parameters values may shift over the search space during the evolution of the data. As a consequence, classifiers with obsolete internal parameters (mainly those related to regularization) will disturb and ruin the system's updating in terms of generalization power and complexity of models.

The proposed method herein for adaptive incremental learning optimizes, selects, and combines incremental SVM classifiers overtime. More specifically, it is designed to dynamically point out optimum solutions for sequences of datasets $\mathcal{D}(k)$ by using the best solutions found so far, or by starting new dynamic optimization processes. As we employ incremental support vectors machines as our base classifiers and dynamic particle swarm optimization for searching optimum hyper-parameter values, each solution $\mathbf{s}$ represents a particle codifying an SVM hyper-parameter set, e.g. $\{C, \gamma\}$. Change detection mechanisms monitors 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 $\mathcal{C}^*$. Details on the framework of proposed approach are described below.

5.1.1 Framework for Adaptive Incremental Learning (AIL)

Our framework for adaptive incremental learning is composed of five main modules, as shown in Figure 5.2 and listed in Algorithm 4: change detection, adapted grid-search, dynamic particle swarm optimization (DPSO), incremental support vector machines, and decision fusion. In particular, this framework represents many upgrades in relation to our first version introduced

in [57], such as the use of incremental classifiers, dynamic selection and building of ensembles from optimized models. Below, details on each module are provided.

The *upgrade_stm* and *recall_stm* functions are respectively 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^* means support vectors obtained from the final incremental model $\mathcal{M}^*$ trained with hyper-parameters found by 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 sake of simplicity, in the equations, $\mathcal{D}(k)$ represents the merge of 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).

Algorithm 4 Adaptive Incremental Learning (AIL)

Input: A training set of data $\mathcal{D}(k)$.
Output: Optimized SVM classifier/ensemble.
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/update/combine the final incremental SVM classifiers from incoming data $\mathcal{D}(k)$, $\Delta(k)$, and SV.

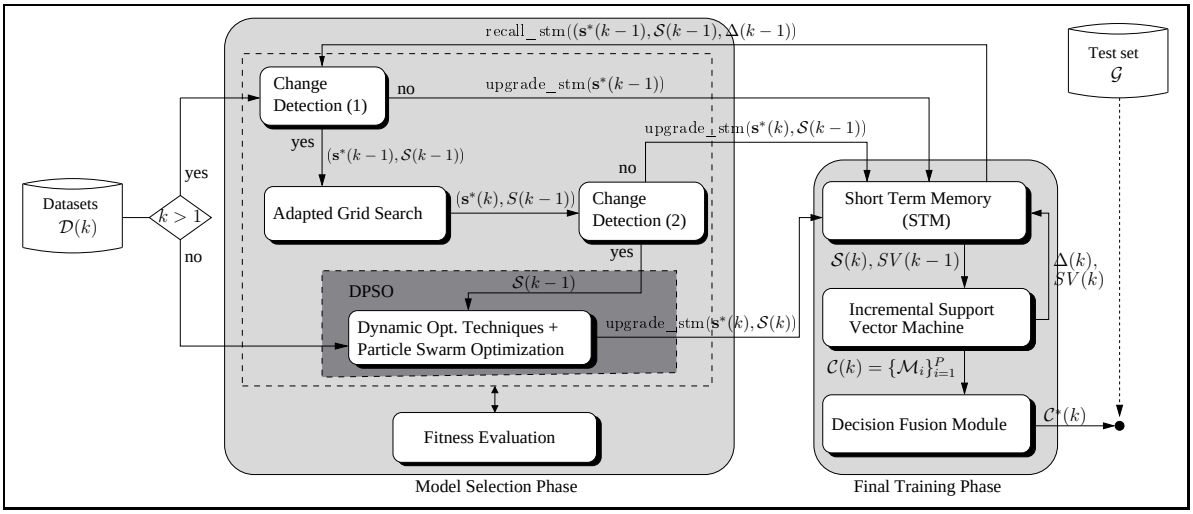


Figure 5.2 General framework of the proposed method for incremental learning with dynamic SVM model selection. Δ represents a set of data sv^* composed of support vectors and relevant samples rs selected during the training of final model $\mathcal{M}$ from best particle s^*. So $\Delta = \{sv^* \cup rs\}$, where sv^* means support vectors obtained specifically from final model $\mathcal{M}$ trained with hyper-parameters found by best particle 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 $\mathcal{S}(k-1)$.

5.1.2 Additional modules

As this new framework is built based on similar components already introduced in our first framework presented in chapter 3, for sake of simplicity, in this chapter we outline only the major modifications added to its original version. Such modifications are mainly related to the creation of two modules: one for incremental learning with support vector machines and another to fusion and select classifiers into optimized ensembles. They are both decribed below.

5.1.2.1 Incremental Support Vector Machine Module

In this thesis, we implement an incremental SVM version based on the Syed et al. method [109] due to three reasons: (1) it focus on the updating of models over sequences of datasets overtime, (2) this method has produced the best results in this comparative study [35], and (3) it does not require the tuning of extra-parameters, which may need a careful setting, as it occurs in [80, 95, 91, 35, 94, 1].

This latter is important because the setting of extra parameters can be very critical. It is because they control when samples might be either exchanged among temporary sets or when learning processes should stop. Moreover, the SVM implementation used in here [18] already provides mechanisms to accelerate the SVM training through the Sequential Minimal Optimization (SMO) technique. Therefore, it demands less computational efforts than traditional quadratic programming solvers, as shown in [92].

Like in [109], 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 a previous learning at a given time k. However, unlike in [109] where only support vectors are stored, our incremental SVM module also retains additional training samples relying in a “relevant region" which exceeds the SVM margins in half of their sizes.

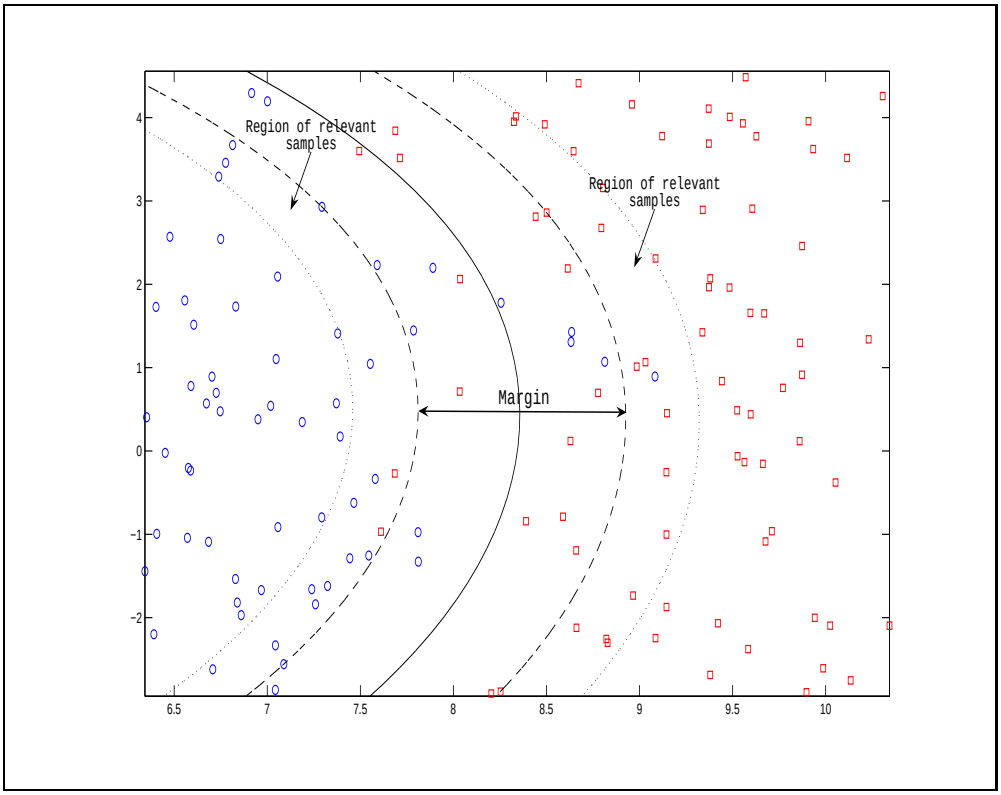


Figure 5.3 Example of regions defined around the SVM margin separating two classes (circles and squares) in which relevant samples are selected from.

It is interesting to note that, even if we fix this region as being half of the margins, the size of this region varies according to difficulties of classification problems (e.g. complex decision boundaries, overlapping between classes, etc.) and hyper-parameters selected. Although the storage of additional samples is not a desirable property in incremental learning algorithms [93], it is necessary because these additional samples can become support vectors during optimizations of SVM hyper-parameters in the future.

Algorithm 5 The incremental SVM module

1: **Input:** Current datachunk $\mathcal{D}(k)$, relevant samples $rs(k-1)$, model $\mathcal{M}_i(k-1)$.
2: **Output:** SVM model $\mathcal{M}_i(k-1)$ updated.
3: sv =selected_support_vector($\mathcal{M}_i(k-1)$)
4: $working\_set = \mathcal{D}(k) \cup sv(k-1) \cup rs(k-1)$
5: $\mathcal{M}_i(k-1)$ =train_svm($working\_set$)

5.1.2.2 Decision Fusion Module

The decision fusion module dynamically selects, and combines incremental classifiers into ensembles. Our dynamic selection strategy is implemented based on a generalization bound introduced in [8], which we first studied its application for “static" SVM ensembles in [54].

In this dynamic strategy, only classifiers whose combination minimizes this bound (called here CI measure) are selected to compose the final ensemble. In particular, this measure is computed as $CI = \sigma(\tau)/\mu(\tau)^2$, where σ and μ denotes 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. $\tau_i = \vartheta_t(\mathbf{x}_i) - \max_{\substack{j=1,\ldots,c \\ j \neq t}}\{\vartheta_j(\mathbf{x}_i)\}$. In 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 of all, the pool of classifiers $\mathcal{C}(k)$ generated from $\mathcal{S}(k)$ are sorted according to their respective individual confidence levels (average margins). Then, the selection process starts by adding a classifier at time until reach the maximum number of classifiers, i.e. number of particles P. Each time a classifier is added, the CI selection criterion is recomputed. The best ensemble selected $\mathcal{C}^*$ is that whose CI value is minimal.

Thus, the key idea is to select the ensemble with the strongest, i.e. the highest confidences, and less correlated classifiers over the current training set. Finally, once the best ensemble $\mathcal{C}^*$ is selected, they are combined using weighted average voting based on classifiers' performances. Although with different criteria, forward searches for best ensembles seem to be very promising [114].

In order to calibrate the outputs of the SVM in estimates of probabilities, we use the approach introduced by Wu et al [39], which is implemented in the LIBSVM software [18]. In such approach, given k classes of data, for any x, the goal is to estimate $p_i = p(y = i|x), i = 1, \ldots, k$. The estimated pairwise class probabilities for multi-class classification is defined as $r_{i,j} \approx p(y = i|y = i \; or \; j, x)$, that is, using the implementation of Lin et al. [76]:

$$r_{i,j} \approx \frac{1}{1 + e^{A\hat{f}+B}}, \tag{5.1}$$

where A and B are estimated by minimizing the negative log-likelihood function using known training data and their decision values $\hat{f}$. The p_i from all $r_{i,j}$ is obtained by solving:

$$min_p \tfrac{1}{2} \sum_{i=1}^{k} \sum_{j:j \neq i} (r_{j,i}p_i - r_{i,j}p_j)^2 \quad \text{subject to } \sum_{i=1}^{k} p_i = 1, \, p_i \geq 0, \, \forall i \tag{5.2}$$

Based on this framework, therefore, the proposed method is capable of evolving and accommodating new data by automatically selecting internal hyper-parameters, updating, and combining incremental SVM classifiers. The experimental protocol and results obtained are described in next section.

5.2 Experimental Protocol

In order to validate the concept of adaptive incremental learning system as well as to show efficiency of proposed method, the following experimental protocol has been carried out. First of all, to characterize with more impact the occurrence of population drifts, the original training sets were divided into small datasets. The total number of datasets and their sizes were determined based on a minimum amount of samples required for each class, which was set to at least 16. The distribution of samples were firstly separated for the class with the minor number of samples, and then proportionally for the other classes. Such procedure determined the total number of chunks. Thus, the same original proportion of samples per class was kept in each datachunk. In other words, it means that if the original problem contains unbalanced classes, this same real scenario is simulated in this experimental protocol.

Therefore, as in most of the incremental learning approaches, this experimental protocol focused on incremental learning from datachunks with suitable-size of samples at time, i.e. block by block, and not one sample at a time, which is called online, or instance by instance learning [109]. A detailed description of the datasets and number of chunks used are listed in Table 5.1. We have employed classification problems with different number of features, classes, training and testing samples. As the proposed method uses a stochastic algorithm, the results represent averages drawn over 10 replications.

Table 5.1 Specifications on the datasets 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	4,435	2,000

5.2.1 Strategies Tested

The following incremental learning strategies were tested:

5.2.1.1 Batch SVM-PSO

In this strategy, the whole original training datasets are used for selecting of optimum SVM hyper-parameters and training the final model. The hyper-parameter 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 a batch strategy.

5.2.1.2 Incremental no-less classifiers (1-Nearest Neighbor (1-NN) and Naive Bayes (NB))

These two classifiers were tested because they are widely employed in the incremental learning/concept drift literatures [119, 30, 87], since they are considered *no less* incremental learners, i.e. their results in incremental mode are similar to those obtained in batch mode [87].

5.2.1.3 Incremental SVM (ISVM)

In this approach, an incremental SVM classifier tailored from [109] is updated from successive datachunks $\mathcal{D}(k)$. Its hyper-parameters are firstly 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 incremental learning process.

5.2.1.4 Optimized Random Aggregation (ORA-DMS)

This method represents a common incremental ensemble approach, more specifically, a random aggregation approach as described in section 2.4. In here, it consists of combining SVM classifiers with optimum hyper-parameters values trained from independent datachunks in a serial way (i.e. one classifier by datachunk) [119]. Two combination rules were tested with this scheme: majority and simple average voting. We set the maximum ensemble size to 20. When the total number is reached, the oldest model is replaced for the new one.

5.2.1.5 Single Incremental SVM (IS-AIL)

This approach denotes the proposed method in single classifier mode (i.e. only the best solution found so far is used by the decision fusion module). In other words, when only one incremental SVM classifier and its respective hyper-parameters are updated from every datachunk $\mathcal{D}(k)$.

5.2.1.6 Incremental EoC-DMS Swarm-based (IEoC-AIL)

The proposed approach in EoC mode presented in section 5.1. Therefore, it is employed with its full capacity, i.e., dynamically updating, selecting, and combining the ISVMs into ensembles.

5.2.2 Experiments Parameters Setting

We have used these parameters setting:

- *Optimization Algorithms Parameters*: The maximum number of iterations and the swarm size was set to 100 and 20, respectively. The dimensions of the parameters (C and γ) search space, where the maximum and minimum values were set to $[2^{-6}, 2^{14}]$, $[2^{-15}, 2^{10}]$, respectively. The DPSO topology used was the $lbest$ with $\lambda = 3$. We also consider to stop the optimization if the best value of fitness does not improve over 10 consecutive iterations.

- *Objective Function*: Several objective functions have been proposed for searching for optimum SVM hyper-parameters, e.g. radius margin bound, span bound, etc. [20]. 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 ν-cross-validation procedure is a good option. A $\nu = 5$ is used here as suggested in [18]. The results for each strategy are presented in next section.

5.2.3 Obtained Results

The obtained results are presented in this section as follows. First of all, we examine the performance of each strategy tested by evaluating their generalization errors achieved on each database. Then, we analyze the data storage required and complexity of models generated. Finally, we discuss results related to the adaptation of hyperparameters and combination/selection of ensembles regarding different functions and methods.

5.2.3.1 Performance evaluation

The generalization errors achieved by each strategy tested are reported in Table 5.2. These results were tested with 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 and incremental learning strategy are shown in bold. Values underlined indicate when an incremental strategy was significantly better than the others.

By analyzing the results in this table, we can see 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 learning all datachunks. It occurs even if with its hyper-parameters were kept fixed with value found on $\mathcal{D}(1)$ (ISVM). Most importantly, we could observe the efficiency of the proposed method as well as 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. It shows the importance of the adaptation of hyper-parameters and of the use of relevant samples during the incremental learning process. Besides, it could be observed that the proposed method (IEoC-AIL) has achieved results similar to, and sometimes, even better than SVM-PSO in batch mode. The latter proves that the dynamic selection and combination of optimum solutions can actually improve the overall performance of the system. Figures 5.4 (a) and (b) illustrate these results with two case studies concerning these generalization error results with the most performing strategies during different incremental learning steps at times k.

Table 5.2 Mean and standard deviation of error rates obtained after learning from all subsets available. The best results concerning the incremental strategies are shown in bold. Values underlined indicate when an incremental strategy was significantly better than the others. Results were computed over mean values draw from 10 replications

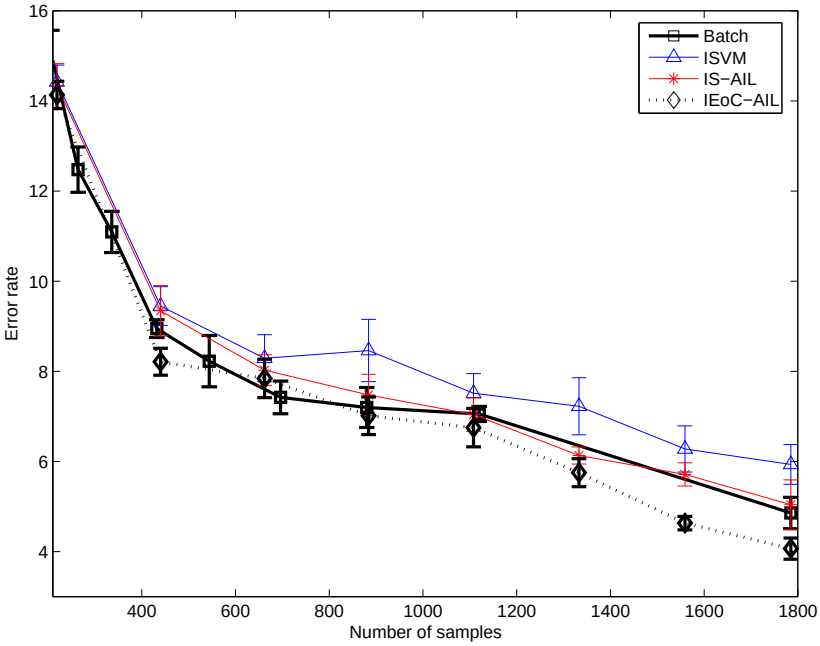
Databases	**Approaches tested**								
	SVM-PSO	SVM-PSO($\mathcal{D}(1)$)	1-NN	NB	ISVM	ORA-MV.	ORA-SA.	IS-AIL	IEoC-AIL
Adult	15.55 (0.06)	24.77 (0.80)	24.06	24.05	23.93 (0.50)	20.07 (1.46)	**19.29 (1.53)**	20.83 (4.34)	20.52 (1.60)
CiS	0.14 (0.03)	11.47 (0.42)	**1.02**	7.78	3.60 (0.97)	3.91 (0.63)	2.68 (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)**

Moreover, it could be seen that serial incremental ensemble approaches (i.e. the ORAs strategies, ORA-MV and ORA-SA) performed well especially on noisy data (e.g. as for the Adult database), although not statistically superior than the proposed method in these tests. By contrast, the need of setting a maximum number of classifiers is determinant for the performance of these methods, since some knowledge may be lost when the oldest classifier is replaced for a new one. This is a drawback, because the results with a single incremental learner (ISVM) were better than these two ensemble approaches for some problems (e.g. IR-Ship, German). These results indicate that the updating of an existing ISVM classifier might be very advantageous in relation to only combine 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 was superior than the majority vote rule. Eventually, the classical “non-less" incremental learners NB and 1-NN have achieved the worst performances. The only exception occurred for the CiS and P2 databases, where the 1-NN classifier outperformed the other methods tested, but of course, with the inconvenience of storing all data.

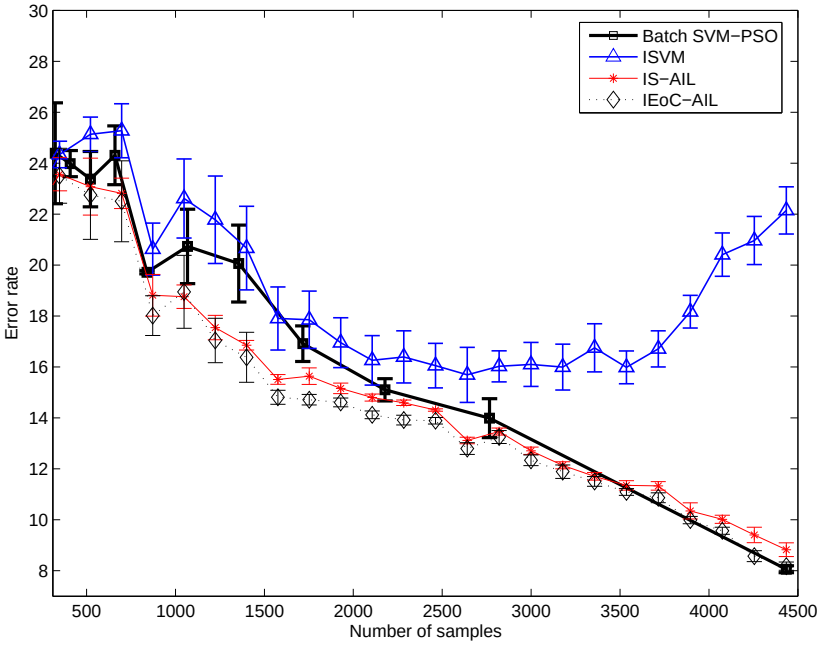
5.2.3.2 Data storage and complexity of models generated

Concerning now the complexity factor of the ISVM classifiers generated, Table 5.3 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 can notice that the dynamic adaptation of the hyper-parameters 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 than when the whole data are available for training.

In contrast, the incremental single SVM classifier strategy with constant hyper-parameters (ISVM) did not adjust its models as effect as the other SVM-PSO and IS-AIL strategies. Of course, these results are related to the single classifier strategies. On the other hand, the complexity of the ensemble version IEoC-AIL may be relatively higher, once the number of classifiers is dynamically selected between 1 and P. Thus, in spite of the fact that the proposed method (IEoC-AIL) supplied remarkable improvements in terms of generalization power, it can also turn the system more complex.



(a) IR-Ship



(b) Satimage

Figure 5.4 Case study: Comparison among generalization error results for batch and the most promising incremental strategies.

Table 5.3 Mean and standard deviation of number of support vectors obtained after learning from all subsets available

Databases	Approaches tested		
	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)

In addition, from these experiments, it can be seen one of the most attractive advantage of incremental learning approaches, which is its capability of reducing the training set size. The results are shown in Table 5.4.

The training 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. The training size reduction is interesting because it accelerates the updating of classifiers, mainly for multi-class problems (e.g. for NistDig with a reduction rate of 61.23%).

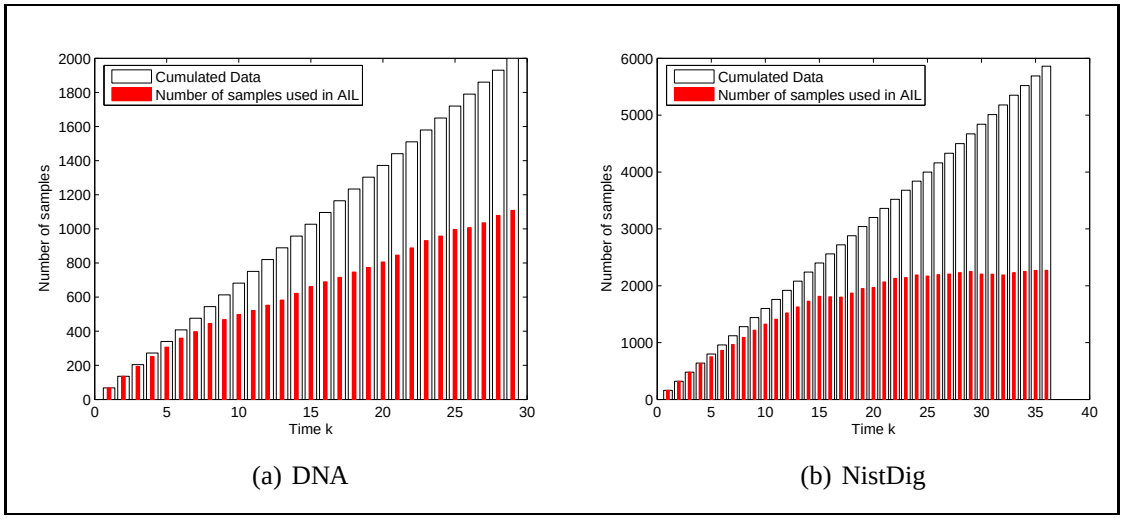
Additionally in the same Table 5.4, we also report the percentage of relevant samples stored by the proposed method with respect to the current total number of support vectors stored and incomming data in the last incremental learning process. We can see that the number of relevant samples may vary depending on each problem, number of classes, data distributions, and density of samples in such relevant 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 was employed involving two problems in Figure 5.5. It can be noticed that the number of training samples retained during system's updating processes can vary depending on the problem and number of

samples. For example in Figure 5.5 (a), the number of samples is smaller than in batch mode, but it seems that the values will always increase. However, as it can be observed in Figure 5.5 (b) for another problem, when more samples are learned after a longer period of time, the number of samples stored may tend to saturate. Other two examples with the lowest and the largest number of samples employed are depicted in Figures 5.6 (a) and (b), respectively.

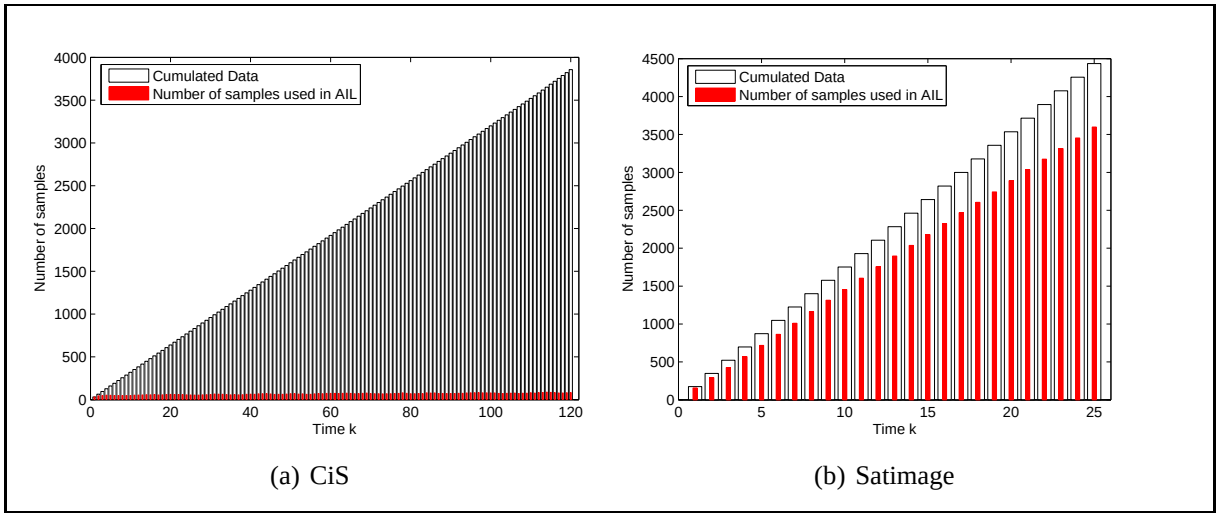
Table 5.4 Training set size reduction (%) by using incremental learning instead batch mode calculated over the last set (first column). Proportion of relevant samples (%) inside the last incremental training set used

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



(a) DNA (b) NistDig

Figure 5.5 Comparison between the number of training samples used by the proposed 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.



(a) CiS (b) Satimage

Figure 5.6 Comparison between the number of training samples used by the proposed 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.

5.2.3.3 On the system parameters' dynamism

These experiments also confirm empirically our underlying hypothesis about the importance of concerning the incremental learning process as a dynamic optimization problem. In order to demonstrate this, we have depicted some results to exhibit the shifting and tracking of optimum solutions over the search space given sequences of datasets $\mathcal{D}(\cdot)$. Through a case study in Figure 5.7, we show that the hyper-parameters selection process represents actually a dynamic optimization problem of type III.

In this example, the search space covered by optimum solutions (denoted here as circles) is depicted for each dataset $\mathcal{D}(k)$ from the Satimage database in one replication. The different sizes of circles represent how the fitness varied between values of 14.38% and 4.56%. The symbol "*" indicates a best solution position found when the whole training data was used in the searching process.

It can be observed that optimum solutions $\mathbf{s}(k)^*$ can vary in both fitness and hyper-parameters values depending on incoming data at different incremental learning steps. For instance, they

can be located in a region for a given intervals 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 fact, therefore, demonstrates that this problem must be dealt as a dynamic optimization problem. It also explains why approaches with fixed parameters (i.e. on $\mathcal{D}(1)$) might perform in a sub-optimum way, as shown in Table 5.2 when IS-AIL is compared with ISVM).

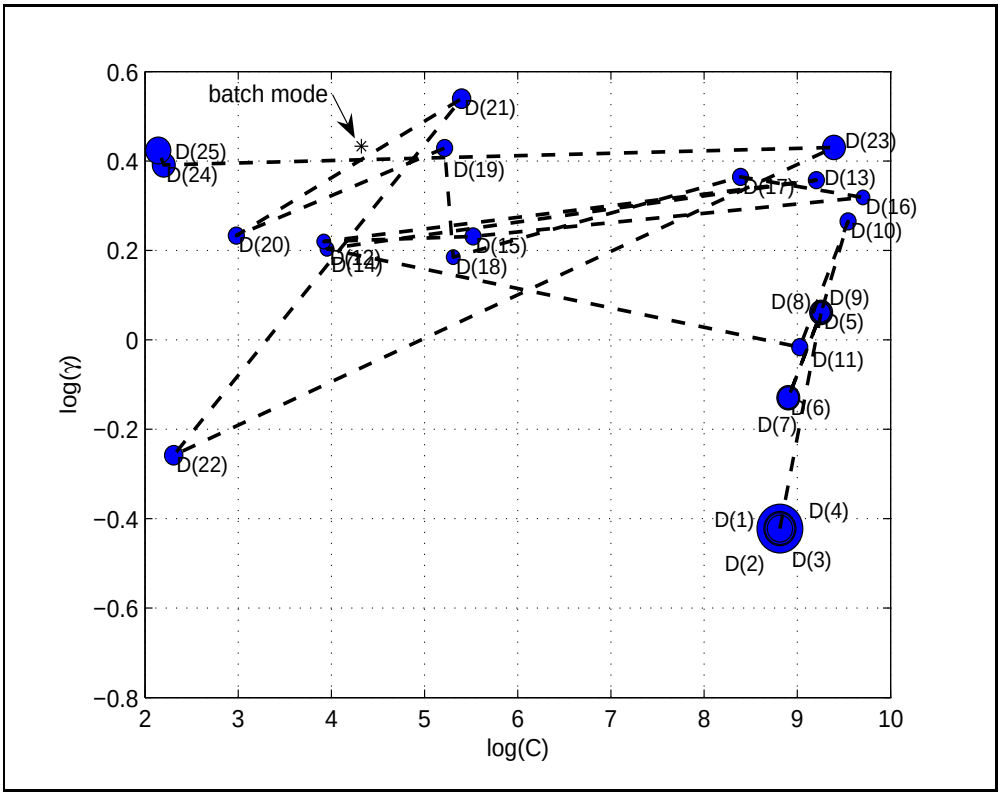


Figure 5.7 Trajectory covered by the best solution found (circles) from incremental steps for each new dataset $\mathcal{D}(k)$. The circles' sizes illustrate how the solutions' fitness can vary. Symbol "*" depicts a best solution position found if the whole training data is used at once (batch mode).

Additionally, Figures 5.8 (a) and (b) report details on which module has pointed out these solutions for each dataset $\mathcal{D}(k)$ and C and γ hyper-parameters, respectively. It can be seen that in most of times, the best values for the hyper-parameters have changed and tracked by the DPSO module. By contrast, in more stable cases, optimized solutions stored in the system's memory could be profited for new learning process by being kept (BK) or selected from the adapted grid-search (AG) module. The frequencies of the AIL modules' activations are listed in

Table 5.5. In these experiments, the dynamic optimization module has produced more often the final hyper-parameter values solution, followed by searches over previous solutions (adapted grid search or keeping the best one).

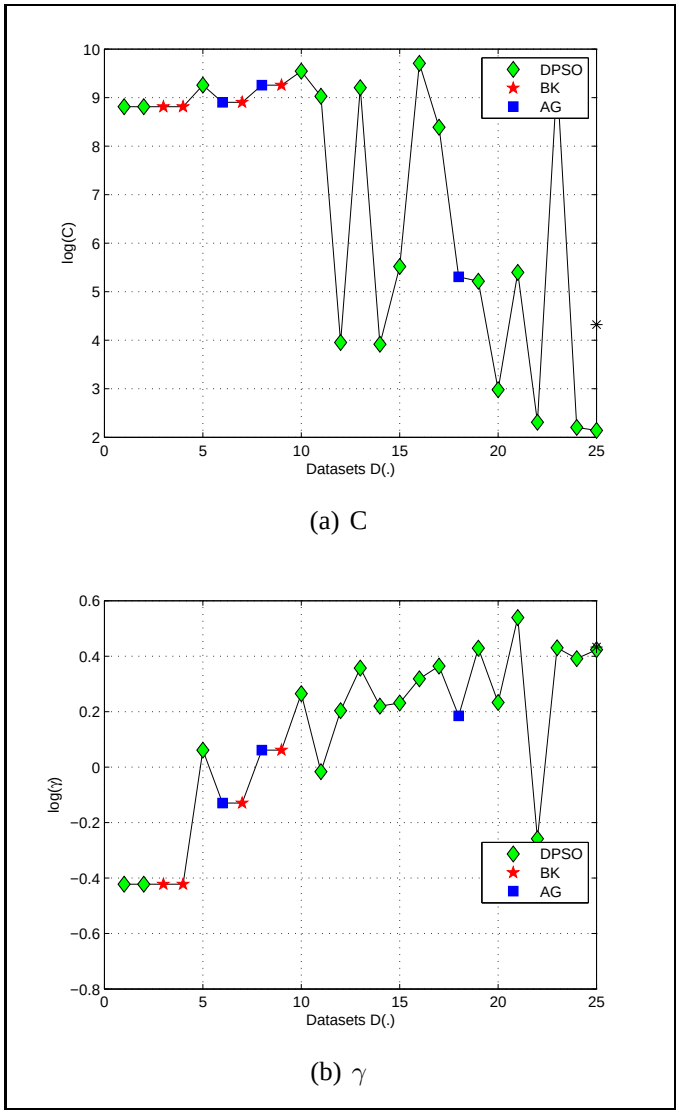


Figure 5.8 Case study: example on how the solutions were pointed out for each dataset $\mathcal{D}(k)$, C, and γ hyper-parameters when using IS-AIL.

Table 5.5 Frequencies (%) of AIL modules' activations over all the training datasets

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

5.2.3.4 On the selection and fusion of solutions into ensembles

Turning now the focus on the dynamic selection of ensemble issue. So far in Table 5.2 we have seen that combining solutions improves the overall system's performance. In this section, the effect of our decision fusion module devoted to this task is outlined. First, Figure 5.9 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.

Based on these results, and others already listed in Table 5.2, it is first demonstrated that the dynamic selection of hyper-parameters and ensembles is very advantageous to provide stability during the incremental learning process and hence to achieve higher performances. Then, in Figures 5.10 and 5.11, we can see some classifiers selected and original pools distributed over the search space for datasets outlined by squares in (a).

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 5.10 or moves over the search space 5.11. That will depend on the problem complexity and current data. In the appendix V, we present the whole sequence of swarms for each dataset $\mathcal{D}(k)$ and complementary results that confirm these same conclusions regarding another case study.

Table 5.6 reports some results on the final cardinalities obtained thereafter the last incremental learning processes. In addition, we also report the variations of cardinalities over all the datasets and replications for three different databases in Figure 5.12.

From these results, we can see that the number of classifiers selected in the ensemble varied around the mean size of the original pool of 20 members. However, more variation among other datachunks were noticed, what indicates that the dynamic selection of classifiers in incremental learning mode is an open issue worth of deeper investigations.

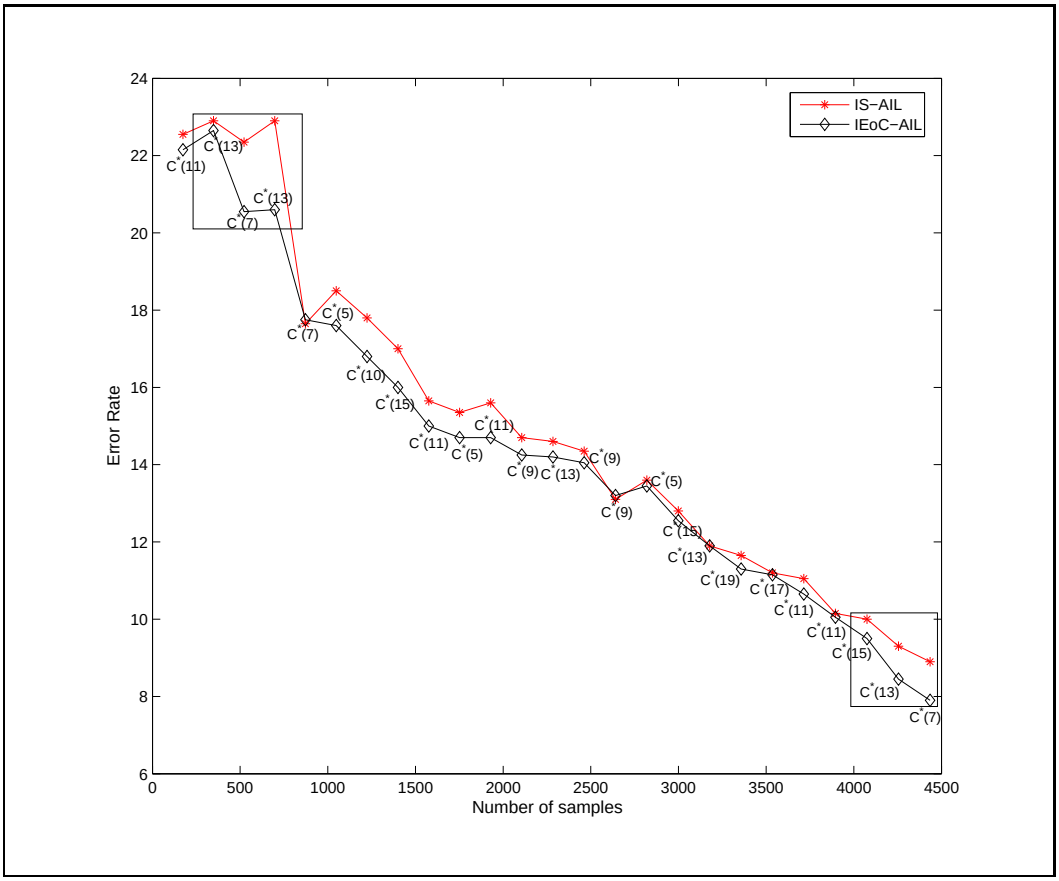
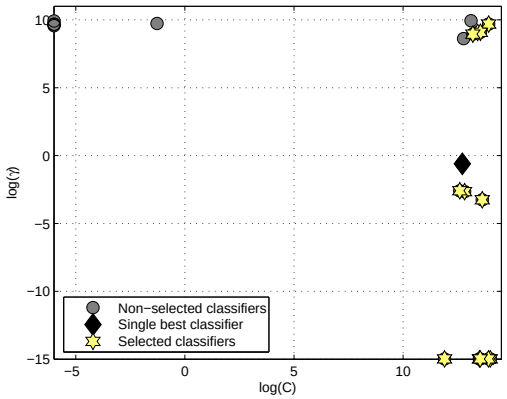
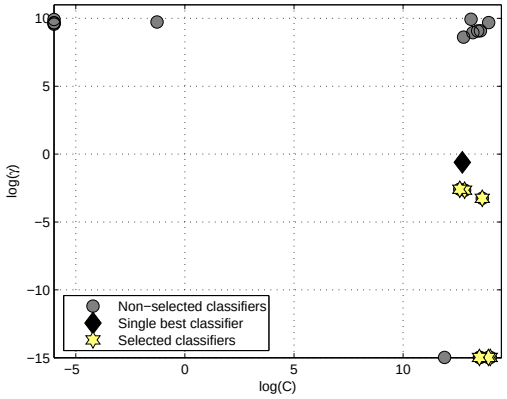


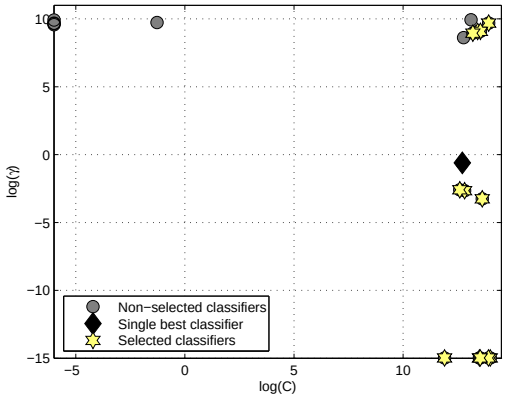
Figure 5.9 Example of results involving performances and cardinalities for each dataset $\mathcal{D}(k)$ for a given replication comparing AIL in single model (IS-AIL) and ensembles dynamically selected (IEoC-AIL).



(a) $\mathcal{D}(2)$, |EoC|=13

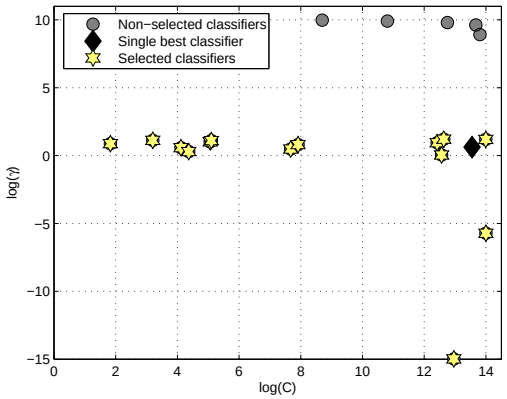


(b) $\mathcal{D}(3)$, |EoC|=7

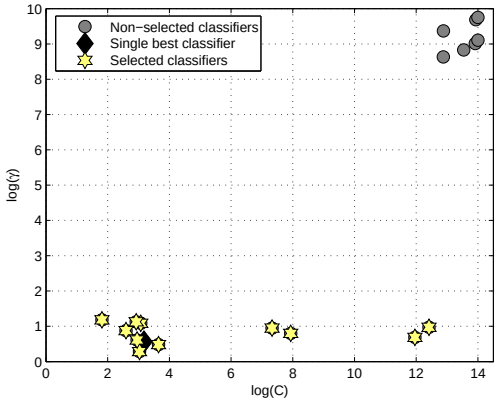


(c) $\mathcal{D}(4)$, |EoC|=13

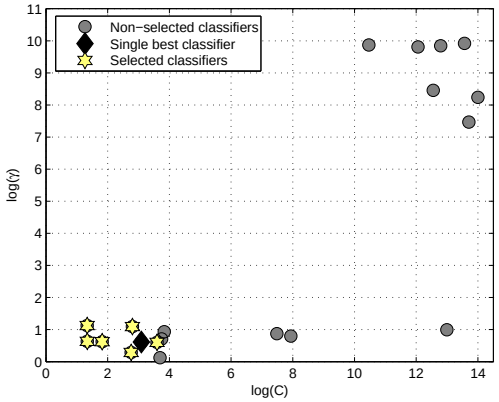
Figure 5.10 Examples of classifiers selected and their original pools distributed over the search space for datachunks outlined by the first square (left side) in Figure 5.9. The entire sequence of swarms for each dataset $\mathcal{D}(k)$ is presented in the appendix V.



(a) $\mathcal{D}(23)$, |EoC|=15

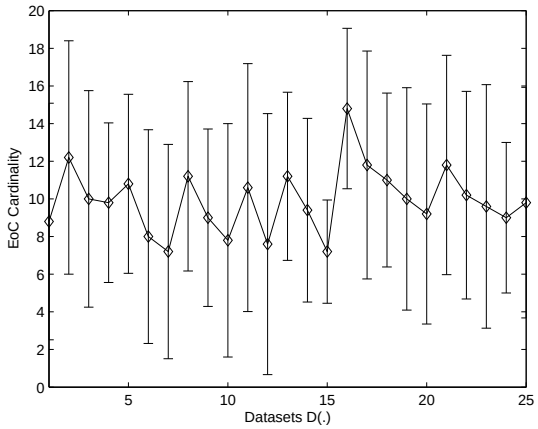


(b) $\mathcal{D}(24)$, |EoC|=13

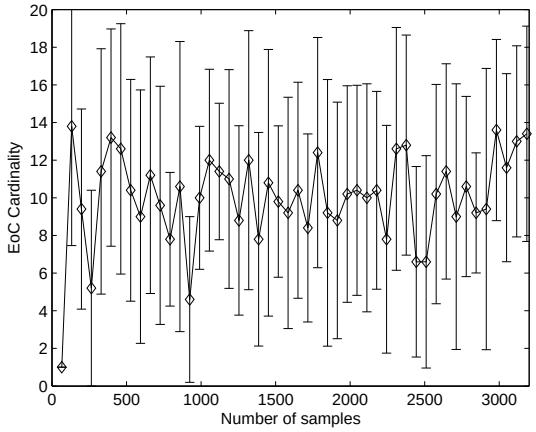


(c) $\mathcal{D}(25)$, |EoC|=7

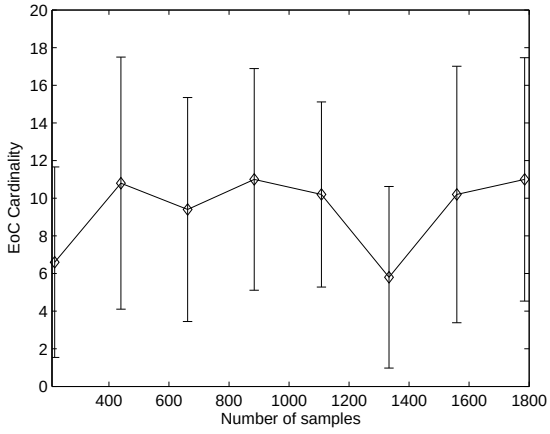
Figure 5.11 Examples of classifiers selected and their original pools distributed over the search space for datachunks outlined by the square (right side) in Figure 5.9. The entire sequence of swarms for each dataset $\mathcal{D}(k)$ is presented in the appendix V.



(a) Satimage



(b) Adult



(c) IR-Ship

Figure 5.12 Results of EoC cardinalities obtained for each dataset $\mathcal{D}(k)$ over 10 replications over different databases.

Table 5.6 EoC-AIL cardinality after dynamic ensemble selection on the last learning step

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

Ending, Table 5.7 lists some results obtained for different configurations investigated when building our decision fusion module. Three combination functions were employed (i.e. majority vote, simple average, and weighted vote), 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 5.1.2.2). By analyzing these results, as occurred for the ORAs strategies in Table 5.2, the simple average combination function achieved better results than the majority vote rule and similar to, or slightly worse than, the weighted vote applied to dynamically selected ensembles. Moreover, these results illustrate the importance of dynamic selection of ensembles, since it improved the results in relation to whole ensembles combined with majority voting. This is possible because they ignore classifiers that could insert some bias in the ensemble's decision and disturb their performances.

Table 5.7 Mean errors obtained with IEoC-AIL concerning different combination functions and ensemble selection rules after learning from all series of datachunks available

Databases	Approaches					
	Majority vote		Simple Average			Weighted vote
	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)

5.3 Discussion

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

Through experiments on different synthetic and real-word databases, we empirically demonstrated that the dynamic optimization of an incremental classification system could improve its performances, so that they could overcome classifiers without adaptation and other classical methods. Therefore, the performance of a classification system depends further than on updating of existing models only, but also on adapting its internal parameters. Furthermore, it was seen that the application of the latter with a multiple classifier approach becomes the classification system more flexible and, at the same time, robust for performing incremental learning and dealing with population drifts.

CONCLUSION

This thesis focused on the implementation of a classification system to perform adaptive incremental learning. Towards the building of the system, our efforts were concentrated on the problems of efficiently accommodation new data, adaptation of internal system's parameters, and combination of multiple hypotheses. We have seen that solving these problems is crucial to increase the overall performance of the system.

In our first investigation, we have seen that a well tuning and updating of classifier's parameters with respect to new data is very important to reach high performance overtime. In order to solve this problem, two main challenges were involved: (1) to overcome common difficulties involving optimization processes, such as the presence of multi-modality or discontinuities in the parameter search space, and (2) to quickly identify optimum solutions which fit both historical and new incoming data. To cope with these two issues, the SVM model selection problem was undertaken 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 particular, we introduced a Particle Swarm Optimization based framework which combines the power of the swarm intelligence theory with the conventional grid-search method to progressively identify and sort out potential solutions for gradually updated training datasets. The idea was to obtain optimal solutions via re-evaluations of previous solutions (adapted grid-search) or via new dynamic re-optimization processes (dynamic particle swarm optimization).

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 pro-

cesses 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) The performance of DMS was very similar to full optimization processes, but less computationally expensive, mainly due to the use of the dynamic optimization techniques. Thus, the experimental results demonstrate that the proposed method outperforms the traditional approaches tested against it while saving considerable computational time. However, even if the optimization of a single classifier is important to increase its performances, the combination of different members can improve the overall performance of a classification system. Mainly when the members composing the ensemble are especially selected, which makes them still more accurate.

Taking this into account, the evaluation and selection of such classifiers depend on the choice of an adequate objective function. Therefore, in order to better understand and employ classifier ensembles for composing our adaptive incremental system in the context of this thesis, the investigation of measures to perform such tasks proceeded this work. We have empirically analyzed several objective functions for the evaluation and so the selection of ensembles of classifiers. In order to achieve this, we empirically investigated classifiers fusion through the relationship between two theories related to ensemble's success, i.e. diversity measures and margin theory, with ensemble accuracy. Most importantly, they revealed valuable insights on how these two theories can influence each other and showed us how confidence based measures can be more interesting than diversity measures for the selection of classifier ensembles.

Finally, we proposed a modular dynamic optimization approach to perform adaptive incremental learning. It was implemented based on these two principles: to incrementally accommodate new data by updating models and to dynamically track new optimum system's parameters for self-adaptation. Thus, the goal was to overcome a problem that occurs when performing incremental learning, which is the obsoleting of best set of classification system's parameters according to incoming data. The proposed method relied on a new framework based on the ideas and components mentioned above. The use of a modified version of incremental Support Vector Machine (ISVM) classifier and a dynamic strategy for the selection of classifier ensembles were the main innovations in relation to our base framework. In particular, from

this framework, the system's optimality in respect to internal parameters, computational cost, and generalization performance could be maintained through the generation of classifiers from optimum regions of parameters search space and the dynamic selection of ensembles based on the classifiers' confidence levels.

As a result, adaptations are realized in two levels, further than by the incremental learning aspect only, but also in the levels of base model parameters and decision fusion. Thus, unlike classical methods considering the incremental system's parameters setting in a static way, we showed that this process should be treated as a dynamic optimization process. This is because their optimum parameters values may shift over the search space depending on incoming data. As additional contributions, we provided insights on strategies to optimize and select classifiers, on the use of memory-based mechanisms, and methods for dynamic optimization processes.

The proposed approach was validated and showed its efficiency through experiments with synthetic and real-world databases, e.g. involving handwritten digits, multisensor remote-sensing images, forward-looking infra-red ship images, etc. Results in single and multiple classifiers configurations demonstrated that the proposed approach actually outperformed classification methods often used in incremental learning scenarios. Moreover, they also demonstrated that the dynamic optimization of an incremental classification system could improve its performances, so that they could overcome classifiers without adaptation and other classical methods. Therefore, the performance of a classification system depends further than on updating of existing models only, but also on adapting its internal parameters. Furthermore, we have observed that the application of the multiple classifier approach becomes the classification system more flexible and, at the same time, robust for performing incremental learning and dealing with population drifts.

Future Works

The results obtained in this thesis were very encouraging and also provide strong foundation for future works. However, some issues were not investigated due to time constraints. Thus, probing deeper, the next stage and future directions of this research might involve:

- Determining new strategies for making the system adaptable to real drifts. In this case, the design of mechanisms to “forget" samples from the system’s memory must be considered to discard old samples that be conflicting with new concepts.

- Carrying on with population drifts situations, but using semi-supervised learning to overcome the dependency of labeled data. This direction requires the developing and embedding of an approach in the framework to label the data before these be used by the other modules.

- Investigating new strategies for the selection of relevant samples and ensembles. The use of information from different times k could be also employed.

- Creating other strategies for better managing the system’s memory. As an example, instead of using only a short term memory, the implementation of an additional long term memory could reduce even more the time for searching for new solutions in those situations in which data changes become recurrent.

Therefore, by following these directions the system surely will be even more versatile.

APPENDIX I

DATABASES

In this appendix we describe more details about some synthetic and real-world databases employed in this thesis.

1 Synthetic Problems

Synthetic problems are useful tools to evaluate learning algorithms. In our experiments, we have used two synthetic problems already employed in the machine learning literature:

- *Circle-in-Square (CiS)* [14]: This problem consists of two classes. The decision boundary is nonlinear, and the samples are uniformly distributed between the range of 0 to 1. One class is represented by a circle inside a square, while the second class is formed of data from the area outside the circle (see Figure I.1). The area of the circle is equal to half of the square [14].

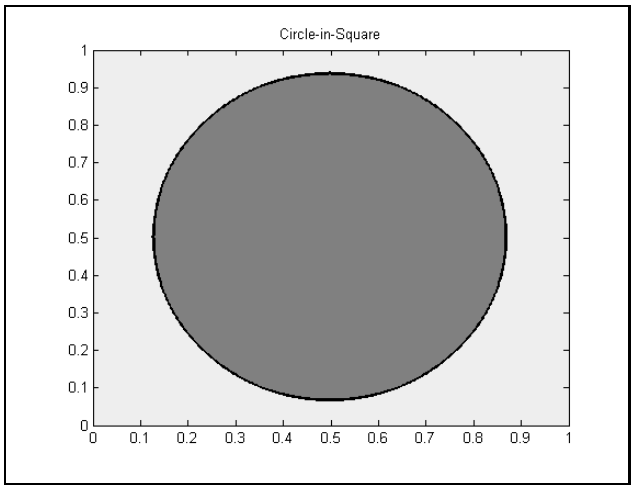


Figure I.1 Illustration of the Circle-in-Square problem.

- *P2* [115]: The P2 problem also consists of two classes (I and II). Each decision region is delimited by one or more of the four simple polynomial and trigonometric functions

(i.e. $Eq_{1-4}(x)$) belongs to one of two classes (see Figure I.2). We consider the same modification on $Eq_4(x)$ suggested in [45], so that the classes have the same area without overlapping. The samples are uniformly distributed between ranges of 0 to 10, and then normalized between 0 and 1.

$$\begin{aligned} Eq_1(x) &= 2 \times sin(x) + 5 \\ Eq_2(x) &= (x-2)^2 + 1 \\ Eq_3(x) &= -0.1x^2 + 0.6 \times sin(4x) + 8 \\ Eq_4(x) &= \tfrac{(x-10)^2}{2} + 7.902 \end{aligned} \tag{I.1}$$

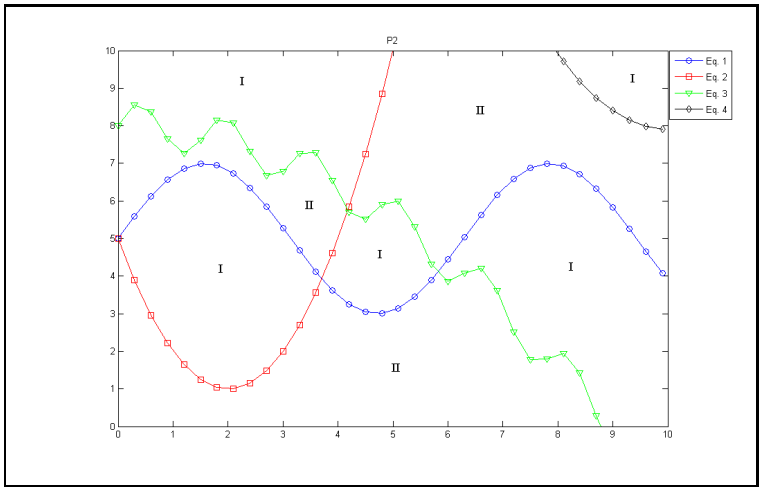


Figure I.2 Illustration of the P2 problem.

The data generated were normalized into a range of [0,1] according to min-max technique defined by Equation I.2. a'_i and a_i are normalized and non-normalized values of the ith feature; min_i and max_i are the minimum and maximum value of the ith feature in the entire dataset.

$$a'_i = \frac{a_i - min_i}{max_i - min_i} \tag{I.2}$$

As in the literature, each class ω_i is represented by 50% of the samples, that is, $P(\omega_i) = 0.5$.

2 Real-world Problems

In this section we summarize two special real-world problems employed in the thesis: NIST-SD19 and the IR-SHIP.

- *NIST-SD19*: It is one of the most popular real-world databases used to evaluate handwritten digit recognition methods. Basically, it is composed of images of handwritten samples forms (hsf) from 0 to 9 organized in eight series. In the literature, it is commonly divided into 3 sets hsf-0123, hsf-4, and hsf-7, for training, validation, and test respectively. Table I.1 depicts the number of samples for each digit class in the test set, where the total number of samples is 60,089. In this work, the maximum number of samples used for training is 5860 (586 samples per class).

Figure I.3 Examples of isolated digits from the NIST-DIG database [85].

The features set extracted from the images of isolated digits were the same suggested by Oliveira et al. [85]. Basically, the features are a mixture of concavity, contour and surface of characters, where the final feature vector is composed of 132 components normalized between 0 and 1. Oliveira et al. have obtained with this features set a recognition rate of

99.13% on the test set samples from hsf-7 using a Multilayer Perceptron Neural Network and a training set of 195,000 samples from hsf-0123.

Table I.1 Number of samples for each digit class in the test set (hsf-7)- NIST-SD19

Class	0	1	2	3	4	5	6	7	8	9
#	5,893	6,567	5,967	6,036	5,873	5,684	5,900	6,254	5,889	5,813

- *IR-SHIP*: The IR-SHIP database is a military database that consist of 2545 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. Images and descriptions of the eight classes of ship are depicted in Figure I.4. In particular, we use the same features set employed by Park and Sklansky [89], which implies in 11 attributes for each FLIR image. In particular, the first seven attributes represent moments and the others four remaining denoted parameters from an auto regressive model. More information about this database can be encountered in [51]. Table I.2 lists the total number of samples for each class. In here, we divided the entire original dataset into 80% and 20% samples for training and test, respectively.

Table I.2 Number of samples for each class in the IR-SHIP database

Class	1	2	3	4	5	6	7	8
#	340	455	186	490	348	279	239	208

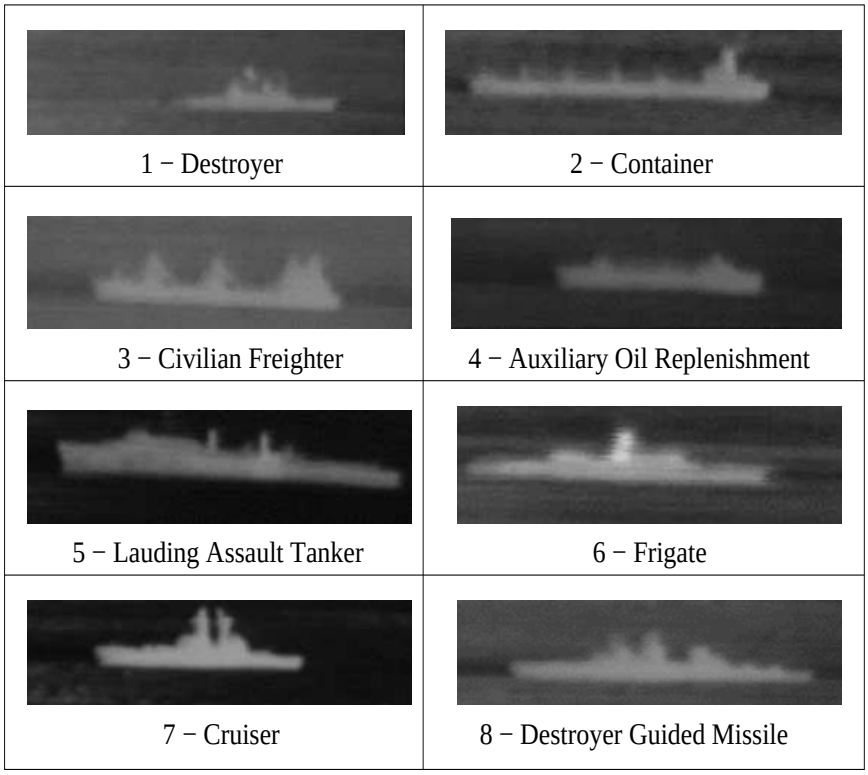


Figure I.4 Examples of FIR images from the IR-SHIP database [51].

APPENDIX II

ADDITIONAL DYNAMIC MODEL SELECTION RESULTS

In this appendix we summarize some additional results related to our PSO-based framework for the dynamic selection of SVM models over five different databases. A brief description on the databases is listed in Table II.1.

Segment, Splice, Mushrooms, and Usps are also databases from [4]. The Segment database contains instances randomly drawn from outdoor images. Each instance is a 3x3 region, where each region represents a class, such as: brickface, sky, foliage, etc. 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 [18].

The results are presented according to the same criteria investigated in chapter 3. First, we report the results involving generalization error rates, number of support vector, and computational time required in Tables II.2, II.3, and II.4, respectively. Then, the average of frequencies that each module was employed to identify the final solution are depicted in Figure II.1.

Table II.1 Databases' descriptions.

Database	Number of Classes	Number of Features	Number of Training Samples	Number of Sets	Number of Test Samples
Segment	7	19	1,848	12	462
Svmguide	2	4	3,089	20	4,000
Splice	2	60	1,000	15	2,175
Mushrooms	2	112	6,498	24	1,626
Usps	10	256	7,291	17	2,007

Table II.2 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
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 II.3 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
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)**

Table II.4 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
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)**

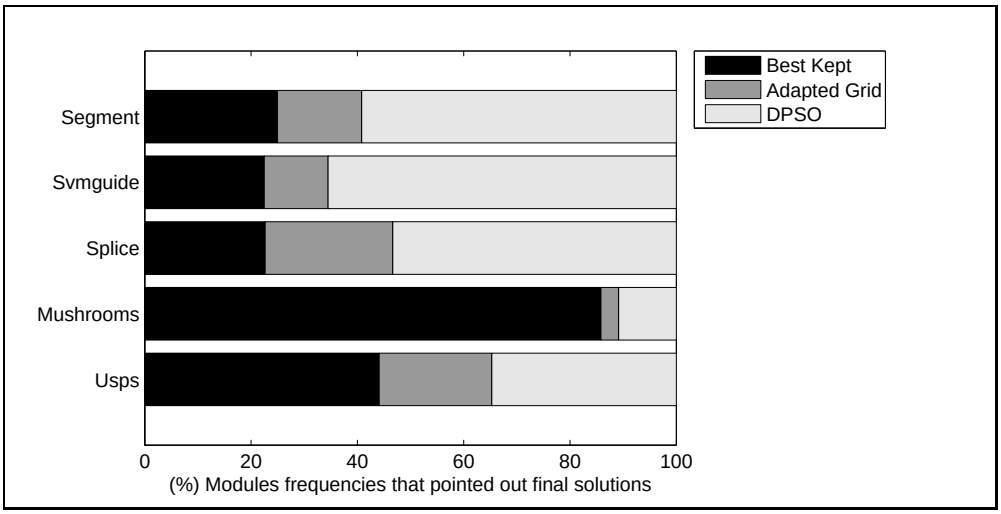


Figure II.1 Average of frequencies which indicates how many times each module was responsible for pointing out the final solution.

APPENDIX III

BIAS-VARIANCE DECOMPOSITION OF ERROR RESULTS

In this appendix we depict some results related to the Bias-Variance Decomposition of the Error theory for ensemble of classifiers introduced by Domingos [36]. We have used two classification problems in the experiments: a synthetic one (P2) and another with real-world data (Satimage).

From this theory, we could observe that the lowest bias corresponds not always to the lowest expected average loss. In fact, the best ensembles have obtained a higher variance, which demonstrates that some variations among the ensemble members is important to achieve better performances. Results illustrating this fact are listed in Tables III.1 and III.2. Therefore, a balance between bias-variance is indeed crucial for developing performing ensembles.

In addition, the Domingos's decomposition of the variance component into unbiased and biased variances allows to analyze the cases in which every measure seems to provide the same result with different ensembles. For instance, in Table III.1 we have two different ensembles, i.e. with $C = 5$ and $C = 10$, that present similar generalization errors. In this case, through this theory we can see that the former ensemble with $C = 5$ and $\gamma = 100$ should be diagnosticated as better than the second one because it has a slightly higher unbiased variance. In other words, both ensembles provided correct answers, but the first one provided with more variated opinions regarding the same dataset.

Furthermore, as demonstrated in [115, 116], we have also observed that the value of the C and γ hyperparameters can actually determine different regions of transition with high bias or stabilized ones for both two-classes and multi-classes problems. Some examples of these regions can be seen in Figures III.1 and III.2. Moreover, we can also see the influence of the hyperparameter values when composing ensembles. For example, for the P2 problem, while lower values of regularization (i.e. for C) results almost in no learning (Figure III.1(a)), the increasing of such parameters notably changes the behavior of the ensembles (Figures

III.1(b)-III.1(d)). The same observations can be outlined for the real-word multi-class database, Satimage, in Figures III.2(a)-III.2(d).

Table III.1 P2 problem

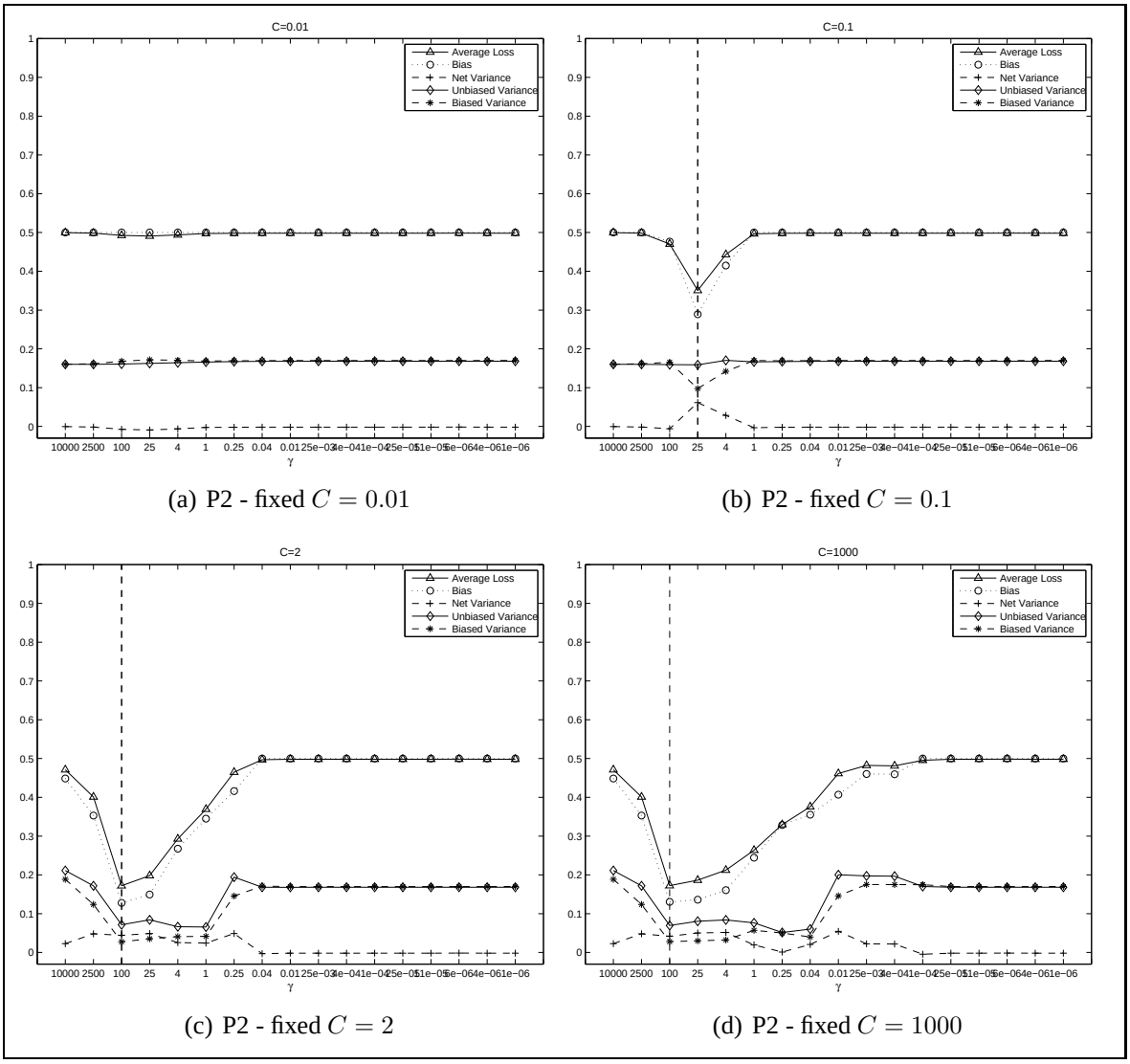
C	γ	**Loss**	**Bias**	**Net Variance**	**Unbiased Variance**	**Biased Variance**
2	100	0.171912	**0.1278**	0.044517	0.070218	0.025698
5	100	0.171568	0.1295	0.042071	0.068568	0.026497
10	100	0.171568	0.1295	0.042068	0.068468	0.026400
20	100	0.172191	0.130400	0.041791	0.069552	0.027764

Table III.2 Satimage problem

C	γ	**Loss**	**Bias**	**Net Variance**	**Unbiased Variance**	**Biased Variance**
5	1	0.109076	0.099174	0.009902	0.022404	0.014455
10	1	0.109572	0.098422	0.011149	0.024493	0.015357
20	1	0.110578	**0.09692**	0.013659	0.026521	0.014831
50	1	0.110834	0.097671	0.013163	0.026702	0.015582

Table III.3 Letter problem

C	γ	**Loss**	**Bias**	**Net Variance**	**Unbiased Variance**	**Biased Variance**
10	1	0.045618	0.034444	0.011173	0.017773	0.009596
20	1	0.045849	**0.033556**	0.012293	0.018880	0.009698
50	1	0.046587	**0.035556**	0.011031	0.018520	0.010676
100	1	0.046680	0.035333	0.011347	0.018702	0.010582



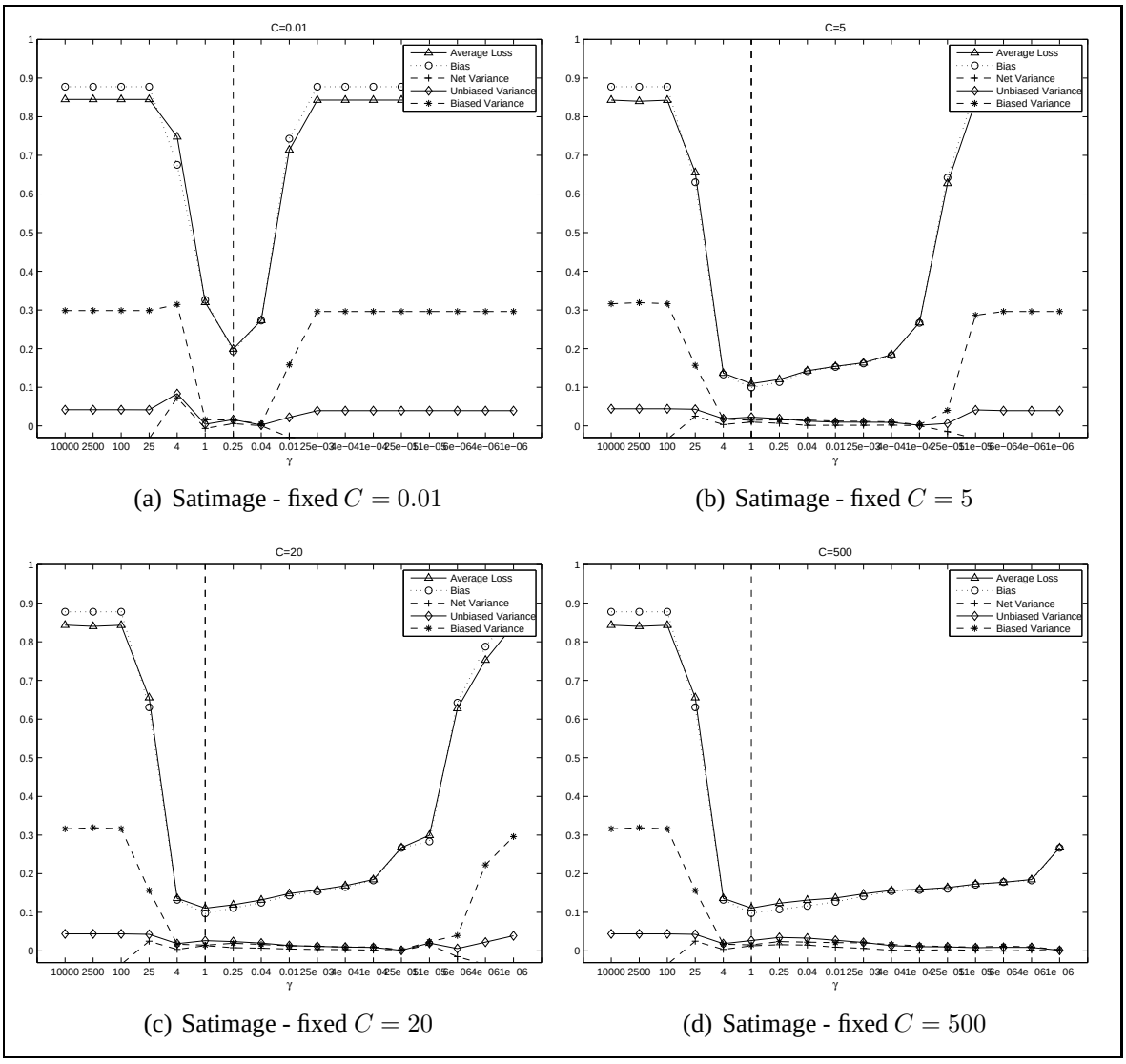
(a) P2 - fixed $C = 0.01$

(b) P2 - fixed $C = 0.1$

(c) P2 - fixed $C = 2$

(d) P2 - fixed $C = 1000$

Figure III.1 Some results concerning the bias-variance decomposition of error theory with ensembles over the P2 database. The vertical dashed lines indicates where the minimal generalization error is attained.



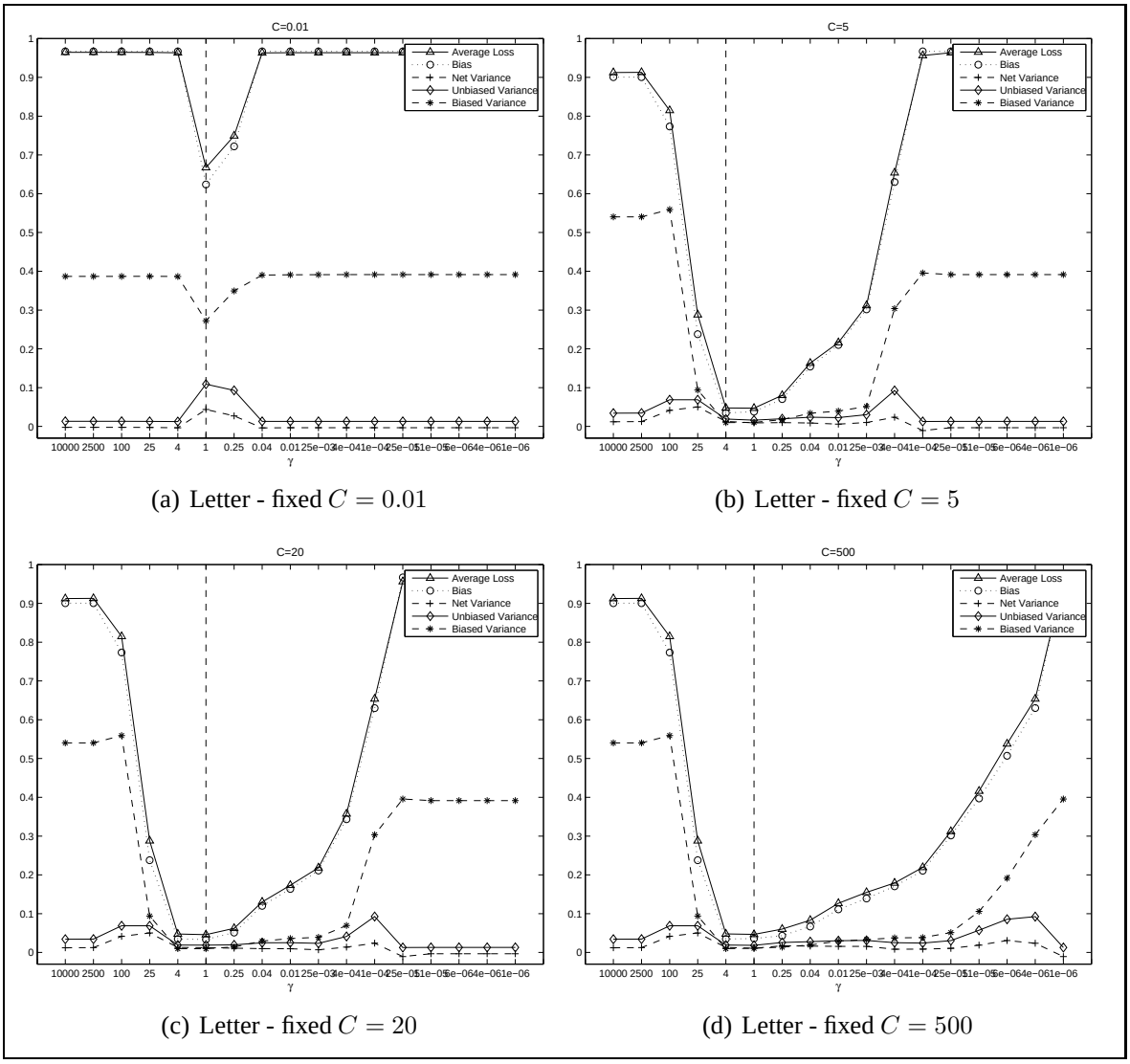
(a) Satimage - fixed $C = 0.01$

(b) Satimage - fixed $C = 5$

(c) Satimage - fixed $C = 20$

(d) Satimage - fixed $C = 500$

Figure III.2 Some results concerning the bias-variance decomposition of error theory with ensembles over the Satimage database. The vertical dashed lines indicates where the minimal generalization error is attained.



(a) Letter - fixed $C = 0.01$

(b) Letter - fixed $C = 5$

(c) Letter - fixed $C = 20$

(d) Letter - fixed $C = 500$

Figure III.3 Some results concerning the bias-variance decomposition of error theory with ensembles over the Letter database. The vertical dashed lines indicates where the minimal generalization error is attained.

APPENDIX IV

EXPERIMENTS WITH CLASSIFIER ENSEMBLE SELECTION

The goal of this appendix is to present some results related to the selection of classifier ensembles using the margin-based measure and the ensemble accuracy studied in chapter 4.

In order to achieve this, we employ a common strategy called: overproduce and choose [42]. In this strategy, several classifiers are created by some ensemble generation method and then a selection process is applied to choose the best ensemble. The aim is, therefore, to (1) improve the overall performance or/and to (2) decrease the complexity of the ensemble by reducing the number of members.

In here, as we are interested in obtaining strong classifiers with low bias, we considered the adjustment of parameters before creating our ensembles. Thus, this experimental protocol can be summarized as follows. First, the parameters of a base classifier were set based on a grid-search. In other words, given a set of parameters and the original training set, a five-fold cross validation was employed to find the best parameter values for a base classifier.

Second, once the best parameters have been defined, individual classifiers were built based on the Bagging ensemble generation method [5]. In this method, ensemble members are trained from L subsets composed of bootstrapped samples from the original training data. Therefore, if the original training set has n examples, a bootstrap replicate of it is constructed by taking n samples with replacement from it, where each example has a probability of $1/n$ of being selected at each turn.

Third, once that the pool of base classifiers were already generated, the minimization of the ensemble generalization error rate and the CI-measure were tested as objective functions for the selection process. The margins used by the CI-measure were computed according to Equation 4.17. In addition, our ensemble selection process was implemented as an optimization process, which employs a genetic algorithm (GA) and an optimization set of samples. Finally, thereafter the best ensembles were selected for each database, they were tested on the respective test sets.

The description of each database used is listed in Table IV.1. As Naive Bayes, KNN and SVM with RBF kernel were employed as base classifiers, we also indicate the best values found for K, C and γ in the same table for each database. The values tested for the SVM hyperparameters are described in chapter 4, and for K were 1,3,5. The final results represent averages over 30 replications.

Table IV.1 Information on the databases

Database	Number of Classes	Number of Features	Training Set size	Optimization Set size	Test Set size	Best K	Best C,γ
P2	2	2	100	100	10,000	1	10,100
Satimage	6	36	3,104	1,331	2,000	5	10,1
Letter	7	19	770	770	770	1	20,1

For each database, ensembles composed of 50 members were built through the bagging method, as previously explained. In addition, the genetic algorithm parameters were set as listed in Table IV.2.

Table IV.2 Genetic algorithm parameter setting

Parameter	Value
Population size	128
Chromosome size (L)	50
Probability of crossover	0.8
Probability of mutation	$1/L$, i.e. 0.02

As each gene of a chromosome represents a classifier, if all bits were selected, all classifiers composed the ensemble. The operations of crossover and mutation were implemented based on the one-point crossover and bit-flip mutation, respectively. The results obtained and the conclusions drawn from these experiments are presented in the next section.

1 Results

The results obtained in this experiment are reported for each database, single classifiers (K-NN, NB, and SVM) in Tables IV.3, IV.5, IV.7. Results for the original pool of classifiers and objective function employed are reported in Tables IV.4, IV.6, IV.8. The best results for each

database are in bold and underlined if they are significantly better than the others. Additionally, Figures IV.1, IV.2, IV.3 depict the generalization error rates and cardinalities achieved by each ensemble type and objective function.

From these results, we can see that in general the minimization of the CI-measure is very promising for selection purposes because it selected very performing ensembles, and sometimes even better than those ensembles selected through the minimization of the generalization error. In addition, regarding the complexity of the ensembles, this measure also selected ensembles with the lowest cardinalities concerning all types of classifier ensembles tested, i.e with NB, KNN, or even SVMs. Thus, it seems to be very advantageous for both accuracy improvement and ensemble reduction size.

Table IV.3 Obtained results with a single classifier on the P2 problem.

Classifier	Generalization Error (%)
K-NN	14.01
NB	28.80
SVM	**13.90**

Table IV.4 Obtained results with ensemble of classifiers on the P2 problem.

	EoC	Average Loss (%)	Generalization Error (%)	
Original EoC	*KNN*	17.38	13.70	
	NB	30.50	26.29	
	SVM	17.23	13.04	
Obj. Func.	**EoC**	**Average Loss (%)**	**Generalization Error (%)**	**Cardinality**
Gen. Error (↓)	*KNN*	17.63 (0.30)	13.53 (0.63)	17
	NB	30.79 (0.49)	25.75 (1.25)	15
	SVM	18.86 (0.88)	12.53 (0.46)	21
CI (↓)	*KNN*	17.39 (0.20)	13.52 (0.38)	9
	NB	30.58 (0.39)	25.93 (1.10)	12
	SVM	17.40 (0.10)	**12.46 (0.20)**	8

Table IV.5 Obtained results with a single classifier on the Satimage problem.

Classifier	Generalization Error (%)
K-NN	10.70
NB	18.95
SVM	**9.55**

Table IV.6 Obtained results on the Satimage problem.

Original EoC	**EoC**	**Average Loss (%)**	**Generalization Error (%)**	
	KNN	12.14	10.40	
	NB	18.97	18.70	
	SVM	10.23	9.40	
Obj. Func.	**EoC**	**Average Loss (%)**	**Generalization Error (%)**	**Cardinality**
Gen. Error (↓)	*KNN*	12.03 (0.05)	10.40 (0.17)	19
	NB	18.82 (0.08)	18.55 (0.13)	16
	SVM	10.26 (0.05)	9.26 (0.14)	19
CI (↓)	*KNN*	11.94 (0.02)	10.26 (0.06)	15
	NB	18.66 (0.03)	18.31 (0.02)	8
	SVM	10.24 (0.00)	**9.24 (0.00)**	9

Table IV.7 Obtained results with a single classifier on the Letter problem.

Classifier	Generalization Error (%)
K-NN	5.46
NB	30.90
SVM	**3.26**

Table IV.8 Obtained results on the Letter problem.

Original EoC	**EoC**	**Average Loss (%)**	**Generalization Error (%)**	
	KNN	7.19	5.45	
	NB	31.40	30.54	
	SVM	4.30	3.08	
Obj. Func.	**EoC**	**Average Loss (%)**	**Generalization Error (%)**	**Cardinality**
Gen. Error (↓)	*KNN*	7.17 (0.04)	5.47 (0.07)	16
	NB	31.28 (0.07)	30.15 (0.18)	20
	SVM	4.27 (0.02)	**3.04 (0.07)**	24
CI (↓)	*KNN*	7.14 (0.00)	5.45 (0.04)	19
	NB	30.96 (0.03)	29.82 (0.12)	14
	SVM	4.26 (0.01)	3.06 (0.03)	20

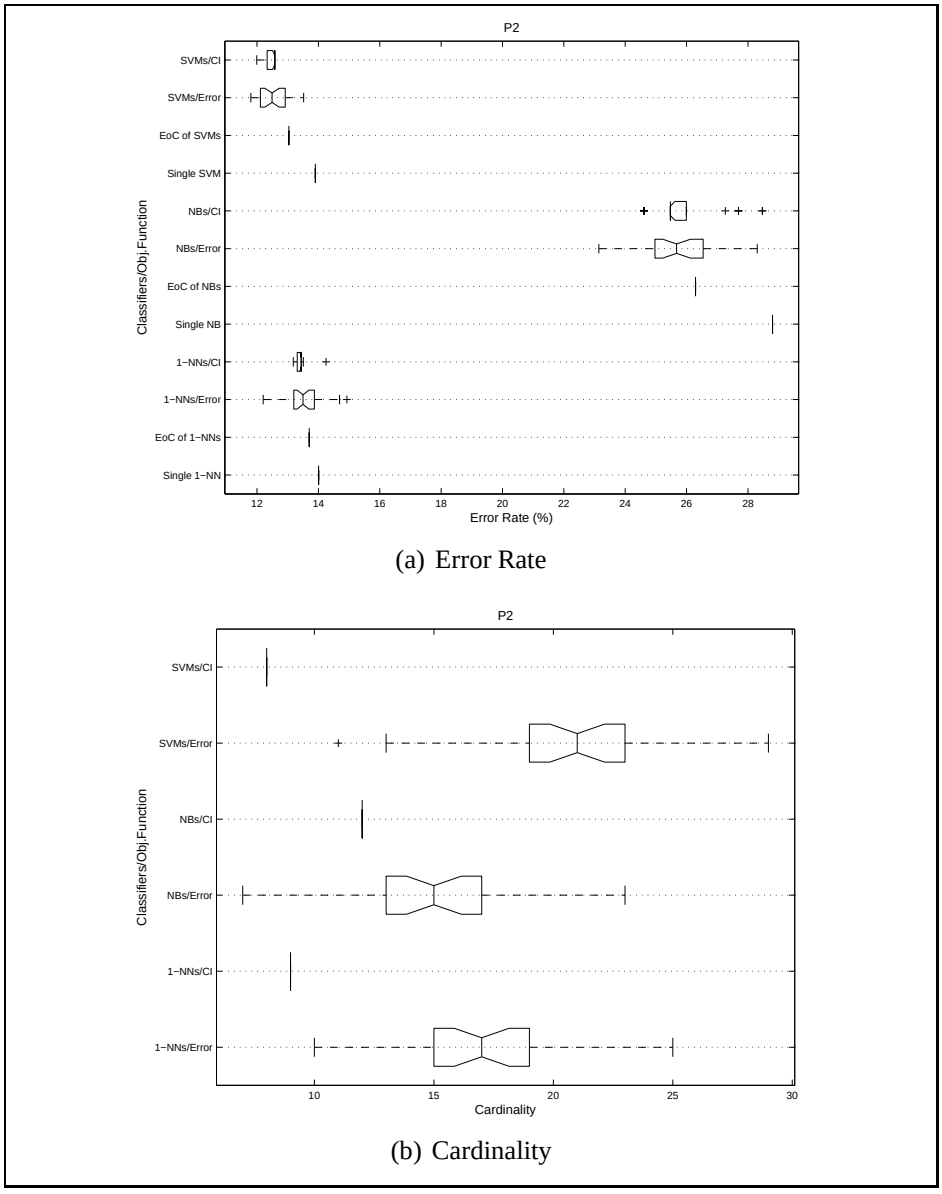


Figure IV.1 Generalization error rates and cardinalities obtained after selection processes for the P2 problem.

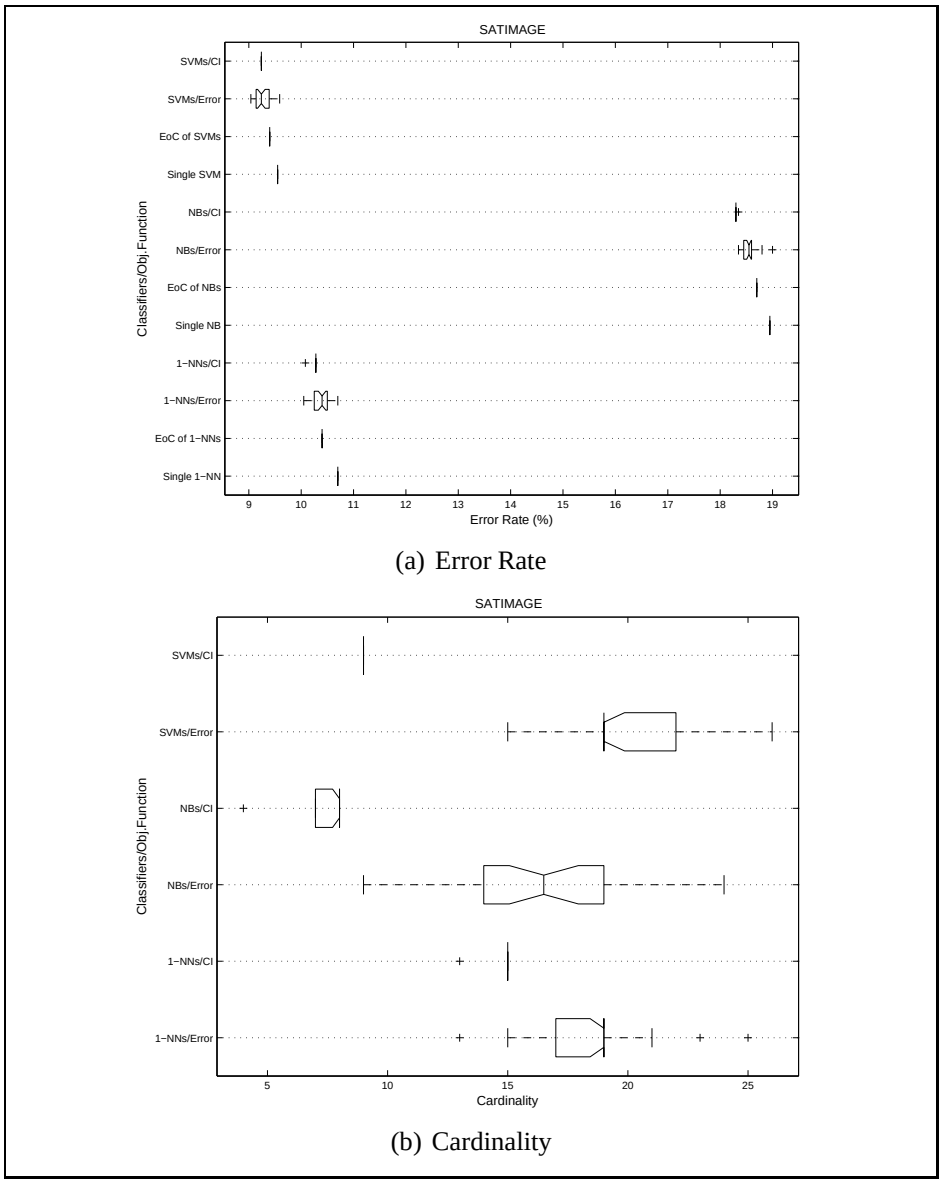


Figure IV.2 Generalization error rates and cardinalities obtained after selection processes for the Satimage problem.

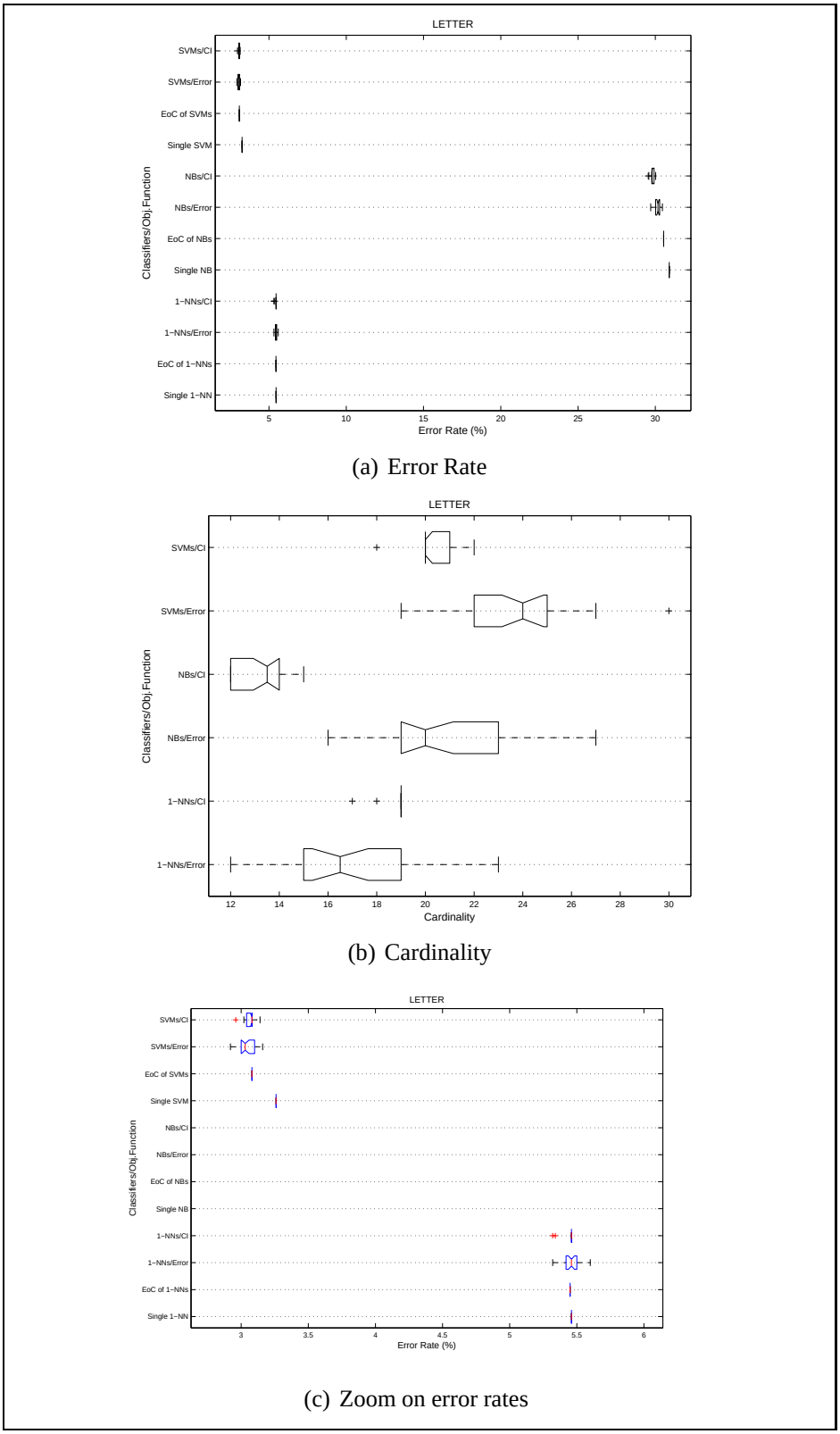


Figure IV.3 Generalization error rates and cardinalities obtained after selection processes for the Letter problem.

APPENDIX V

ADDITIONAL ADAPTIVE INCREMENTAL LEARNING RESULTS

In this appendix we report complementary results involving our adaptive incremental learning strategy presented in chapter 5.

1 Satimage - Swarm Results

In this section, we have depicted from Figure V.2 to V.6 the entire sequence of swarms involving the case study presented in Figure V.1 in section 5.2.3.4. Through this example, it can be clearly seen (1) the dynamism concerning the moving of the particles and (2) the different classifier ensembles selected from several datasets $\mathcal{D}(k)$.

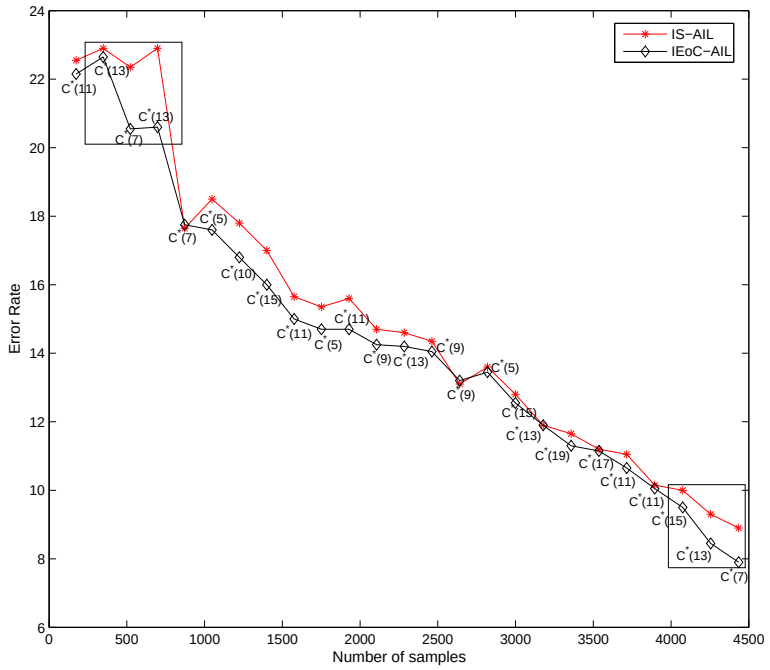
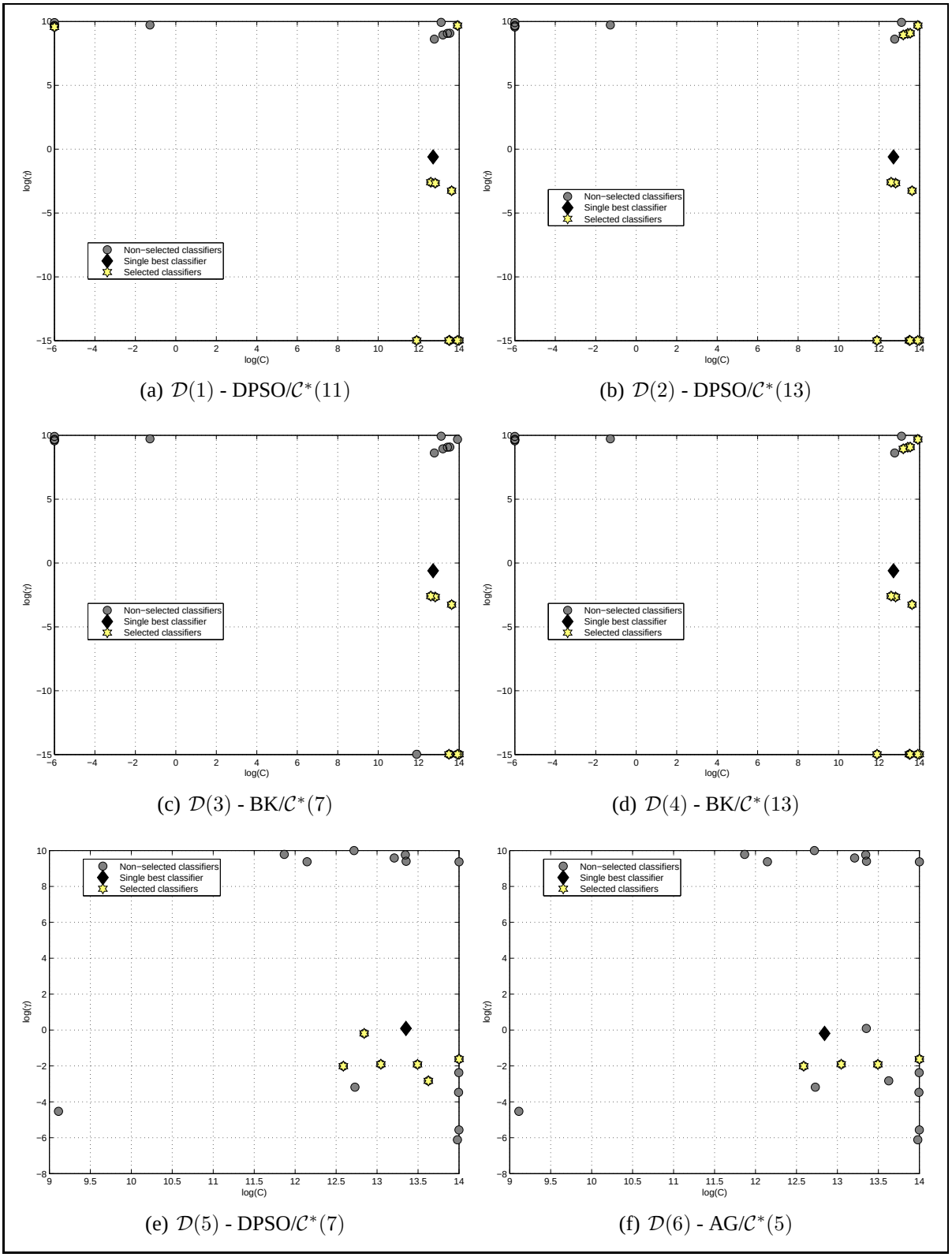


Figure V.1 Results concerning generalization errors and cardinalities for each dataset $\mathcal{D}(k)$ for a given replication comparing AIL in single model (IS-AIL) and ensembles dynamically selected (IEoC-AIL).



(a) $\mathcal{D}(1)$ - DPSO/$\mathcal{C}^*(11)$

(b) $\mathcal{D}(2)$ - DPSO/$\mathcal{C}^*(13)$

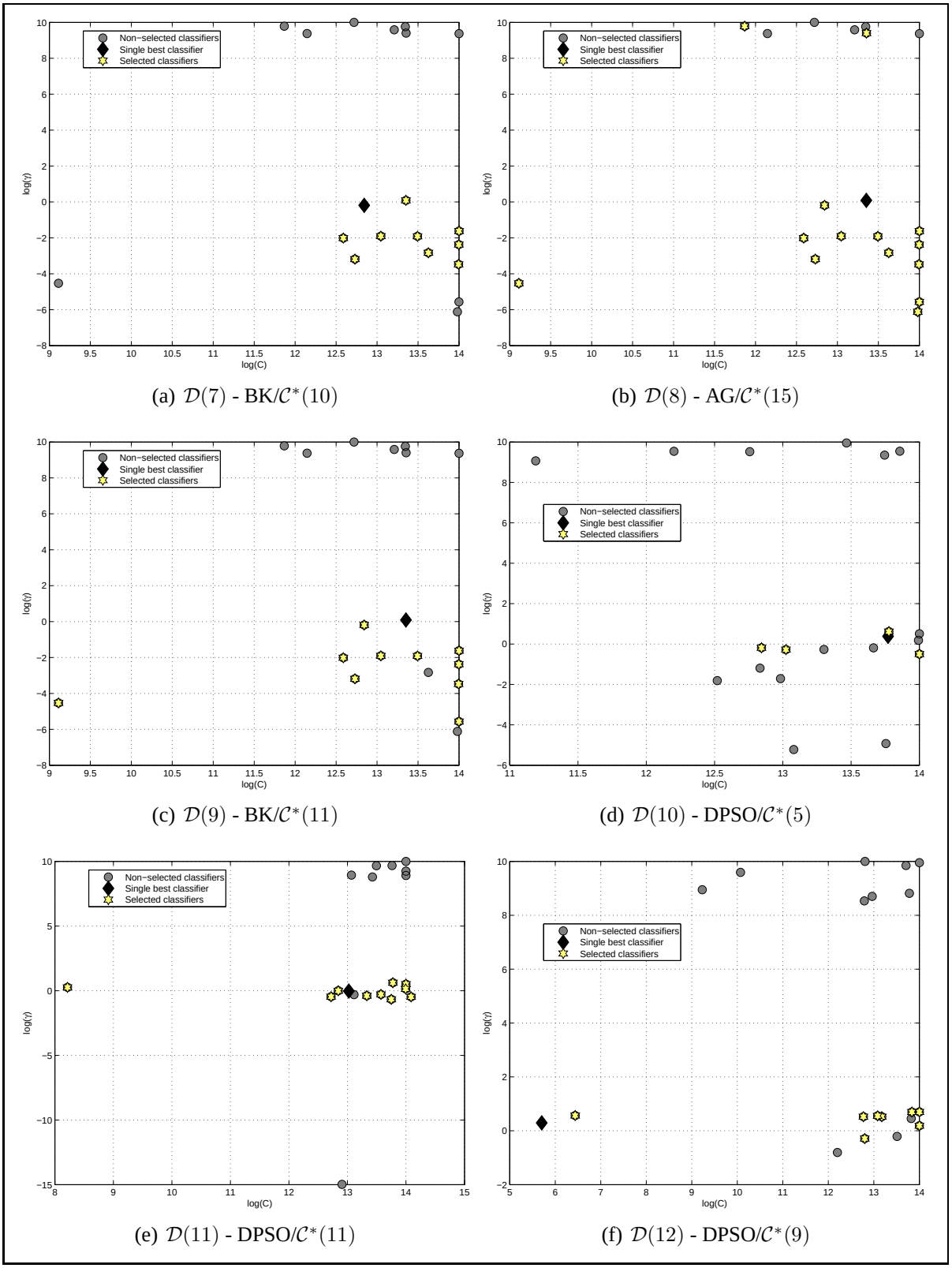
(c) $\mathcal{D}(3)$ - BK/$\mathcal{C}^*(7)$

(d) $\mathcal{D}(4)$ - BK/$\mathcal{C}^*(13)$

(e) $\mathcal{D}(5)$ - DPSO/$\mathcal{C}^*(7)$

(f) $\mathcal{D}(6)$ - AG/$\mathcal{C}^*(5)$

Figure V.2 Sequence of swarms and corresponding particles selected as ensembles between the datasets $\mathcal{D}(1)$ and $\mathcal{D}(6)$. The sequence continues in Figures V.3, V.4, V.5, and V.6



(a) $\mathcal{D}(7)$ - BK/$\mathcal{C}^*(10)$

(b) $\mathcal{D}(8)$ - AG/$\mathcal{C}^*(15)$

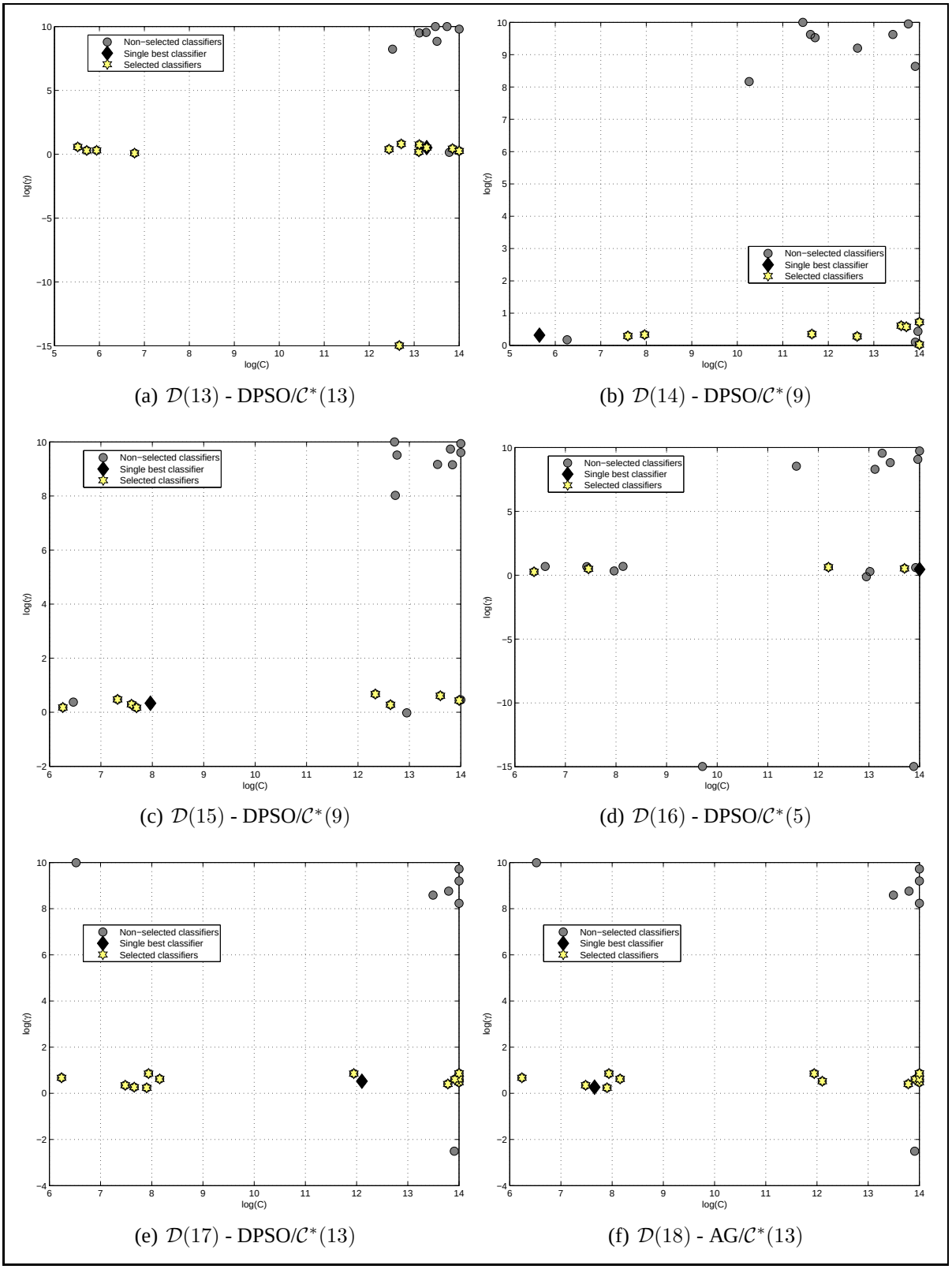
(c) $\mathcal{D}(9)$ - BK/$\mathcal{C}^*(11)$

(d) $\mathcal{D}(10)$ - DPSO/$\mathcal{C}^*(5)$

(e) $\mathcal{D}(11)$ - DPSO/$\mathcal{C}^*(11)$

(f) $\mathcal{D}(12)$ - DPSO/$\mathcal{C}^*(9)$

Figure V.3 Sequence of swarms and corresponding particles selected as ensembles between the datasets $\mathcal{D}(7)$ and $\mathcal{D}(12)$. The other swarms are depicted in Figures V.4, V.5, and V.6.



(a) $\mathcal{D}(13)$ - DPSO/$\mathcal{C}^*(13)$

(b) $\mathcal{D}(14)$ - DPSO/$\mathcal{C}^*(9)$

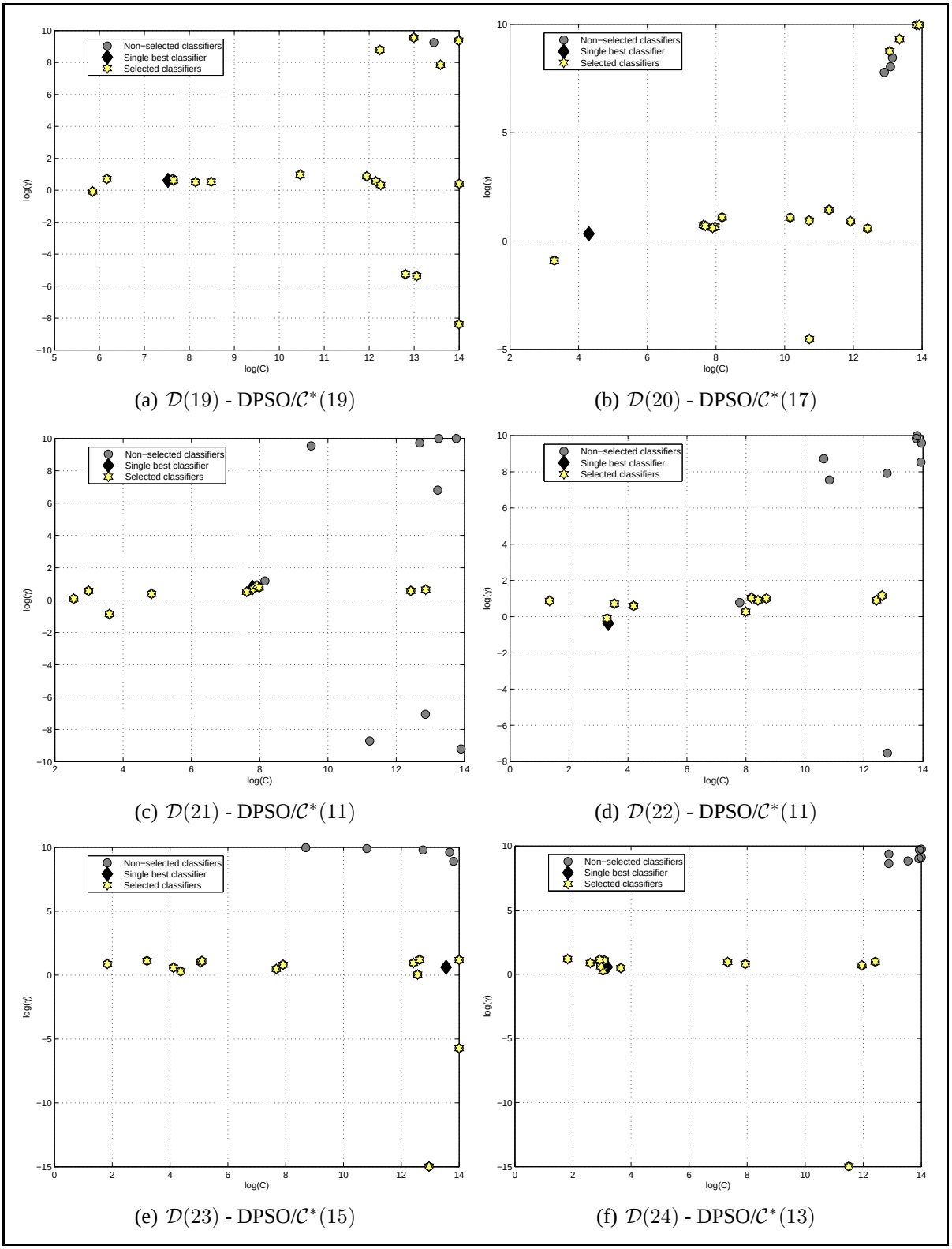
(c) $\mathcal{D}(15)$ - DPSO/$\mathcal{C}^*(9)$

(d) $\mathcal{D}(16)$ - DPSO/$\mathcal{C}^*(5)$

(e) $\mathcal{D}(17)$ - DPSO/$\mathcal{C}^*(13)$

(f) $\mathcal{D}(18)$ - AG/$\mathcal{C}^*(13)$

Figure V.4 Sequence of swarms and corresponding particles selected as ensembles between the datasets $\mathcal{D}(13)$ and $\mathcal{D}(18)$. The final swarms are depicted in Figures V.5 and V.6.



(a) $\mathcal{D}(19)$ - DPSO/$\mathcal{C}^*(19)$

(b) $\mathcal{D}(20)$ - DPSO/$\mathcal{C}^*(17)$

(c) $\mathcal{D}(21)$ - DPSO/$\mathcal{C}^*(11)$

(d) $\mathcal{D}(22)$ - DPSO/$\mathcal{C}^*(11)$

(e) $\mathcal{D}(23)$ - DPSO/$\mathcal{C}^*(15)$

(f) $\mathcal{D}(24)$ - DPSO/$\mathcal{C}^*(13)$

Figure V.5 Sequence of swarms and ensembles selected between the datasets $\mathcal{D}(19)$ and $\mathcal{D}(24)$. The last swarm is depicted in Figure V.6

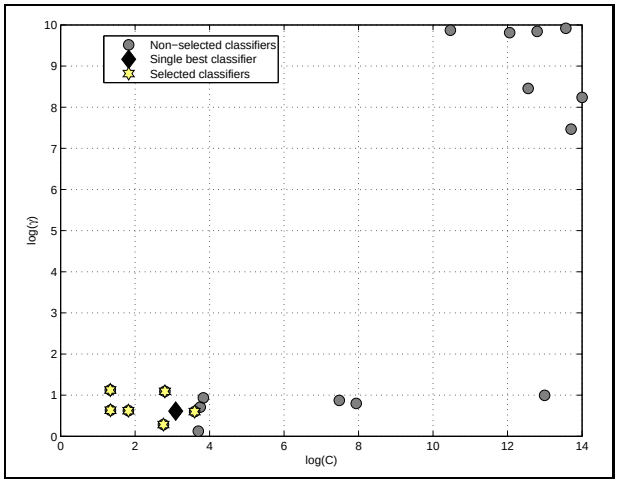


Figure V.6 Last swarm of the sequence. $\mathcal{D}(25)$ **- DPSO/**$\mathcal{C}^*(7)$

2 Case Study - DNA results

This section presents further results and evidences on the system parameters' dynamism and selection of solutions into ensembles for an additional case study, which corresponds to the DNA database. The results are presented through illustrations following the same way employed in chapter 5 (sections 5.2.3.3 and 5.2.3.4).

First of all, we depict in Figure V.7 the trajectory covered by the best solution found for each incremental learning process over different datasets $\mathcal{D}(k)$. Next, Figure V.8 shows an example on how the solutions were found for each dataset $\mathcal{D}(k)$ hyper-parameters when using IS-AIL.

Finally, the generalization errors obtained by combining the solutions found are plotted in Figure V.9. The sequences of swarms computed for each dataset $\mathcal{D}(k)$ are listed in Figures V.10, V.11, and V.12. Overall, these results also confirm the same conclusions discussed in this thesis and exposed in chapter 5.

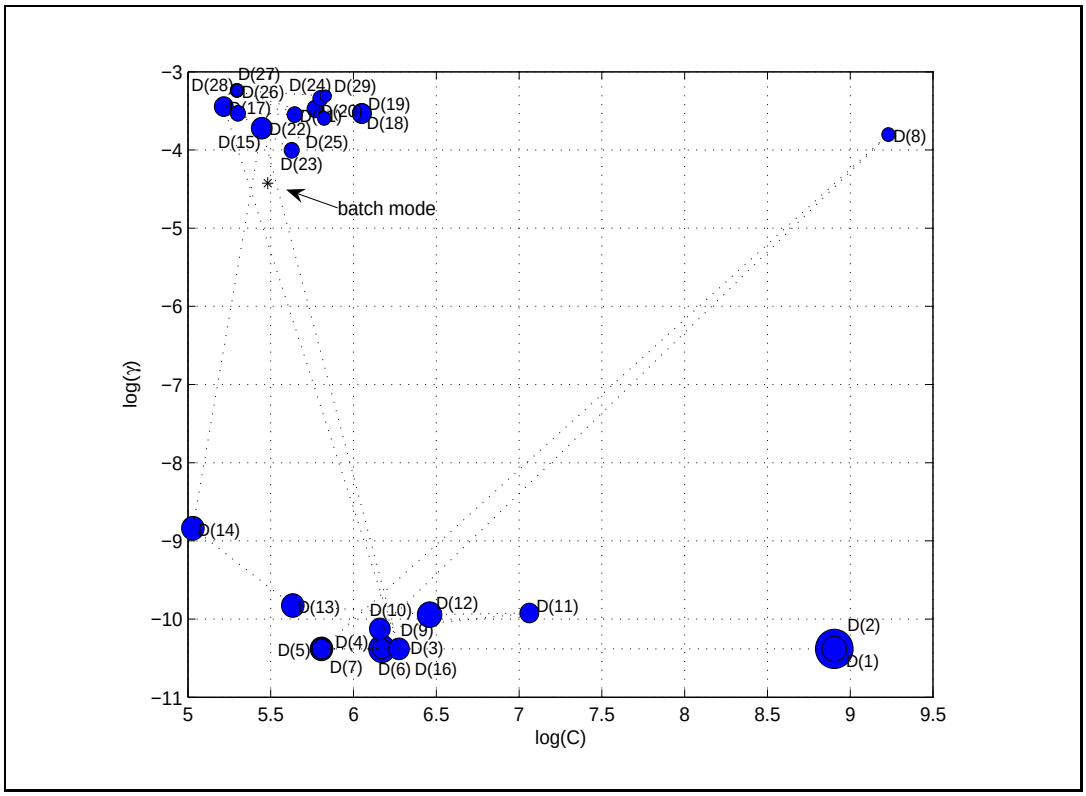
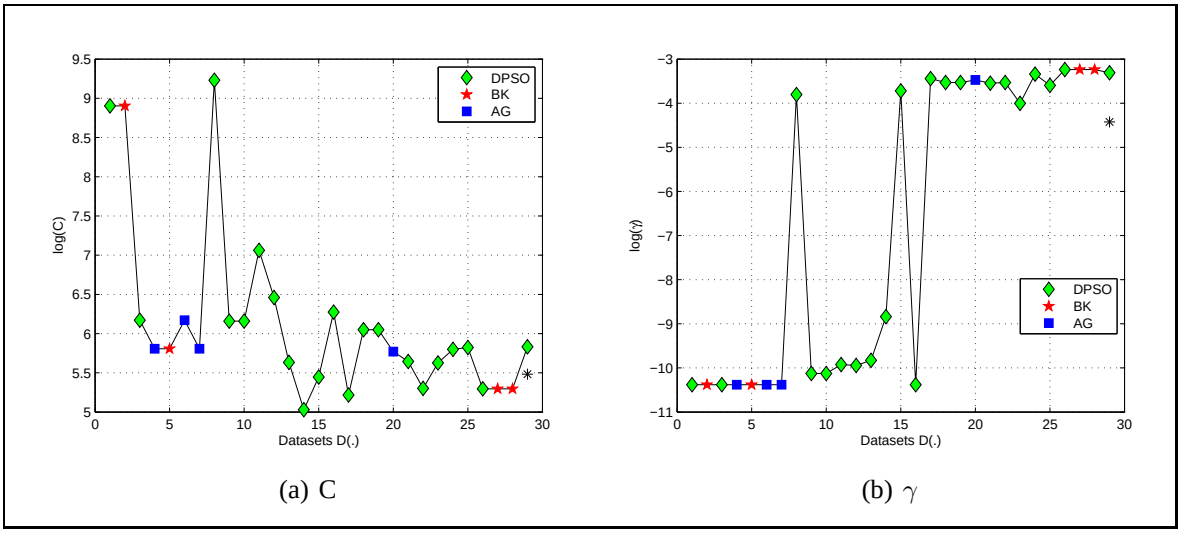


Figure V.7 Trajectory covered by the best solution found (circles) from incremental steps for each new dataset $\mathcal{D}(k)$. The circles' sizes illustrate how the solutions' fitness can vary. Symbol "*" depicts a best solution position found if the whole training data is used at once (batch mode).



(a) C (b) γ

Figure V.8 Case study: example on how the solutions were pointed out for each dataset $\mathcal{D}(k)$, C, and γ hyper-parameters when using IS-AIL.

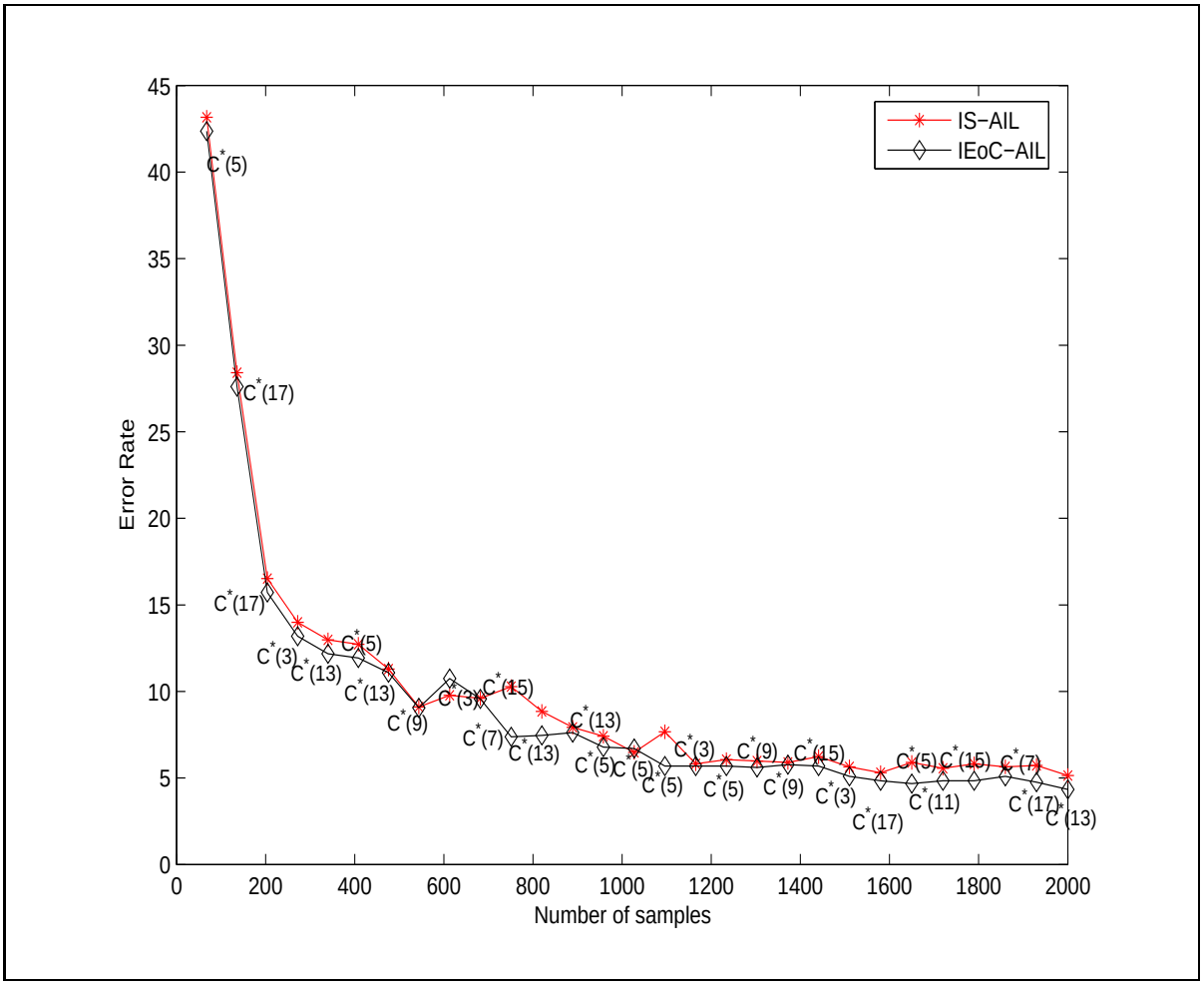
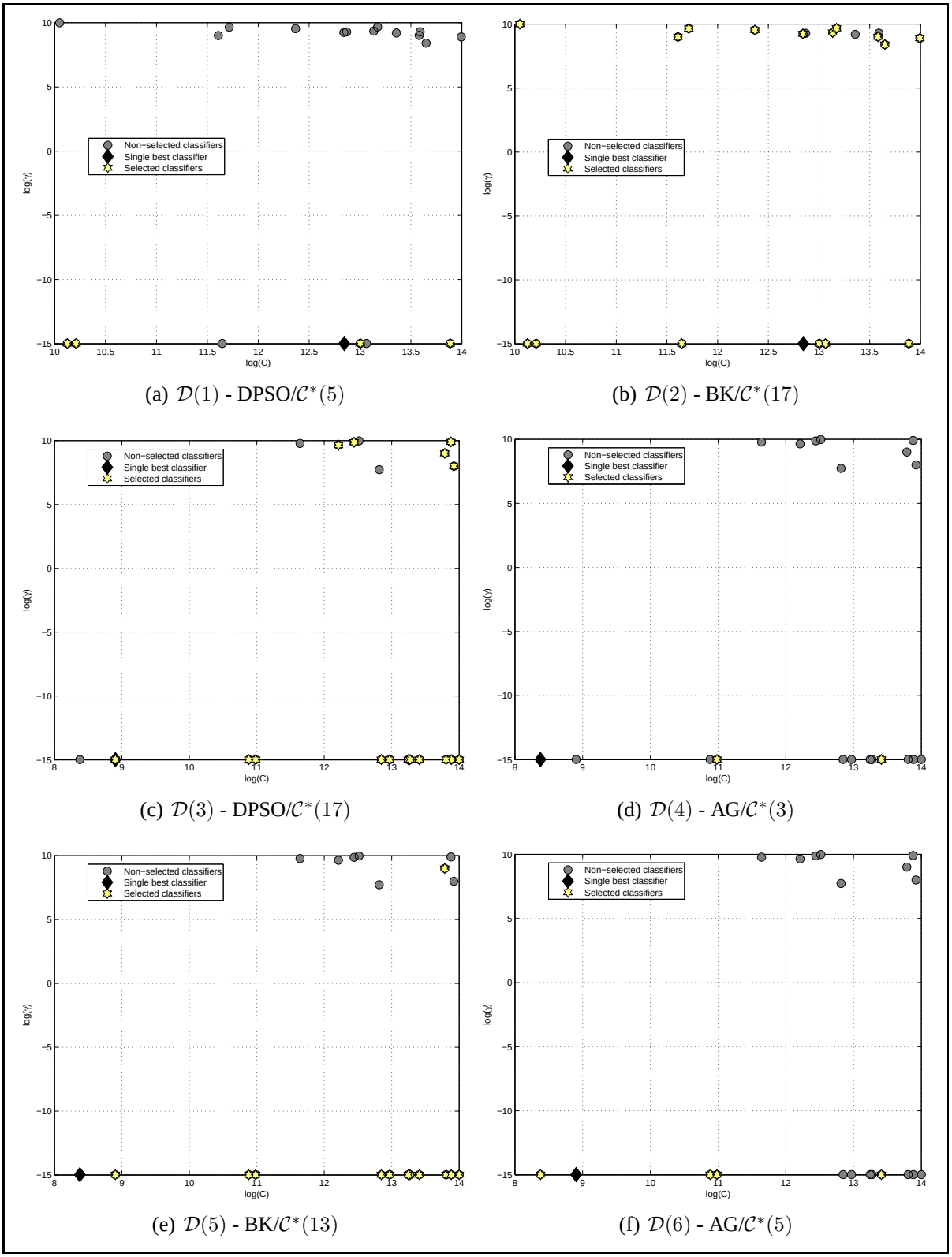


Figure V.9 Generalization error rates and and cardinalities for each dataset $\mathcal{D}(k)$ for a given replication compared to AIL in single model (IS-AIL) and ensembles dynamically selected (IEoC-AIL).



(a) $\mathcal{D}(1)$ - DPSO/$\mathcal{C}^*(5)$

(b) $\mathcal{D}(2)$ - BK/$\mathcal{C}^*(17)$

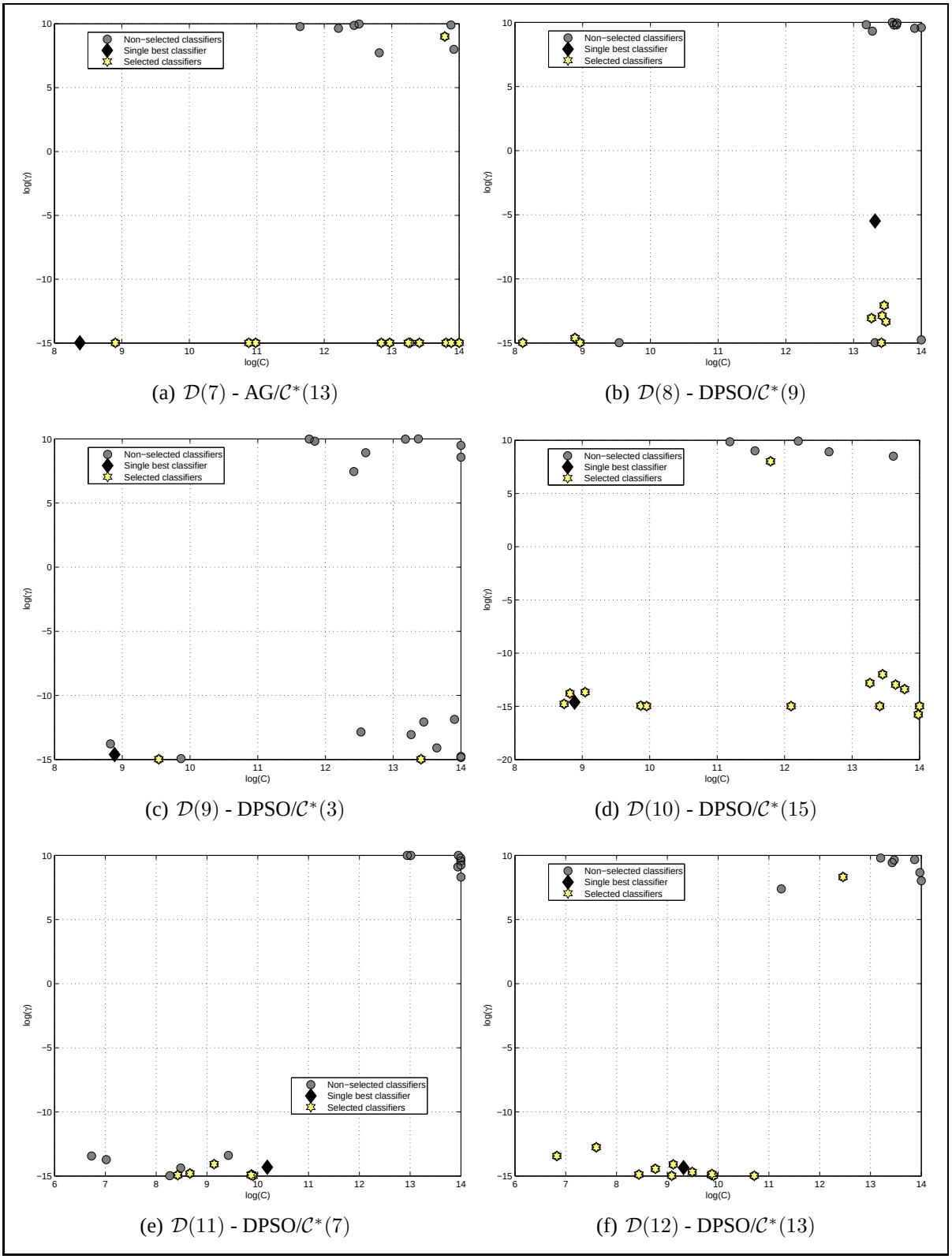
(c) $\mathcal{D}(3)$ - DPSO/$\mathcal{C}^*(17)$

(d) $\mathcal{D}(4)$ - AG/$\mathcal{C}^*(3)$

(e) $\mathcal{D}(5)$ - BK/$\mathcal{C}^*(13)$

(f) $\mathcal{D}(6)$ - AG/$\mathcal{C}^*(5)$

Figure V.10 Swarms and particles selected as ensembles between the datasets $\mathcal{D}(1)$ and $\mathcal{D}(6)$. The sequence continues in Figure V.11



(a) $\mathcal{D}(7)$ - AG/$\mathcal{C}^*(13)$

(b) $\mathcal{D}(8)$ - DPSO/$\mathcal{C}^*(9)$

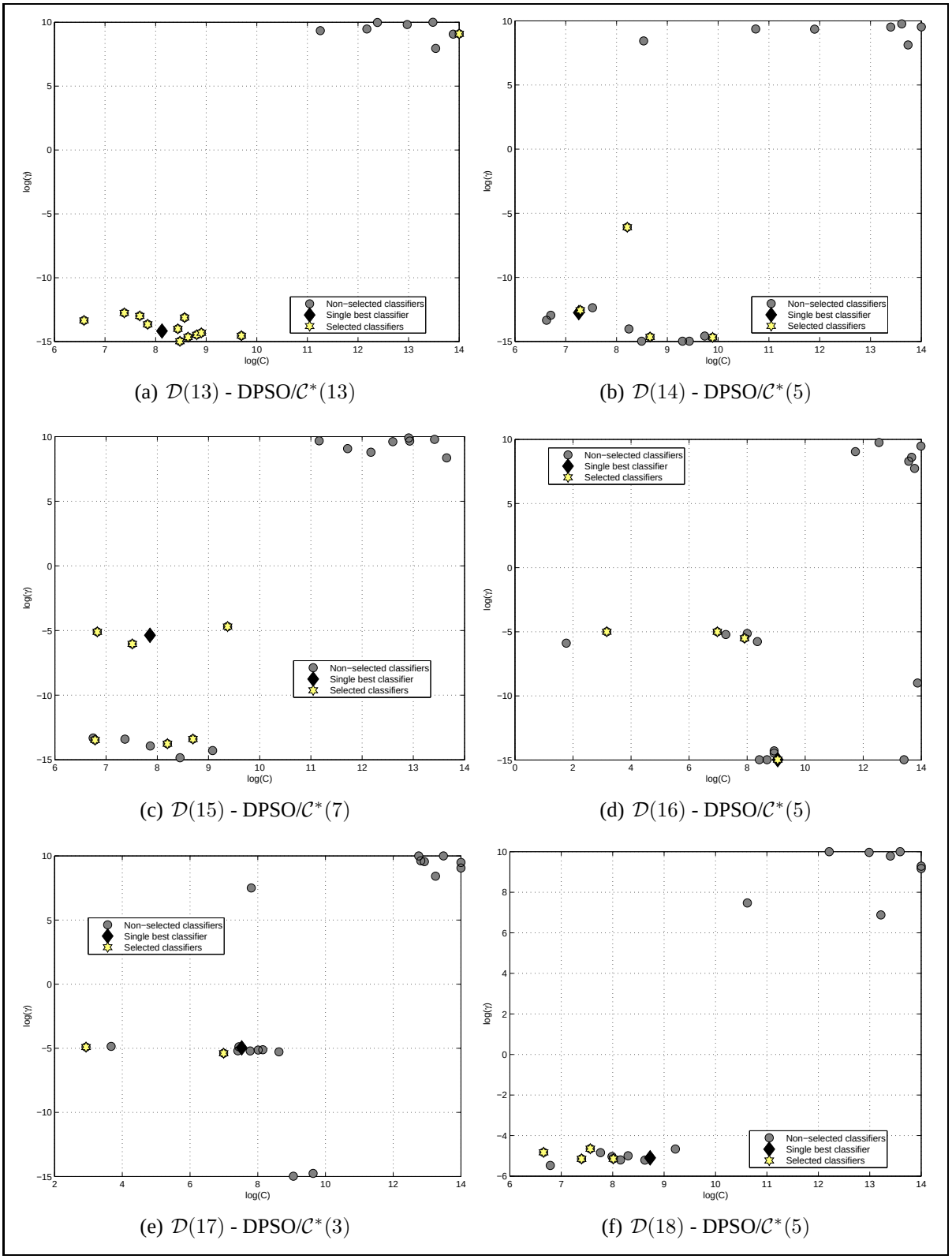
(c) $\mathcal{D}(9)$ - DPSO/$\mathcal{C}^*(3)$

(d) $\mathcal{D}(10)$ - DPSO/$\mathcal{C}^*(15)$

(e) $\mathcal{D}(11)$ - DPSO/$\mathcal{C}^*(7)$

(f) $\mathcal{D}(12)$ - DPSO/$\mathcal{C}^*(13)$

Figure V.11 Swarms and particles selected as ensembles between the datasets $\mathcal{D}(7)$ and $\mathcal{D}(12)$. The sequence continues in Figure V.12



(a) $\mathcal{D}(13)$ - DPSO/$\mathcal{C}^*(13)$

(b) $\mathcal{D}(14)$ - DPSO/$\mathcal{C}^*(5)$

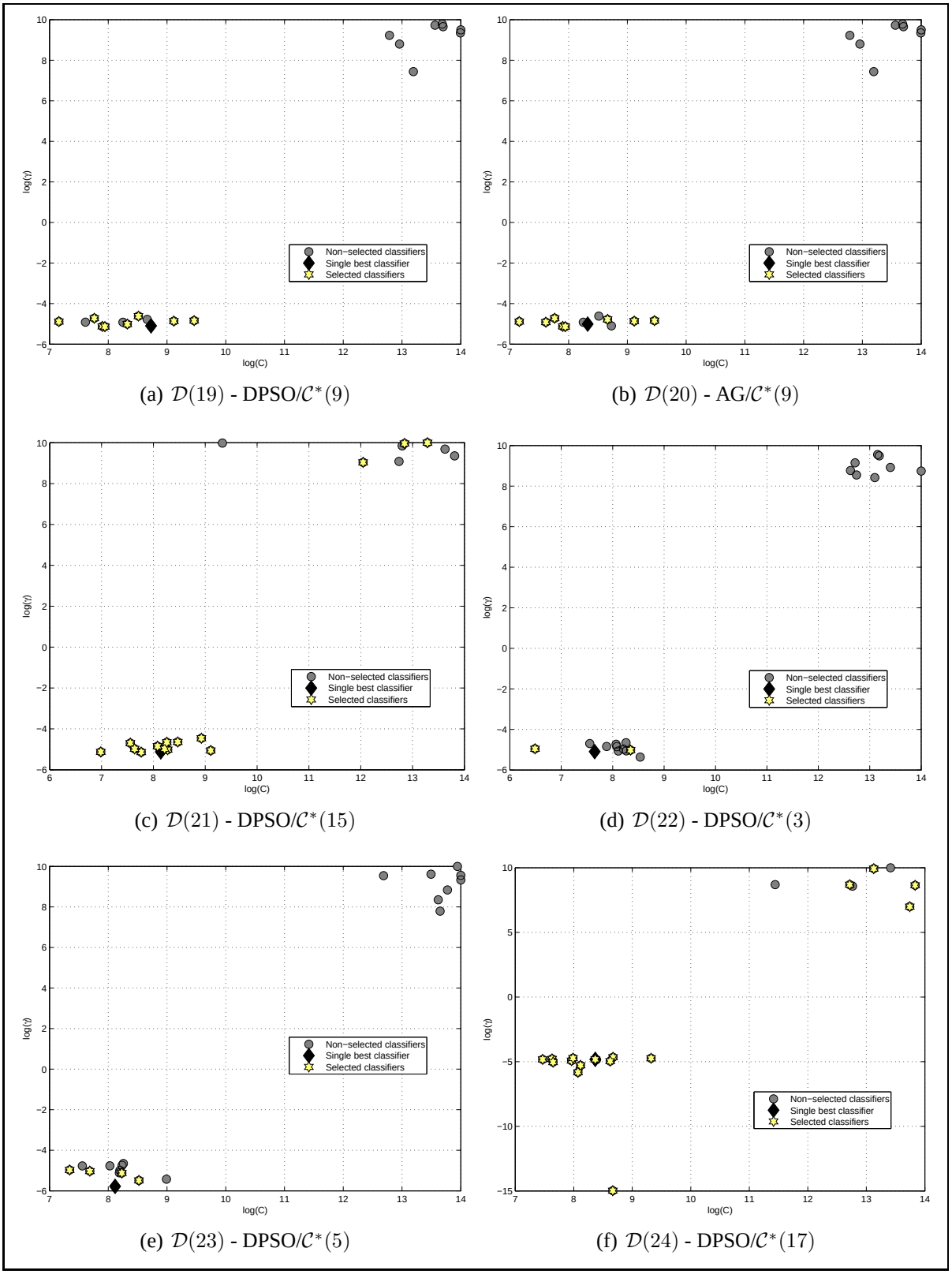
(c) $\mathcal{D}(15)$ - DPSO/$\mathcal{C}^*(7)$

(d) $\mathcal{D}(16)$ - DPSO/$\mathcal{C}^*(5)$

(e) $\mathcal{D}(17)$ - DPSO/$\mathcal{C}^*(3)$

(f) $\mathcal{D}(18)$ - DPSO/$\mathcal{C}^*(5)$

Figure V.12 Swarms and particles selected as ensembles between the datasets $\mathcal{D}(13)$ and $\mathcal{D}(18)$. The sequence continues in Figure V.13



(a) $\mathcal{D}(19)$ - DPSO/$\mathcal{C}^*(9)$

(b) $\mathcal{D}(20)$ - AG/$\mathcal{C}^*(9)$

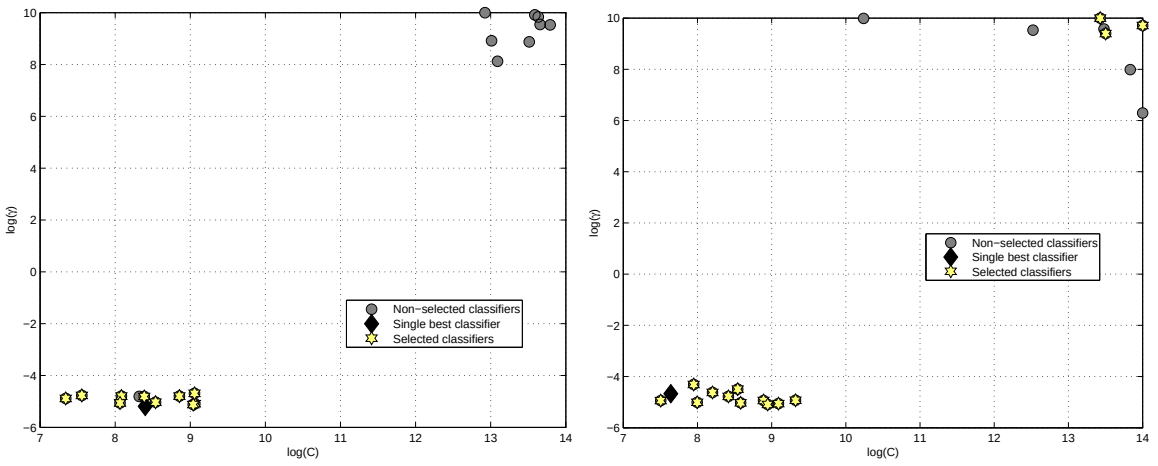
(c) $\mathcal{D}(21)$ - DPSO/$\mathcal{C}^*(15)$

(d) $\mathcal{D}(22)$ - DPSO/$\mathcal{C}^*(3)$

(e) $\mathcal{D}(23)$ - DPSO/$\mathcal{C}^*(5)$

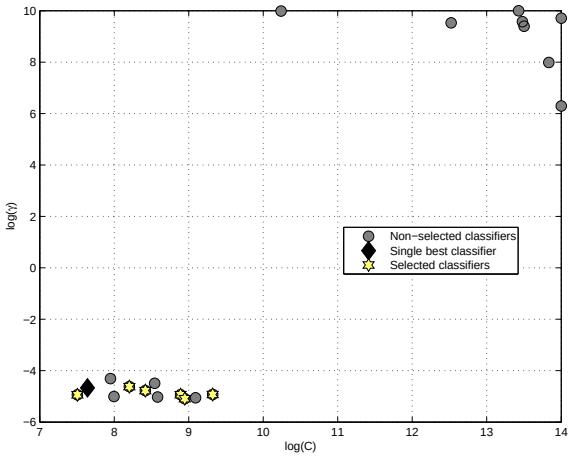
(f) $\mathcal{D}(24)$ - DPSO/$\mathcal{C}^*(17)$

Figure V.13 Swarms and particles selected as ensembles between the datasets $\mathcal{D}(19)$ and $\mathcal{D}(24)$. The sequence continues in Figure V.14

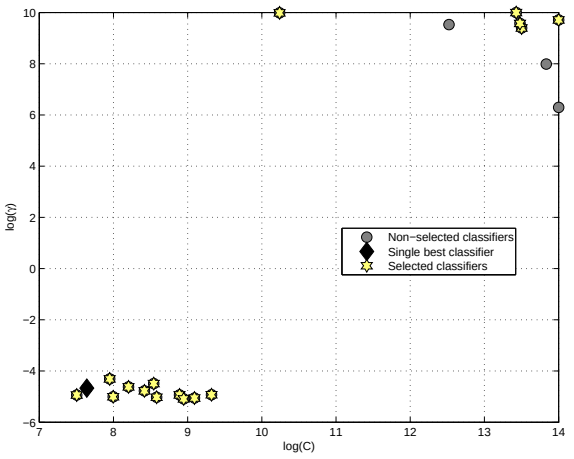


(a) $\mathcal{D}(25)$ - DPSO/$\mathcal{C}^*(11)$

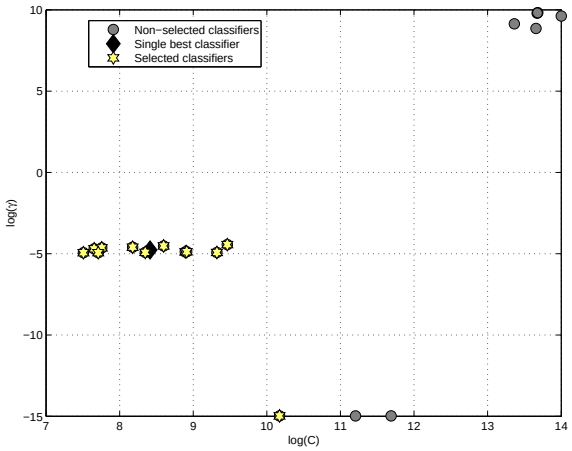
(b) $\mathcal{D}(26)$ - DPSO/$\mathcal{C}^*(15)$



(c) $\mathcal{D}(27)$ - BK/$\mathcal{C}^*(7)$



(d) $\mathcal{D}(28)$ - BK/$\mathcal{C}^*(17)$



(e) $\mathcal{D}(29)$ - DPSO/$\mathcal{C}^*(13)$

Figure V.14 Last swarms and particles selected as ensembles between the datasets $\mathcal{D}(25)$ and $\mathcal{D}(29)$.

BIBLIOGRAPHY

[1] Jin-Long An, Zheng-Ou Wang, and Zhen-Ping Ma. Incremental learning algorithm for support vector machine. In *Proceedings of the 2nd International Conference on Machine Learning and Cybernetics*, pages 1153–1156, 2003.

[2] Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Georgios Sakkis, Constantine D. Spyropoulos, and Panagiotis Stamatopoulos. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. In *Proceedings of the 4th Workshop on Machine Learning and Textual Information Access*, pages 1–13, 2000.

[3] N.E. Ayat, M. Cheriet, and C.Y. Suen. Automatic model selection for the optimization of SVM kernels. *Pattern Recognition*, 38(10):1733–1745, October 2005.

[4] C. L. Blake and C. J. Merz. UCI repository of machine learning databases., 1998.

[5] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[6] Leo Breiman. Arcing the edge. technical report, Department of Statistics, University of California, Berkeley, CA, 1997.

[7] Leo Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1-2):85–103, 1999.

[8] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[9] Gavin Brown, Jeremy L. Wyatt, Rachel Harris, and Xin Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, 2005.

[10] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[11] Doina Caragea, Adrian Silvescu, and Vasant Honavar. Incremental and distributed learning with support vector machines. In *Proceedings of the AAAI Conference on Artificial Intelligence*, page 1067, 2000.

[12] A. Carlisle and G. Dozier. Tracking changing extrema with adaptive particle swarm optimizer. In *Proceedings of the 5th Biannual World Automation Congress*, pages 265–270, Orlando, USA, 2002.

[13] Gail A. Carpenter, Stephen Grossberg, Natalya Markuzon, John H. Reynolds, and David B. Rosen. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3(5):698–713, 1992.

[14] Gail A. Carpenter, Stephen Grossberg, and John H. Reynolds. ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks*, 4(5):565–588, 1991.

[15] Gail A. Carpenter and Natalya Markuzon. ARTMAP-ic and medical diagnosis: Instance counting and inconsistent cases. *Neural Networks*, pages 323–336, 1998.

[16] Gail A. Carpenter and William D. Ross. Art-emap: A neural network architecture for object recognition by evidence accumulation. *IEEE Transactions on Neural Networks*, 6(4):805–818, 1995.

[17] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Advances in Neural Information Processing Systems*, pages 409–415. MIT Press, 2001.

[18] C. C. Chang and C. J. Lin. Libsvm: a library for support vector machines, 2005.

[19] Olivier Chapelle and Vladimir Vapnik. Model selection for support vector machines. *In Advances in Neural Information Processing Systems*, pages 230–236, 1999.

[20] Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3):131–159, 2002.

[21] C. Chatelain, S. Adam, Y. Lecourtier, L. Heutte, and T. Paquet. Multi-objective optimization for SVM model selection. In *Proceedings of the 9th International Conference on Document Analysis and Recognition*, pages 427–431, 2007.

[22] Zheng Chunhong and Jiao Licheng. Automatic parameters selection for SVM based on GA. In *Proceedings of the 5th World Congress on Intelligent Control and Automation*, pages 1869–1872, 2004.

[23] Maurice Clerc. *Particle Swarm Optimization*. ISTE, London, 2006.

[24] Maurice Clerc and James Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[25] Gilles Cohen, Melanie Hilario, and Antoine 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*, pages 200–211, 2004.

[26] Lior Cohen, Gil Avrahami-Bakish, Mark Last, Abraham Kandel, and Oscar Kipersztok. Real-time data mining of non-stationary data streams from sensor networks. *Information Fusion*, 9(3):344–353, 2008.

[27] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

[28] Padraig Cunningham and John Carney. Diversity versus quality in classification ensembles based on feature selection. In *Proceedings of the 11th European Conference on Machine Learning*, pages 109–116, London, UK, 2000. Springer-Verlag.

[29] Bruno Feres de Souza, Andre C. P. L. F. de Carvalho, Rodrigo Calvo, and Renato Porfirio Ishii. Multiclass SVM model selection using particle swarm optimization. In *Proceedings of the 6th International Conference on Hybrid Intelligent Systems*, pages 31–34, 2006.

[30] S. J. Delany, P. Cunningham, A. Tsymbal, and L. Coyle. A case-based technique for tracking concept drift in spam filtering. *Knowledge-Based Systems*, 18(4–5):187–195, 2005.

[31] S.J. Delany, P. Cunningham, A. Tsymbal, and L. Coyle. A case-based technique for tracking concept drift in spam filtering. In *Proceedings of the XII Applications and Innovations in Intelligent Systems*, pages 3–16, 2004.

[32] Christopher P. Diehl and Gert Cauwenberghs. SVM incremental learning, adaptation and optimization. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2685–2690, 2003.

[33] Thomas G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000.

[34] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[35] Carlotta Domeniconi and Dimitrios Gunopulos. Incremental support vector machine construction. In *Proceedings of the International Conference on Data Mining*, pages 589–592, 2001.

[36] Pedro Domingos. A unified bias-variance decomposition and its applications. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 231–238, San Francisco, CA, USA, 2000.

[37] Weilin Du and Bin Li. Multi-strategy ensemble particle swarm optimization for dynamic optimization. *Information Sciences*, 178(15):3096 – 3109, 2008.

[38] R.O. Duda, P.E. Hart, and David G. Stork. *Pattern classification - Second Edition*. Wiley Interscience, NY, 2000.

[39] Ting fan Wu, Chih-Jen Lin, and Ruby C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, 2003.

[40] Frauke Friedrichs and Christian Igel. Evolutionary tuning of multiple SVM parameters. In *Proceedings of the 12th European Symposium on Artificial Neural Networks*, pages 519–524, 2004.

[41] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Mining data streams under block evolution. *ACM SIGKDD Explorations Newsletter*, 3(2):1–10, 2002.

[42] Giorgio Giacinto and Fabio Roli. An approach to the automatic design of multiple classifier systems. *Pattern Recognition Letters*, 22(1):25–33, 2001.

[43] Adam J. Grove and Dale Schuurmans. Boosting in the limit: maximizing the margin of learned ensembles. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 692–699, Menlo Park, CA, USA, 1998.

[44] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10):993–1001, 1990.

[45] P. Henniges, E. Granger, and R. Sabourin. Factors of overtraining with fuzzy ARTMAP neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–4, 2005.

[46] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8):832–844, 1998.

[47] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, March 2002.

[48] Xiaohui Hu and Russell C. Eberhart. Adaptive particle swarm optimization: Detection and response to dynamic systems. In *Proceedings of the Congress on Evolutionary Computation*, pages 1666–1670, 2002.

[49] Chien-Ming Huang, Yuh-Jye Lee, Dennis K.J. Lin, and Su-Yun Huang. Model selection for support vector machines via uniform design. *Computational Statistics & Data Analysis*, 52(1):335–346, September 2007.

[50] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the 17th International Conference on Knowledge Discovery and Data Mining*, pages 97–106, 2001.

[51] Khaled Jabeur and Adel Guitouni. Automated learning multi-criteria classifiers for FLIR ship imagery classification. In *International Conference on Information Fusion*, pages 200–211, 2007.

[52] M. Jiang and 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, pages 158–161, 2007.

[53] Yaochu Jin and J. Branke. Evolutionary optimization in uncertain environments - A survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[54] Marcelo N. Kapp, Robert Sabourin, and Patrick Maupin. An empirical study on diversity measures and margin theory for ensembles of classifiers. In *Proceedings of the 10th International Conference on Information Fusion*, pages 1–8, 2007.

[55] Marcelo N. Kapp, Robert Sabourin, and Patrick Maupin. A dynamic model selection strategy for support vector machine classifiers. *Submitted to Pattern Recognition*, 2009.

[56] Marcelo N. Kapp, Robert Sabourin, and Patrick Maupin. A dynamic optimization approach for adaptive incremental learning in static environments. *Submitted to Information Science*, 2009.

[57] Marcelo N. Kapp, Robert Sabourin, and Patrick Maupin. A PSO-based framework for dynamic SVM model selection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1227–1234, 2009.

[58] M. G. Kelly, D. J. Hand, and N. M. Adams. The impact of changing populations on classifier performance. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, pages 367–371, 1999.

[59] J. Kennedy and R. C. Eberhart. Particle swarm intelligence. In *Proceedings of the International Conference on Neural Networks*, pages 1942–1948, 1995.

[60] James Kennedy. Some issues and practices for particle swarms. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 801–808, 2007.

[61] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:226–239, 1998.

[62] R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *Proceedings of the 17th International Conference on Machine Learning*, pages 487–494, 2000.

[63] Ralf Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligence Data Analysis.*, 8(3):281–300, 2004.

[64] Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In *Proceedings of the ICML-00, 17th International Conference on Machine Learning*, pages 487–494, 2000.

[65] Ralf Klinkenberg and Stefan Ruping. Concept drift and the importance of examples. *Text Mining - Theoretical Aspects and Applications*, pages 55–77, 2002.

[66] Albert Hung-Ren Ko, Robert Sabourin, and Alceu de Souza Britto Jr. Compound diversity functions for ensemble selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(4):659–686, 2009.

[67] Ron Kohavi and David H. Wolpert. Bias plus variance decomposition for zero-one loss functions. In Lorenza Saitta, editor, *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 275–283. Morgan Kaufmann, 1996.

[68] J. Kolter and M. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Proceeding of the International Conference on Data Mining*, pages 123–130, 2003.

[69] Ludmila Kuncheva, Marina Skurichina, and Robert P. W. Duin. An experimental study on diversity for bagging and boosting with linear classifiers. *Information Fusion*, 3(4):245–258, 2002.

[70] Ludmila I. Kuncheva. Classifier ensembles for changing environments. In *Proceedings of the 5th International Workshop on Multiple Classifier Systems*, pages 1–15, 2004.

[71] Ludmila I. Kuncheva. *Combining Pattern Classifiers*. John Wiley & Sons, Inc., New Jersey, 2004.

[72] Ludmila I. Kuncheva, James C. Bezdek, and Robert P.W. Duin. Decision templates for multiple classifier fusion: An experimental comparison. *Pattern Recognition*, 34(2):299–314, 2001.

[73] Ludmila I. Kuncheva and Christopher J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Mach. Learn.*, 51(2):181–207, 2003.

[74] Louisa Lam. Classifier combinations: Implementations and theoretical issues. In *Multiple Classifier Systems*, pages 77–86, 2000.

[75] Mihai Lazarescu, Svetha Venkatesh, and Hai Hung Bui. Using multiple windows to track concept drift. *Intelligent Data Analysis Journal*, 8(1):29–59, 2004.

[76] H.-T. Lin, C.-J. Lin, and R. C. Weng. A note on Platt's probabilistic outputs for support vector machines. Technical report, Department of Computer Science and Information Engineering, National Taiwain University, 2003.

[77] M. Maloof. Incremental rule learning with partial instance memory for changing concepts. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2764–2769, 2003.

[78] M. Maloof and R. Michalski. Selecting examples for partial memory learning. *Machine Learning*, 41:27–52, 2000.

[79] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. Machine learning. Neural and Statistical Classification. ftp.ncc.up.pt/pub/statlog/.

[80] Pabitra Mitra, C. A. Murthy, and Sankar K. Pal. Data condensation in large databases by incremental learning with support vector machines. In *Proceedings of the 15th International Conference on Pattern Recognition*, pages 708–711, 2000.

[81] H.S. Mohammed, J. Leander, M. Marbach, and R. Polikar. Comparison of ensemble techniques for incremental learning of new concept classes under hostile non-stationary environments. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 4838–4844, 2006.

[82] Michael Muhlbaier and Robi Polikar. An ensemble approach for incremental learning in nonstationary environments. In *International Workshop on Multiple Classifiers Systems*, pages 490–500, 2007.

[83] A. Nickabadi, M. M. Ebadzadeh, and R. Safabakhsh. DNPSO: A dynamic niching particle swarm optimizer for multi-modal optimization. In *Proceedings of the IEEE Congress on Computational Intelligence*, pages 26–32, 2008.

[84] L. S. Oliveira. *Automatic Recognition of Handwritten Numerical Strings*. PhD thesis, École de Technologie Supérieure, Canada, 2003.

[85] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and 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):903–930, 2003.

[86] Nikunj Chandrakant Oza. *Online Ensemble Learning*. PhD thesis, University of California, California, Berkeley, 2001.

[87] S. Ozawa, S. Pang, and N. Kasabov. Incremental learning of chunk data for online pattern classification systems. *IEEE Transactions on Neural Networks*, 19(6):1061–1074, 2008.

[88] D. Parikh and R. Polikar. An ensemble-based incremental learning approach to data fusion. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(2):437–450, April 2007.

[89] Y. Park and J. Sklansky. Automated design of linear tree classifiers. *Pattern Recognition*, 23:1393–1412, 1990.

[90] D. Partridge and W. Krzanowski. Design of effective neural network ensembles for image classification purposes. *Software diversity: practical statistics for its measurement and exploitation*, 39(10):707–717, 1997.

[91] Binbin Peng, Liu Wenyin, Yin Liu, and Guanglin Huang. An SVM-based incremental learning algorithm for user adaptation of sketch recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(8):1529–1550, 2004.

[92] J.C. Platt. *Advances in Kernel Methods - Support Vector Learning*. MIT Press, New York, 1999.

[93] Robi Polikar, Lalita Udpa, Satish S. Udpa, and Vasant Honavar. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 31(4):497–508, 2001.

[94] Liva Ralaivola and Florence d'Alché Buc. Incremental support vector machine learning: A local approach. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 322–329, 2001.

[95] Stefan Rüping. Incremental learning with support vector machines. In *Proceedings of the IEEE International Conference on Data Mining*, pages 641–642, 2001.

[96] Dymitr Ruta and Bogdan Gabrys. A theoretical analysis of the limits of majority voting errors for multiple classifier systems. *Pattern Anal. Appl.*, 5(4):333–350, 2002.

[97] Dymitr Ruta and Bogdan Gabrys. Classifier selection for majority voting. *Information Fusion*, 6(1):63 – 81, 2005.

[98] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[99] Robert E. Schapire, Yoav Freund, Peter Barlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 322–330, San Francisco, CA, USA, 1997.

[100] J. C. Schlimmer and R. H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[101] D. Sculley and Gabriel M. Wachman. Relaxed online svms for spam filtering. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 415–422. ACM, 2007.

[102] A. J. C. Sharkey. *Combining Artificial Neural Nets : Ensemble and Modular Multi-Net Systems (Perspectives in Neural Computing)*. Springer Verlag, 1999.

[103] Alistair Shilton, M. Palaniswami, Daniel Ralph, and Ah Chung Tsoi. Incremental training of support vector machines. *IEEE Transactions on Neural Networks*, 16:114–131, 2005.

[104] D. B. Skalak. The sources of increased accuracy for two proposed boosting algorithms. In *Working Notes of the AAAI Workshop on Integrating Multiple Learned Models*, pages 120–125, 1996.

[105] K. O. Stanley. Learning concept drift with a committee of decision trees. Technical Report AI-03-302, Department of Computer Sciences, University of Texas at Austin, USA, 2003.

[106] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the 17th International Conference on Knowledge Discovery and Data Mining*, pages 377–382, 2001.

[107] Thorsten Suttorp and Christian Igel. Multi-objective optimization of support vector machines. In *Multi-Objective Machine Learning*, volume 16 of *Studies in Computational Intelligence*, pages 199–220. 2006.

[108] N. Syed, H. Liu, and K. Sung. Incremental learning with support vector machines. In *Proceedings of the International Conference on Artificial Intelligence*, 1999.

[109] Nadeem Ahmed Syed, Huan Liu, and Kah Kay Sung. Handling concept drifts in incremental learning with support vector machines. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, pages 317–321, 1999.

[110] E. K. Tang, P. N. Suganthan, and X. Yao. An analysis of diversity measures. *Machine Learning*, 65(1):247–271, 2006.

[111] Alexey Tsymbal, Mykola Pechenizkiy, and Padraig Cunningham. Dynamic integration with random forests. In *ECML*, pages 801–808, 2006.

[112] Alexey Tsymbal, Mykola Pechenizkiy, Padraig Cunningham, and Seppo Puuronen. Handling local concept drift with dynamic integration of classifiers: Domain of antibiotic resistance in nosocomial infections. In *Proceedings of the 19th IEEE Symposium on Computer-Based Medical Systems*, pages 679–684, 2006.

[113] Alexey Tsymbal, Mykola Pechenizkiy, Padraig Cunningham, and Seppo Puuronen. Dynamic integration of classifiers for handling concept drift. *Information Fusion*, 9(1):56–68, 2008.

[114] Aydin Ulas, Murat Semerci, Olcay Taner Yildiz, and Ethem Alpaydin. Incremental construction of classifier and discriminant ensembles. *Information Sciences*, 179(9):1298–1318, 2009.

[115] G. Valentini. *Ensemble methods based on bias-variance analysis*. PhD thesis, University of Genova, Genova, Italy, 2003.

[116] Giorgio Valentini. An experimental bias-variance analysis of SVM ensembles based on resampling techniques. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(6):1252–1271, 2005.

[117] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, NY, 1995.

[118] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, NY, 1998.

[119] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235, 2003.

[120] Peng Wang, Haixun Wang, Xiaochen Wu, Wei Wang, and Baile Shi. On reducing classifier granularity in mining concept-drifting data streams. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pages 474–481, 2005.

[121] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.

[122] Terry Windeatt. Vote counting measures for ensemble classifiers. *Pattern Recognition*, 36(12):2743–2756, 2003.

[123] Rong Xiao, Jicheng Wang, and Fayan Zhang. An approach to incremental SVM learning algorithm. In *Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence*, pages 268–273, 2000.

[124] Hwanjo Yu, Jiong Yang, and Jiawei Han. Classifying large data sets using SVMs with hierarchical clusters. In *Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 306–315, 2003.

[125] Gabriele Zenobi and Padraig Cunningham. Using diversity in preparing ensembles of classifiers based on different feature subsets to minimize generalization error. In *Proceedings of the 12th European Conference on Machine Learning*, pages 576–587, 2001.