Journal of Machine Learning Research 10 (2009) 405-440 Submitted 1/07; Revised 8/08; Published 2/09

Particle Swarm Model Selection

Hugo Jair Escalante HUGOJAIR@CCC.INAOEP.MX
Manuel Montes MMONTESG@INAOEP.MX
Luis Enrique Sucar ESUCAR@INAOEP.MX
Department of Computational Sciences
National Institute of Astrophysics, Optics and Electronics
Puebla, México, 72840

Editor: Isabelle Guyon and Amir Saffari

Abstract

This paper proposes the application of particle swarm optimization (*PSO*) to the problem of *full model selection, FMS*, for classification tasks. *FMS* is defined as follows: given a pool of preprocessing methods, feature selection and learning algorithms, to select the combination of these that obtains the lowest classification error for a given data set; the task also includes the selection of hyperparameters for the considered methods. This problem generates a vast search space to be explored, well suited for stochastic optimization techniques. *FMS* can be applied to any classification domain as it does not require domain knowledge. Different model types and a variety of algorithms can be considered under this formulation. Furthermore, competitive yet simple models can be obtained with *FMS*. We adopt *PSO* for the search because of its proven performance in different problems and because of its simplicity, since neither expensive computations nor complicated operations are needed. Interestingly, the way the search is guided allows *PSO* to avoid overfitting to some extend. Experimental results on benchmark data sets give evidence that the proposed approach is very effective, despite its simplicity. Furthermore, results obtained in the framework of a model selection challenge show the competitiveness of the models selected with *PSO,* compared to models selected with other techniques that focus on a single algorithm and that use domain knowledge.

Keywords: full model selection, machine learning challenge, particle swarm optimization, experimentation, cross validation

1. Introduction

Model selection is the task of picking the model that best describes a data set (Hastie et al., 2001). Since the phrase *describing a data set* can be interpreted in several different ways, the model selection task can denote diverse related problems, including: variable and feature selection (Bengio and Chapados, 2003; Guyon et al., 2006a; Guyon and Elisseeff, 2003), system identification (Voss and Feng, 2002; Nelles, 2001), parameter-hyperparameter optimization (Guyon et al., 2006b; Kim et al., 2002; Hastie et al., 2001; Cawley and Talbot, 2007b; Escalante et al., 2007), and discretization (Boullé, 2007; Hue and Boullé, 2007). In this paper we give a broader interpretation to this task and call it *full model selection* (*FMS*). The *FMS* problem consists on the following: given a pool of preprocessing methods, feature selection and learning algorithms, select the combination of these that obtains the lowest classification error for a given data set. This task also includes the selection of hyperparameters for the considered methods, resulting in a vast search space that is well suited for stochastic optimization techniques.

Adopting a broader interpretation to the model selection problem allows us to consider different model types and a variety of methods, in contrast to techniques that consider a single model type (i.e., either learning algorithm or feature selection method, but not both) and a single method (e.g., neural networks). Also, since neither prior domain knowledge nor machine learning knowledge is required, *FMS* can be applied to any classification problem without modification. This is a clear advantage over ad-hoc model selection methods that perform well on a single domain or that work for a fixed algorithm. This will help users with limited machine learning knowledge, since *FMS* can be seen as a black-box tool for model selection. Machine learning experts can also benefit from this approach. For example, several authors make use of search strategies for the selection of *candidate models* (Lutz, 2006; Boullé, 2007; Reunanen, 2007; Wichard, 2007), the *FMS* approach can be adopted for obtaining such candidate models.

One could expect a considerable loss of accuracy by gaining generality. However, this is not the case of the proposed approach since in international competitions it showed comparable performance to other techniques that were designed for a single algorithm (i.e., doing hyperparameter optimization) and to methods that took into account domain knowledge (Guyon et al., 2008). The main drawback is the computational cost to explore the vast search space, particularly for large data sets. But, we can gain efficiency without a significant loss in accuracy, by adopting a random subsampling strategy, see Section 4.3. The difficult interpretability of the selected models is another limitation of the proposed approach. However, naive users may accept to trade interpretably for ease-of-use, while expert users may gain insight in the problem at hand by analyzing the structure of the selected model (type of preprocessing chosen, number of features selected, linearity or non-linearity of the predictor).

In this paper, we propose to use particle swarm optimization (*PSO*) for exploring the full-models search space. *PSO* is a bio-inspired search technique that has shown comparable performance to that of evolutionary algorithms (Angeline, 1998; Reyes and Coello, 2006). Like evolutionary algorithms, *PSO* is useful when other techniques such as gradient descend or direct analytical discovery are not applicable. Combinatoric and real-valued optimization problems in which the optimization surface possesses many locally optimal solutions, are well suited for swarm optimization. In *FMS* it must be found the best combination of methods (for preprocessing, feature selection and learning) and simultaneously optimizing real valued functions (finding pseudo-optimal parameters for the considered methods), in consequence, the application of *PSO* is straightforward.

The methodological differences between swarm optimization and evolutionary algorithms have been highlighted by several authors (Angeline, 1998; Kennedy and Eberhart, 1995, 2001). However a difference in performance has not been demonstrated in favor of either method. Such demonstration would be a difficult task because no black-box stochastic optimization algorithm can outperform another over all optimization problems, not even over random search (Wolpert and Macready, 1997; van den Bergh, 2001). We selected *PSO* instead of evolutionary algorithms because of its simplicity and generality as no important modification was made for applying it to *FMS*. *PSO* is easier to implement than evolutionary algorithms because it only involves a single operator for updating solutions. In contrast, evolutionary algorithms require a particular representation and specific methods for cross-over, mutation, speciation and selection. Furthermore, *PSO* has been found to be very effective in a wide variety of applications, being able to produce good solutions at a very low computational cost (Gudise and Venayagamoorthy, 2003; Hernández et al., 2004; Xiaohui et al., 2003; Yoshida et al., 2001; Robinson, 2004; Kennedy and Eberhart, 2001; Reyes and Coello, 2006).

PSO is compared to pattern search (*PS*) in order to evaluate the added value of using the swarm strategy instead of another intensive search method. We consider *PS* among other search techniques because of its simplicity and proved performance in model selection (Momma and Bennett, 2002; Bi et al., 2003; Dennis and Torczon, 1994). Cross validation (*CV*) is used in both techniques for assessing the *goodness* of models. Experimental results in benchmark data give evidence that both *PSO* and *PS* are effective strategies for *FMS*. However, it was found that *PSO* outperforms *PS*, showing better convergence behavior and being less prone to overfitting. Furthermore, the proposed method was evaluated in the context of a model selection competition in which several specialized and prior-knowledge based methods for model selection were used. Models selected with *PSO* were always among the top ranking models through the different stages of the challenge (Guyon et al., 2006c, 2007, 2008; Escalante et al., 2007). During the challenge, our best entry was ranked 8^{th} over all ranked participants, 5^{th} among the methods that did not use domain knowledge and 2^{nd} among the methods that used the software provided by the organizers (Guyon et al., 2006c, 2007, 2008). In this paper we outperform the latter entry while reducing the computational burden by using a subsampling strategy; our best entry is currently the top-ranked one among models that do not use prior domain knowledge and 2^{nd} over all entries, see Section 4.3.

PSO has been widely used for parameter selection in supervised learning (Kennedy and Eberhart, 1995, 2001; Salerno, 1997; Gudise and Venayagamoorthy, 2003). However, parameter selection is related with the first level of inference in which, given a learning algorithm, the task is to find parameters for such algorithm in order to describe the data. For example, in neural networks the adjustment of weights between units according to some training data is a parameter selection problem. Hyperparameter optimization, on the other hand, is related with the second level of inference, that is, finding parameters for the methods that in turn should determine parameters for describing the data. In the neural network example, selecting the optimal number of units, the learning rate, and the number of epochs for training the network is a hyperparameter optimization problem. *FMS* is capable of operating across several levels of inference by simultaneously performing feature selection, preprocessing and classifier selection, and hyperparameter optimization for the selected methods. *PSO* has been used for hyperparameter optimization by Voss and Feng (2002), however they restricted the problem to linear systems for univariate data sets, considering one hundred data observations. In this paper we are going several steps further: we applied *PSO* for *FMS* considering non-linear models in multivariate data sets with a large number of observations.

Parallel to this work, Gorissen et al. used genetic algorithms for meta-model selection in regression tasks (Gorissen, 2007; Gorissen et al., 2008), a similar approach that emerged totally independently to our proposal. One should note, however, that this method has been used for a different task in low dimensional data sets; most results are reported for 2D data. Even with this dimensionality the method has been run for only a few iterations with a small population size. Their use of a genetic algorithm required the definition of specific operators *for each of the considered models*. Gorissen et al. considered seven different models (including neural networks and kernel methods) that required of 18 different genetic operators for creation, mutation and cross-over (Gorissen, 2007; Gorissen et al., 2008). Additionally, general operators for speciation and selection were also defined. In the present work a single operator was used for updating solutions, regardless of the considered models. This clearly illustrates the main advantage of *PSO* over genetic algorithms, namely generality and simplicity.

The main contribution of this work is experimental: we provide empirical evidence indicating that by using *PSO* we were able to perform intensive search over a huge space and succeeded in selecting competitive models without significantly overfitting. This is due to the way the search is guided in *PSO*: performing a broad search around promising solutions but not overdoing in terms of really fine optimization. This sort of search is known to help avoiding overfitting by *undercomputing* (Dietterich, 1995). Experimental results supported by some a posteriori analysis give evidence of the validity of our approach. The way we approached the model selection problem and the use of a stochastic-search strategy are also contributions. To the best of our knowledge there are no similar works that consider the *FMS* problem for classification tasks.

The rest of this paper is organized as follows. In the next section we describe the general *PSO* algorithm. In Section 3, we describe the application of *PSO* to *FMS*. Section 4 presents experimental results in benchmark data; comparing the performance of *PSO* to that of *PS* in *FMS* and analyzing the performance of *PSO* under different parameter settings; also, are described the results obtained in the framework of a model selection competition. In Section 5, we analyze mechanisms in *PSMS* that allow to select competitive models without overfitting the data. Finally, in Section 6, we present the conclusions and outline future research directions.

2. Particle Swarm Optimization (*PSO*)

PSO is a population-based search algorithm inspired by the behavior of biological communities that exhibit both individual and social behavior; examples of these communities are flocks of birds, schools of fishes and swarms of bees. Members of such societies share common goals (e.g., finding food) that are realized by exploring its environment while interacting among them. Proposed by Kennedy and Eberhart (1995), *PSO* has become an established optimization algorithm with applications ranging from neural network training (Kennedy and Eberhart, 1995; Salerno, 1997; Kennedy and Eberhart, 2001; Gudise and Venayagamoorthy, 2003; Engelbrecht, 2006) to control and engineering design (Hernández et al., 2004; Xiaohui et al., 2003; Yoshida et al., 2001; Robinson, 2004). The popularity of *PSO* is due in part to the simplicity of the algorithm (Kennedy and Eberhart, 1995; Reyes and Coello, 2006; Engelbrecht, 2006), but mainly to its effectiveness for producing good results at a very low computational cost (Gudise and Venayagamoorthy, 2003; Kennedy and Eberhart, 2001; Reyes and Coello, 2006). Like evolutionary algorithms, *PSO* is appropriate for problems with immense search spaces that present many local minima.

In *PSO* each solution to the problem at hand is called a particle. At each time t, each particle, i, has a position $\mathbf{x}_i^t =< x_{i,1}^t, x_{i,2}^t, \ldots, x_{i,d}^t >$ in the search space; where d is the dimensionality of the solutions. A set of particles $\mathbf{S} = \{\mathbf{x}_1^t, \mathbf{x}_2^t, \ldots, \mathbf{x}_m^t\}$ is called a swarm. Particles have an associated velocity value that they use for *flying* (exploring) through the search space. The velocity of particle i at time t is given by $\mathbf{v}_i^t =< v_{i,1}^t, v_{i,2}^t, \ldots, v_{i,d}^t >$, where $v_{i,k}^t$ is the velocity for dimension k of particle i at time t. Particles adjust their flight trajectories by using the following updating equations:

$$v_{i,j}^{t+1} = W \times v_{i,j}^t + c_1 \times r_1 \times (p_{i,j} - x_{i,j}^t) + c_2 \times r_2 \times (p_{g,j} - x_{i,j}^t), \tag{1}$$

$$x_{i,j}^{t+1} = x_{i,j}^t + v_{i,j}^{t+1} \tag{2}$$

where $p_{i,j}$ is the value in dimension j of the best solution found so far by particle i; $\mathbf{p}_i =< p_{i,1}, \ldots, p_{i,d} >$ is called personal best. $p_{g,j}$ is the value in dimension j of the best particle found so far in the swarm ($\mathbf{S}$); $\mathbf{p}_g =< p_{g,1}, \ldots, p_{g,d} >$ is considered the leader particle. Note that through

$\mathbf{p}_i$ and $\mathbf{p}_g$ each particle i takes into account individual (local) and social (global) information for updating its velocity and position. In that respect, $c_1, c_2 \in \mathbb{R}$ are constants weighting the influence of local and global best solutions, respectively. $r_1, r_2 \sim U[0,1]$ are values that introduce randomness into the search process. W is the so called inertia weight, whose goal is to control the impact of the past velocity of a particle over the current one, influencing the local and global exploration abilities of the algorithm. This is one of the most used improvements of *PSO* for enhancing the rate of convergence of the algorithm (Shi and Eberhart, 1998, 1999; van den Bergh, 2001). For this work we considered an adaptive inertia weight specified by a triplet $\mathbf{W} = (w_{start}, w_f, w_{end})$; where w_{start} and w_{end} are the initial and final values for W, respectively, and w_f indicates the fraction of iterations in which W is decreased. Under this setting W is decreased by $W = W - wdec$ from iteration $t = 1$ (where $W = w_{start}$) up to iteration $t = I \times w_f$ (after which $W = w_{end}$); where $w_{dec} = \frac{w_{start} - w_{end}}{I \times w_f}$ and I is the maximum number of iterations. This setting allows us to explore a large area at the start of the optimization, when W is large, and to slightly refine the search later by using a smaller inertia weight (Shi and Eberhart, 1998, 1999; van den Bergh, 2001).

An adaptive W can be likened to the temperature parameter in simulated annealing (Kirkpatrick et al., 1983); this is because, in essence, both parameters influence the global and local exploration abilities of their respective algorithms, although in different ways. A constant W is analogous to the momentum parameter p in gradient descend with momentum term (Qian, 1999), where weights are updated by considering both the current gradient and the weight change of the previous step (weighed by p). Interestingly, the inertia weight is also similar to the weight-decay constant (γ) used in machine learning to prevent overfitting. In neural networks the weights are decreased by $(1 - \gamma)$ in each learning step, which is equivalent to add a penalty term into the error function that encourages the magnitude of the weights to decay towards zero (Bishop, 2006; Hastie et al., 2001); the latter penalizes complex models and can be used to obtain sparse solutions (Bishop, 2006).

The pseudo code of the *PSO* algorithm considered in this work is shown in Algorithm 1; default recommended values for the *FMS* problem are shown as well (these values are based on the analysis of Section 2.1 and experimental results from Section 4.2). The swarm is randomly initialized, considering restrictions on the values that each dimension can take. Next, the *goodness* of each particle is evaluated and $\mathbf{p}_g$, $\mathbf{p}_{1,\dots,m}$ are initialized. Then, the iterative *PSO* process starts, in each iteration: i) the velocities and positions of each particle in every dimension are updated according to Equations (1) and (2); ii) the *goodness* of each particle is evaluated; iii) $\mathbf{p}_g$ and $\mathbf{p}_{1,\dots,m}$ are updated, if needed; and iv) the inertia weight is decreased. This process is repeated until either a maximum number of iterations is reached or a minimum fitness value is obtained by a particle in the swarm (we used the first criterion for *FMS*); eventually, an (locally) optimal solution is found.

A fitness function is used to evaluate the aptitude (*goodness*) of candidate solutions. The definition of a specific fitness function depends on the problem at hand; in general it must reflect the proximity of the solutions to the optima. A fitness function $F : \Psi \rightarrow \mathbb{R}$, where Ψ is the space of particles positions, should return a scalar $f_{\mathbf{x}_i}$ for each particle position $\mathbf{x}_i$, indicating how far particle i is from the optimal solution to the problem at hand. For *FMS* the goal is to improve classification accuracy of full models. Therefore, any function F, which takes as input a model and returns an estimate of classification performance, is suitable (see Section 3.3).

Note that in Equation (1) every particle in the swarm knows the best position found so far by any other particle within the swarm, that is $\mathbf{p}_g$. Under this formulation a fully-connected swarm-topology is considered in which every member knows the leader particle. This topology has shown to converge faster than any other topology (Kennedy and Mendes, 2002; Reyes and Coello, 2006;

Algorithm 1 Particle swarm optimization.

```
**Require: [Default recommended values for FMS]**
– $\mathbf{c}_1, \mathbf{c}_2$: individual/social behavior weights; $[\mathbf{c}_1 = \mathbf{c}_2 = 2]$
– $\mathbf{m}$: swarm size; $[\mathbf{m} = 5]$
– $\mathbf{I}$: number of iterations; $[\mathbf{I} = 50]$
– $\mathbf{F}(\Psi \rightarrow \mathbb{R})$: fitness function; $[\mathbf{F}(\Psi \rightarrow \mathbb{R}) = 2-$fold $CV\ BER$ ]
– $\mathbf{W}$: Inertia weight $\mathbf{W}$ = (1.2, 0.5, 0.4)
Set decrement factor for $W$ ($w_{dec} = \frac{w_{start} - w_{end}}{I \times w_f}$)
Initialize swarm ($\mathbf{S} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$)
Compute $f_{\mathbf{x}_{1,\ldots,m}} = F(\mathbf{x}_{1,\ldots,m})$ (Section 3.3)
Locate leader ($\mathbf{p}_g$) and set personal bests ($\mathbf{p}_{1,\ldots,m} = \mathbf{x}_{1,\ldots,m}$)
$t = 1$
**while** $t < I$ **do**
  **for all** $\mathbf{x}_i \in \mathbf{S}$ **do**
    Calculate velocity $\mathbf{v}_i$ for $\mathbf{x}_i$(Equation (1))
    Update position of $\mathbf{x}_i$ (Equation (2))
    Compute $f_{\mathbf{x}_i} = F(\mathbf{x}_i)$
    Update $\mathbf{p}_i$ (*if necessary*)
  **end for**
  Update $\mathbf{p}_g$ (*if necessary*)
  **if** $t < \lfloor I \times w_f \rfloor$ **then**
    $W = W - w_{dec}$
  **end if**
  $t^{++}$
**end while**
**return** $\mathbf{p}_g$
```

Kennedy and Eberhart, 2001; Engelbrecht, 2006). With this topology, however, the swarm is prone to converge to local minima. We tried to overcome this limitation by using the adaptive inertia weight, W.

2.1 PSO Parameters

Selecting the best parameters (W, c_1, c_2, m, I) for *PSO* is another model selection task. In the application of *PSO* for *FMS* we are dealing with a very complex problem lying in the third level of inference. Fortunately, several empirical and theoretical studies have been performed about the parameters of *PSO* from which useful information can be obtained (Shi and Eberhart, 1998, 1999; Kennedy and Mendes, 2002; Reyes and Coello, 2006; Ozcan and Mohan, 1998; Clerc and Kennedy, 2002; van den Bergh, 2001). In the rest of this section the *PSO* parameters are analyzed in order to select appropriate values for *FMS*. Later on, in Section 4.2, results of experiments with *PSO* for *FMS* under different parameter settings are presented.

We will start analyzing $c_1, c_2 \in \mathbb{R}$, the weighting factors for individual and social behavior. It has been shown that convergence is guaranteed[1] for *PSO* under certain values of c_1, c_2 (van den Bergh, 2001; Reyes and Coello, 2006; Ozcan and Mohan, 1998; Clerc and Kennedy, 2002). Most convergence studies have simplified the problem to a single one-dimensional particle, setting $\phi = c_1 \times r_1 + c_2 \times r_2$, and $\mathbf{p}_g$ and $\mathbf{p}_i$ constant (van den Bergh, 2001; Reyes and Coello, 2006; Ozcan and Mohan, 1998; Clerc and Kennedy, 2002). A complete study including the inertia weight is carried out by van den Bergh (2001). In agreement with this study the value $\phi < 3.8$ guarantees eventual convergence of the algorithm when the inertia weight W is close to 1.0. For the experiments in this work it was fixed $c_1 = c_2 = 2$, with these values we have $\phi < 3.8$ with high probability $P(\phi < 3.8) \approx 0.9$, given the uniformly distributed numbers r_1, r_2 (van den Bergh, 2001). The configuration $c_1 = c_2 = 2$ also has been proven, empirically, to be an effective choice for these parameters (Kennedy and Eberhart, 1995; Shi and Eberhart, 1998, 1999; Kennedy and Mendes, 2002).

Note, however, the restriction that W remains close to 1.0. In our experiments we considered a value of $W = 1.2$ at the beginning of the search, decreasing it during 50% (i.e., $w_f = 0.5$) of the *PSO* iterations up to the value $W = 0.4$. In consequence, it is possible that at the end of the search process the swarm may show divergent behavior. In practice, however, the latter configuration for W resulted very useful for *FMS*, see Section 4. This selection of W should not be surprising since even the configuration $\mathbf{W} = (1, 1, 0)$ has obtained better results than using a fixed W in empirical studies (Shi and Eberhart, 1998). Experimental results with different configurations of $\mathbf{W}$ for *FMS* give evidence that the configuration we selected can provide better models than using constant $\mathbf{W}$, see Section 4.2; although further experiments need to be performed in order to select the best $\mathbf{W}$ configuration for *FMS*.

With respect to m, the size of the swarm, experimental results suggest that the size of the population does not damage the performance of *PSO* (Shi and Eberhart, 1999), although slightly better results have been obtained with a large value of m, (Shi and Eberhart, 1999). Our experimental results in Section 4.2 confirm that the selection of m is not crucial, although (contrary to previous results) slightly better models are obtained by using a small swarm size. The latter is an important result for *FMS* because using a small number of particles reduces the number of fitness function evaluations, and in consequence the computational cost of the search.

Regarding the number of iterations, to the best of our knowledge, there is no work on the subject. This is mainly due to the fact that this issue depends mostly on the complexity of the problem at hand and therefore a general rule cannot be derived. For *FMS* the number of iterations should not be large to avoid oversearching (Dietterich, 1995). For most experiments in this paper we fixed $I = 100$. However, experimental results in Section 4.2 show that by running *PSO* for a smaller number of iterations is enough to obtain models of the same, and even better, generalization performance. This gives evidence that early stopping can be an useful mechanism to avoid overfitting, see Section 5.

3. Particle Swarm Model Selection

Since one of the strong advantages of *PSO* is its simplicity, its application to *FMS* is almost direct. Particle swarm full model selection (hereafter *PSMS*, that is the application of *PSO* to *FMS*) is

1. Note that in *PSO* we say that the swarm converges iff $\lim_{t \to \inf} \mathbf{p}_{gt} = \mathbf{p}$, where $\mathbf{p}$ is an arbitrary position in the search space and t indexes iterations of *PSO*. Since $\mathbf{p}$ refers to an arbitrary position, this definition does not mean either convergence to local or global optimum, but convergence to the global best position in the swarm (van den Bergh, 2001; Reyes and Coello, 2006; Engelbrecht, 2006).

ID	Object name	F	Hyperparameters	Description
1	*Ftest*	*FS*	$f_{max}, w_{min}, p_{val}, fdr_{max}$	Feature ranking according the F-statistic
2	*Ttest*	*FS*	$f_{max}, w_{min}, p_{val}, fdr_{max}$	Feature ranking according the T-statistic
3	*aucfs*	*FS*	$f_{max}, w_{min}, p_{val}, fdr_{max}$	Feature ranking according to the AUC criterion
4	*odds-ratio*	*FS*	$f_{max}, w_{min}, p_{val}, fdr_{max}$	Feature ranking according to the odds ratio statistic
5	*relief*	*FS*	$f_{max}, w_{min}, k_{num}$	Relief ranking criterion
6	*rffs*	*FS*	f_{max}, w_{min}	Random forest used as feature selection filter
7	*svcrfe*	*FS*	f_{max}	Recursive feature elimination filter using svc
8	*Pearson*	*FS*	$f_{max}, w_{min}, p_{val}, fdr_{max}$	Feature ranking according to the Pearson correlation coef.
9	*ZFilter*	*FS*	f_{max}, w_{min}	Feature ranking according to a heuristic filter
10	*gs*	*FS*	f_{max}	Forward feature selection with Gram-Schmidt orth.
11	*s2n*	*FS*	f_{max}, w_{min}	Signal-to-noise ratio for feature ranking
12	$pc-extract$	*FS*	f_{max}	Extraction of features with *PCA*
1	*normalize*	*Pre*	center	Normalization of the lines of the data matrix
2	*standardize*	*Pre*	center	Standardization of the features
3	$shift-scale$	*Pre*	$take_{log}$	Shifts and scale data
1	*bias*	*Pos*	none	Finds the best threshold for the output of the classifiers

Table 1: Feature selection (*FS*), preprocessing (*Pre*) and postprocessing (*Pos*) objects available in *CLOP*. A brief description of the methods and their hyperparameters is presented.

described by the pseudocode in Algorithm 1, in this section are presented additional details about *PSMS*: first we describe the pool of methods considered in this work; then, we describe the representation of particles and the fitness function used; finally, we briefly discuss complexity issues. The code of *PSMS* is publicly available from the following website `http://ccc.inaoep.mx/~hugojair/code/psms/`.

3.1 The Challenge Learning Object Package

In order to implement *PSMS* we need to define the models search space. For this purpose we consider the set of methods in a machine learning toolbox from which full models can be generated. Currently, there are several machine learning toolboxes, some of them publicly available (Franc and Hlavac, 2004; van der Heijden et al., 2004; Wichard and Merkwirth, 2007; Witten and Frank, 2005; Saffari and Guyon, 2006; Weston et al., 2005); even there is a track of this journal (*JMLR*) dedicated to machine learning software. This is due to the increasing interest from the machine learning community in the dissemination and popularization of this research field (Sonnenburg, 2006). The *Challenge Learning Object Package*[2] (*CLOP*) is one of such development kits distributed under the GNU license (Saffari and Guyon, 2006; Guyon et al., 2006c, 2007, 2008). *CLOP* is a *Matlab*^{R} toolbox with implementations of feature-variable selection methods and machine learning algorithms (*CLOP* also includes the *PSMS* implementation used in this work). The list of available preprocessing, feature selection and postprocessing methods in the *CLOP* toolbox is shown in Table 1; a description of the learning algorithms available in *CLOP* is presented in Table 2. One should note that this version of *CLOP* includes the methods that best performed in a model selection competition (Guyon et al., 2008; Cawley and Talbot, 2007a; Lutz, 2006).

2. See `http://clopinet.com/CLOP/`.

ID	Object name	Hyperparameters	Description
1	*zarbi*	none	Linear classifier
2	*naive*	none	Naïve Bayes
3	*klogistic*	none	Kernel logistic regression
4	*gkridke*	none	Generalized kridge (VLOO)
5	*logitboost*	units number, shrinkage, depth	Boosting with trees (R)
6	*neural*	units number, shrinkage, maxiter, balance	Neural network (Netlab)
7	*svc*	shrinkage, kernel parameters (coef0, degree, gamma)	SVM classifier
8	*kridge*	shrinkage, kernel parameters (coef0, degree, gamma)	Kernel ridge regression
9	*rf*	units number, balance, mtry	Random forest (R)
10	*lssvm*	shrinkage, kernel parameters (coef0, degree, gamma), balance	Kernel ridge regression

Table 2: Available learning objects with their respective hyperparameters in the CLOP package.

In consequence, for *PSMS* the pool[3] of methods to select from are those methods described in Tables 1 and 2. In *CLOP* a typical model consists of the *chain*, which is a grouping object that allows us to perform serial concatenation of different methods. A chain may include combinations of (several/none) feature selection algorithm(s) followed by (several/none) preprocessing method(s), in turn followed by a learning algorithm and finally (several/none) postprocessing algorithm(s). For example, the model given by:
chain$\{$*gs(*$fmax = 8$*),standardize(center=1),neural(units=10,s=0.5,balance=1,iter=10)*$\}$
uses *gs* for feature selection, *standardization* of data and a balanced *neural network* classifier with 10 hidden units, learning rate of 0.5, and trained for 10 iterations. In this work chain objects that include methods for preprocessing, feature selection and classification are considered full-models. Specifically, we consider models with at most one feature selection method, but allowing to perform preprocessing before feature selection and viceversa, see Section 3.2. The bias method was used as postprocessing in every model tested to set an optimal threshold in the output of the models in order to minimize their error. The search space in *FMS* is given by all the possible combinations of methods and hyperparameters; an infinite search space due to the real valued parameters.

3.2 Representation

In *PSO* each potential solution to the problem at hand is considered a particle. Particles are represented by their position, which is nothing but a $d-$dimensional numerical vector (d being the dimensionality of the solution). In *FMS* potential solutions are full-models, in consequence, for *PSMS* we need a way to codify a full-model by using a vector of numbers. For this purpose we propose the representation described in Equation (3), the dependence on time (t) is omitted for clarity.

$$\mathbf{x}_i = < x_{i,pre}, y_{i,1 \ldots N_{pre}}, x_{i,fs}, y_{i,1,\ldots N_{fs}}, x_{i,sel}, x_{i,class}, y_{i,1,\ldots N_{class}} > \tag{3}$$

where $x_{i,pre} \in \{1,\ldots,8\}$ represents a combination of preprocessing methods. Each combination is represented by a binary vector of size 3 (i.e., the number of preprocessing methods considered), there are $2^3 = 8$ possible combinations. Each element of the binary vector represents a single preprocessing method; if the value of the k^{th} element is set to 1 then the preprocessing method with *ID* = k is used (see Table 1). For example, the first combination $< 0,0,0 >$ means *no preprocessing*; while the seventh $< 1,1,0 >$ means that this model ($\mathbf{x}_i$) uses *normalization* and *standardization* as

3. Notice that the *CLOP* package includes also the spider package (Weston et al., 2005) which in turn includes other implementations of learning algorithms and preprocessing methods. However, in this work we only used *CLOP* objects.

preprocessing. $y_{i,1\ldots N_{pre}}$ codify the hyperparameters for the selected combination of preprocessing methods, $N_{pre} = 3$ because each preprocessing method has a single hyperparameter; note that the order of the preprocessing methods is fixed (i.e., *standardization* can never be performed before *normalization*), in the future we will relax this constraint. $x_{i,fs} \in \{0,\ldots,12\}$ represents the *ID* of the feature selection method used by the model (see Table 1), and $y_{i,1\ldots N_{fs}}$ its respective hyperparameters; N_{fs} is set to the maximum number of hyperparameters that any feature selection method can take. $x_{i,sel}$ is a binary variable that indicates whether preprocessing should be performed before feature selection or viceversa. $x_{i,class} \in \{1,\ldots,10\}$ represents the classifier selected and $y_{i,1,\ldots N_{class}}$ its respective hyperparameters; N_{class} is the maximum number of hyperparameters that a classifier can take. This numerical codification must be decoded and used with the *chain* grouping object for obtaining a full-model from a particle position $\mathbf{x}_i$. Note that the dimensionality of each particle is $d = 1 + N_{pre} + 1 + N_{fs} + 1 + 1 + N_{class} = 16$.

3.3 Fitness Function

In *FMS* it is of interest to select models that minimize classification errors on unseen data (i.e., maximizing generalization performance). Therefore, the fitness function (F) should relate a model with an estimate of its classification performance in unseen data. The simplicity of *PSMS* allows us to use any classification performance measure as F, because the method does not require derivatives. Thus, valid options for F include mean absolute error, balanced error rate, squared root error, recall, precision, area under the *ROC* curve, etcetera. For this work it was used the balanced error rate (*BER*) as F. *BER* takes into account misclassification rates in both classes, which prevents *PSMS* of selecting biased models (favoring the majority class) in imbalanced data sets. Furthermore, *BER* has been used in machine learning challenges as leading error measure for ranking participants (Guyon et al., 2007, 2008). The *BER* of model ψ is the average of the misclassifications obtained by ψ over the classes in a data set, as described in Equation (4):

$$BER(\psi) = \frac{E_+ + E_-}{2} \tag{4}$$

where E_+ and E_- are the misclassifications rates for the positive and negative classes, respectively.

The selection of the fitness function is a crucial aspect in *PSO*. However, for *PSMS*, the critical part lies in the way an estimate of generalization performance of a model (given F and training data) is obtained, and not in the fitness function itself. This is the main challenge of model selection, since error estimates using training data are very optimistic about the behavior of models on unseen data, this phenomenon is known as overfitting (Bishop, 2006; Hastie et al., 2001; Nelles, 2001). In order to overcome overfitting the *BER* was calculated using a $k-$fold cross validation *(k-fold CV)* approach (Note that the *BER* is still obtained from training data). This is the only explicit mechanism of *PSMS* to avoid overfitting. *k-fold CV* is the most used hold-out approach for model selection (Nelles, 2001; Hastie et al., 2001; Cawley and Talbot, 2007b). Using a high value for k, say $k = N$ where N is the training set size (i.e., *leave one out - CV*), the error estimate is almost unbiased, but can have high variance because the N training sets are very similar to each other (Nelles, 2001; Cawley and Talbot, 2007b); furthermore, computation time could be a serious problem (Hastie et al., 2001; Nelles, 2001). With a low value for k, the estimate can have low variance but the bias could be a problem. The selection of an optimal k is still an open problem in the model selection field. For this work were performed experiments with $k \in \{2,5,10\}$, but no statistically-significant difference, among these values, was found, see Section 4.2.

3.4 Computational Complexity

As we have seen, the search space in the *FMS* problem is composed by all the possible models that can be built given the considered methods and their hyperparameters. This is an infinite search space even with the restriction imposed to the values that models can take; this is the main drawback of *FMS*. However, the use of particle swarm optimization (PSO) allows us to harness the complexity of this problem. Most algorithms used for *FMS* cannot handle very big search spaces. But PSO is well suited to large search spaces: it converges fast and has a manageable computational complexity (Kennedy and Eberhart, 2001; Reyes and Coello, 2006). As we can see from Algorithm 1, *PSO* is very simple and does not involve expensive operations.

The computational expensiveness of *PSMS* is due to the fitness function we used. For each selected model the fitness function should train and evaluate such model $k-$times. Depending on the model complexity this process can be performed on linear, quadratic or higher order times. Clearly, computing the fitness function using the entire training set, as opposed to *k-fold CV*, could reduce *PSMS* complexity, although we could easily overfit the data. For a single run of *PSMS* the fitness function should be evaluated $\rho = m \times (I+1)$ times, with m being the swarm size and I the number of iterations. Suppose the computational complexity of model λ is bounded by λ_O then the computational complexity of *PSMS* will be bounded by $\rho \times k \times \lambda_O$. Because λ_O is related to the computational complexity of model λ (which depends on the size and dimensionality of the data set) this value may vary dramatically. For instance, computing the fitness function for a naïve Bayes model in a high dimensional data set takes around two seconds[4], whereas computing the same fitness function for the same data set could take several minutes if a support vector classifier is used.

In order to reduce the computational cost of *PSMS* we could introduce a complexity penalty-term into the fitness function (this is current work). A simpler alternative solution is calculating the fitness function for each model using only a small subset of the available data; randomly selected and different each time. This approach can also be useful for avoiding local minima. The subsampling heuristic was used for high-dimensional data sets and for data sets with a large number of examples. Experimental results show an important reduction of processing time, without a significant loss of accuracy, see Section 4.3. We emphasize that complexity is due to the nature of the *FMS* problem. With this approach, however, users will be able to obtain models for their data without requiring knowledge on the data or on machine learning techniques.

4. Experimental Results

In this section results of experiments with *PSMS* using benchmark data from two different sources are presented. First, we present results on a suite of benchmark machine learning data sets[5] used by several authors (Mika et al., 2000; Rätsch et al., 2001; Cawley and Talbot, 2007b); such data sets are described in Table 3, ten replications (i.e., random splits of training and testing data) of each data set were considered. These data sets were used to compare *PSO* to *PS* in the *FMS* problem (Section 4.1) and to study the performance of *PSMS* under different settings (Section 4.2). Next we applied *PSMS* to the data sets used in a model selection competition (Section 4.3). The goal of the latter experiments is to compare the performance of *PSMS* against other model selection strategies.

4. Most of the experiments were carried out on a workstation with *Pentium*^{TM} 4 processor at 2.5 GHz, and 1 gigabyte in RAM.
5. These data sets are available from `http://ida.first.fraunhofer.de/projects/bench/benchmarks.htm`.

ID	Data set	Training Patterns	Testing Patterns	Input Features
1	Breast cancer	200	77	9
2	Diabetes	468	300	8
3	Flare solar	666	400	9
4	German	700	300	20
5	Heart	170	100	13
6	Image	1300	1010	20
7	Splice	1000	2175	60
8	Thyroid	140	75	5
9	Titanic	150	2051	3

Table 3: Data sets used in the comparison of *PSO* and *PS*, ten replications (i.e., random splits of training and testing sets) for each data set were considered.

4.1 A Comparison of *PSO* and *PS*

In the first set of experiments we compared the performance of *PSO* to that of another search strategy in the *FMS* problem. The goal was to evaluate the advantages of using the swarm strategy over another intensive search method. Among the available search techniques we selected *PS* because of its simplicity and proved performance in model selection (Dennis and Torczon, 1994; Momma and Bennett, 2002; Bi et al., 2003). *PS* is a direct search method that samples points in the search space in a fixed pattern about the best solution found so far (the center of the pattern). Fitness values are calculated for the sampled points trying to find a minimizer; if a new minimum is find then the center of the pattern is changed, otherwise the search step is reduced by half; this process is iterated until a stop criteria is met. The considered *PS* algorithm described in Algorithm 2 is an adaptation of that proposed by Momma et al. for hyperparameter optimization of support vector regression (Momma and Bennett, 2002; Bi et al., 2003).

The input to *PS* is the pattern P and the search step Δ. Intuitively, P specifies the direction of the neighboring solutions that will be explored; while Δ specifies the distance to such neighboring solutions. There are several ways to generate P; for this work we used the nearest neighbor sampling pattern (Momma and Bennett, 2002). Such pattern is given numerically by $P = [\mathbf{I}_d \ -\mathbf{I}_d \ \mathbf{0}^T_{1\times d}]$, where $\mathbf{I}_d$ is the identity matrix of size d; $\mathbf{0}_{1\times d}$ is a vector of size $1_\times d$ with all zero entries; d is the dimensionality of the problem. Δ, a vector of size $1 \times d$, is the search step by which the space of solutions is explored. We defined $\Delta = \frac{\mathbf{q}_{maxvals} - \mathbf{q}_{minvals}}{2}$, where $\mathbf{q}_{maxvals}$ and $\mathbf{q}_{minvals}$ are the maximum and minimum values that the solutions can take, respectively. Each iteration of *PS* involves the evaluation of $N_c - 1$ solutions (where N_c is the number of columns of P). Solutions are updated by adding s_j ($j \in 1, \ldots, N_c - 1$) to the current-best solution $\mathbf{q}_g$; where each s_j is obtained by multiplying (element-by-element) the search step vector Δ and the j^{th} column of P. $\mathbf{q}_g$ is replaced by a new solution $\mathbf{q}_j$ only if $f_{\mathbf{q}_j} < f_{min} = f_{\mathbf{q}_g}$. The output of *PS* is $\mathbf{q}_g$, the solution with the lowest fitness value; we encourage the reader to follow the references for further details (Momma and Bennett, 2002; Dennis and Torczon, 1994).

For applying *PS* to the *FMS* problem (hereafter *PATSMS*) the solution $\mathbf{q}_g$ was initialized in the same way that each particle in the swarm was initialized for *PSMS* (i.e., randomly). The same

Algorithm 2 Pattern search. p_i is the i^{th} column of P, N_c the number of columns in P.

```
Require:
– $I_{ps}$: number of iterations
– $\mathbf{F}(\Psi \rightarrow \mathbb{R})$: fitness function
– $\mathbf{P}$: pattern
– $\Delta$: search step
Initialize solution $\mathbf{q}_i$ $(i = 1)$
Compute $f_{\mathbf{q}_i} = F(\mathbf{q}_i)$ (Section 3.3)
Set $\mathbf{q}_g = \mathbf{q}_i$; $f_{min} = f_{\mathbf{q}_i}$
while $i < I_{ps}$ do
  for all $p_j \in P_{1,\ldots,N_c-1}$ do
    $s_j = \Delta . p_j$
    $\mathbf{q}_j = \mathbf{q}_g + s_j$
    Compute $f_{\mathbf{q}_j} = F(\mathbf{q}_j)$
    if $f_{\mathbf{q}_j} < f_{min}$ then
      Update $\mathbf{q}_g$ ($\mathbf{q}_g = \mathbf{q}_j$, $f_{min} = f_{\mathbf{q}_j}$)
    end if
    $i^{++}$
  end for
  $\Delta = \Delta/2$
end while
return $\mathbf{q}_g$
```

representation, fitness function and restrictions on the values that each dimension can take were considered for both methods, see Section 3. Under these settings *PS* is a very competitive baseline for *PSO*.

In each experiment described below, we let *PS* and *PSO* perform the same number of fitness function evaluations, using exactly the same settings and data sets, guaranteeing a fair comparison. Since both methods use the same fitness function and perform the same number of fitness function evaluations, the difference in performance indicates the gain we have by guiding the search according to Equations (1) and (2). As recommended by Demsar, we used the Wilcoxon signed-rank test for the comparison of the resultant models (Demsar, 2006). In the following we will refer to this statistical test with 95% of confidence when mentioning statistical significance.

We compared the *FMS* ability of *PSMS* and *PATSMS* by evaluating the accuracy of the selected models at the end of the search process; that is, we compared the performance of models $\mathbf{p}_g$ and $\mathbf{q}_g$ in Algorithms 1 and 2, respectively. We also compared the performance of the solutions tried through the search. For each trial we fixed the number of iterations for *PSMS* to $I = 100$ with a swarm size of $m = 10$. In consequence, both search strategies performed $m \times (I + 1) = 1010$ evaluations of the fitness function in each run. Because more than $180,000$ models were tested we used $2-$fold *CV* for computing the fitness function. In each trial the training set was used for *FMS* and the resultant model was evaluated in the test set. This process was repeated for each replica of each data set described in Table 3. Averaged results of this experiment are shown in Table 4. We show the average *CV* error obtained through the search process (*CV-BER*) and the error obtained by the selected model, at the end of the search, in the test set (*test-BER*).

ID	**Data set**	**PATSMS test-BER**	**PSMS test-BER**	**PATSMS CV-BER**	**PSMS CV-BER**
1	Breast-cancer	36.98$\pm$0.08	**33.59$\pm$0.12**	**32.64$\pm$0.06**	32.96$\pm$0.01
2	Diabetes	26.07$\pm$0.03	**25.37$\pm$0.02**	**25.39$\pm$0.02**	26.48$\pm$0.05
3	Flare-solar	32.87$\pm$0.02	**32.65$\pm$0.01**	**32.69$\pm$0.01**	33.13$\pm$0.01
4	German	28.65$\pm$0.02	**28.28$\pm$0.02**	**31.00$\pm$0.00**	31.02$\pm$0.00
5	Heart	19.50$\pm$0.19	**17.35$\pm$0.06**	**16.96$\pm$0.07**	19.93$\pm$0.03
6	Image	3.58$\pm$0.01	**2.50$\pm$0.01**	**11.54$\pm$0.10**	15.88$\pm$0.04
7	Splice	13.94$\pm$0.99	**9.46$\pm$0.25**	**18.01$\pm$0.05**	19.15$\pm$0.07
8	Thyroid	10.84$\pm$0.39	**5.98$\pm$0.06**	**11.15$\pm$0.20**	15.49$\pm$0.12
9	Titanic	29.94$\pm$0.00	**29.60$\pm$0.00**	**27.19$\pm$0.13**	27.32$\pm$0.13

Table 4: Average and variance of *test-BER* and *CV-BER* obtained by models selected with *PSMS* and *PATSMS*, the best results are shown in **bold**. *test-BER* is the *BER* obtained by the selected model using the test set (averaged over 10 replications of each data set). *CV-BER* is the average of *CV BER* obtained by each of the candidate solutions through the search process (averaging performed over all particles and iterations and over 10 replications of each data set) .

The performance of both search strategies is similar. *PSMS* outperformed *PATSMS* through all of the data sets at the end of the search process (Columns 3 and 4 in Table 4). The *test-BER* differences are statistically significant for all but the *flare-solar* and *german* data sets. In the latter data sets the hypothesis that models selected with *PATSMS* and *PSMS* perform equally cannot be rejected. However, note that for these data sets the models selected with *PSMS* outperformed those selected with *PATSMS* in 6 out of 10 replications. Globally, we noticed that from the 90 trials ($9-$data sets, $\times$ $10-$replications for each, see Table 3), 68.9% of the models selected with *PSMS* outperformed those obtained with *PATSMS*. While only in 22.2% of the runs *PATSMS* outperformed *PSMS*, in the rest both methods tied in performance. A statistical test over these 90 results was performed and a statistically-significant difference, favoring *PSMS*, was found. Despite *PATSMS* being a strong baseline, these results give evidence that *PSMS* outperforms *PATSMS* at the end of the search process.

Columns five and six in Table 4 show the average *BER* obtained with each strategy through the 1010 evaluations for each data set (averaged over replications). *CV-BER* reflects the behavior of the search strategies through the search process. *PATSMS* slightly outperformed *PSMS* in this aspect, though the difference is statistically significant only for the *Heart*, *image* and *thyroid* data sets. The slight superior performance of *PATSMS* in *CV-BER* is due to the pattern we used and the search procedure itself. *PATSMS* performs a finer grained search over a restricted search area around the initial solution $\mathbf{q}_g$. The latter results in a lower *CV-BER* because *PATSMS* always moves the pattern towards the local minimum nearest to the initial solution, $\mathbf{q}_g$. *PSMS,* on the other hand, explores a much larger search space because the search is not only guided by the best solution found so far ($\mathbf{p}_g$), but also by the individual best solutions of each particle ($\mathbf{p}_{1,\dots,m}$). The latter produces a higher *CV-BER* in average because, even when the *CV-BER* of the final models is low, many models of varied *CV-BER* performance are tried through the search.

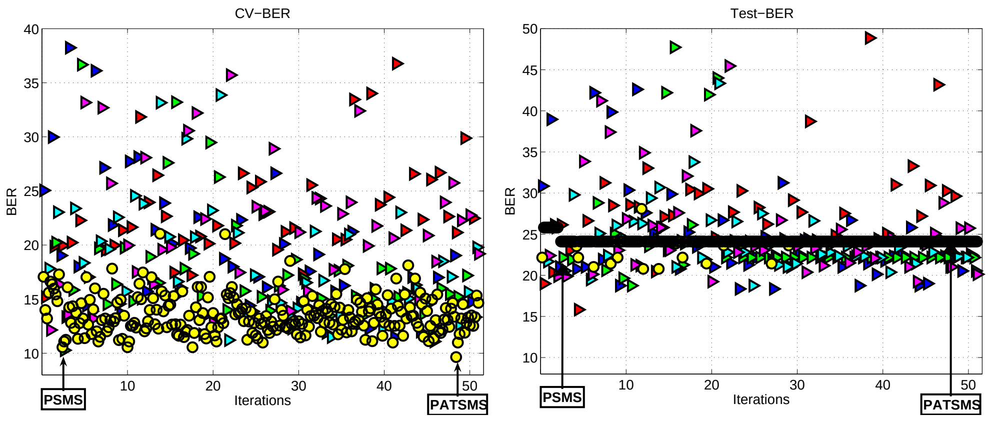

Figure 1: Performance of *PATSMS* (circles) and *PSMS* (triangles) as a function of number of iterations for an experiment with one replication of the *Heart* data set. For *PSMS* we show the performance of each particle with a different color. Left: Cross-validation Balanced Error Rate (*BER*). Right: Test set *BER*. In both plots we indicate with an arrow the model selected by each search strategy.

In Figure 1 we show the performance of the solutions tried through the search by each method for a single replication of the *Heart* data set (for clarity in the plots we used $m = 5$ and $I = 50$ for this experiment). We show the *CV* and test *BER* for every model tried through the search. It can be seen that the *CV-BER* of *PATSMS* is lower than that of *PSMS*, showing an apparent better convergence behavior (left plot in Figure 1). However, by looking the test *BER* of the models tried, it becomes evident that *PATSMS* is trapped into a local minimum since the very first iterations (right plot in Figure 1). *PSMS*, on the other hand, obtains a higher *CV-BER* through the search, though it is less prone to follow a local minimum. This is because with *PSMS* the search is guided by one global and m local solutions, which prevents *PSMS* from performing a pure local search; the latter in turn prevents *PSMS* of overfitting the data. This result gives evidence of the better convergence behavior of *PSMS*.

The model selected with *PSMS* obtained a lower *test-BER* than that selected with *PATSMS* (see right plot in Figure 1). In fact all of the solutions in the final swarm outperformed the solution obtained with *PATSMS* in *test-BER*. With *PSMS* the model of lowest test *BER* was obtained after 3 iterations of *PSMS*, giving evidence of the fast convergence of *PSO*. One should note that the *test-BER* of the worst *PSMS* solutions is higher than that of the best *PATSMS* solution. However, near the end of the search, the possibilities that *PSMS* can select a worse solution than *PATSMS* are small.

We have seen that *PATSMS* is prone to get trapped into a local minimum because it performs a fine grained search over a small area. This causes that only a few methods are considered through the search with this method. *PSMS*, on the other hand, tries a wide variety of classifiers and feature selection methods, exploring a larger search space and being able to escape from local minima. In order to analyze the diversity of the methods considered by *PSMS*, in Figure 2 we show the

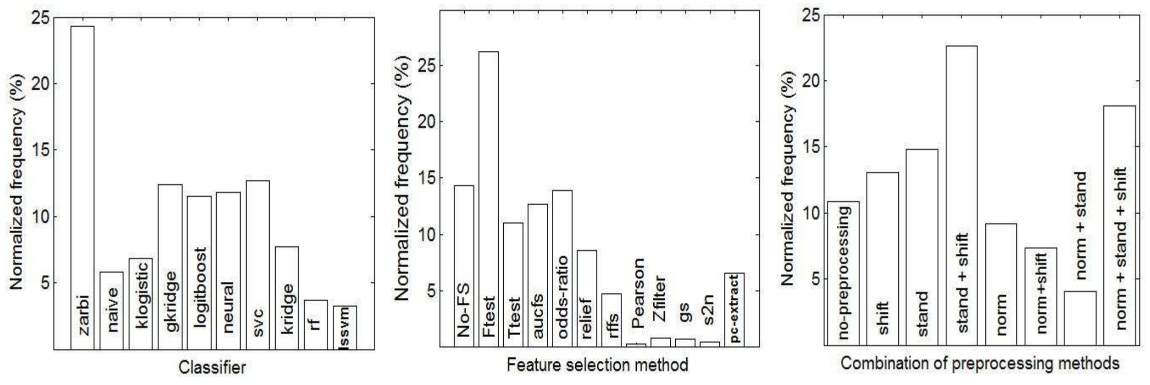

Figure 2: Normalized frequency of classifiers (left), feature selection method (middle) and combination of preprocessing methods (right) preferred by *PSMS* through the search process. In the right plot, the abbreviations *shift*, *stand* and *norm* stand for *shift-scale*, *standardization* and *normalization*, respectively. Results are normalized over the 90,900 models tried for obtaining the results from Table 4.

normalized frequency of methods preferred by *PSMS*[6] through the search. The results shown in this figure are normalized over the 90,900 models tried for obtaining the results from Table 4. From this figure, we can see that most of the classifiers and feature selection methods were considered for creating solutions. No classifier, feature selection method or combination of preprocessing methods was used for more than 27% of the models. This reflects the fact that different methods are required for different data sets and that some are equivalent. There were, however, some methods that were slightly more used than others.

The preferred classifier was the *zarbi CLOP*-object, that was used for about 23% of the models. This is a surprising result because *zarbi* is a very simple linear classifier that separates the classes by using information from the mean and variance of training examples (Golub et al., 1999). However, in 97.35% of the times that *zarbi* was used it was combined with a feature selection method. *Gkridge, svc, neural* and *logitboost* were all equally selected after *zarbi*. *Ftest* was the most used feature selection method, though most of the feature selection strategies were used. Note that *Pearson, Zfilter, gs* and *s2n* were considered only for a small number of models. The combination *standardize + shift-scale* was mostly used for preprocessing, although the combination *normalize + standardize + shift-scale* was also highly used. Interestingly, in 70.1% of the time preprocessing was performed before feature selection. These plots illustrate the diversity of classifiers considered by *PSMS* through the search process, showing that *PSMS* is not biased towards models that tend to perform very well individually (e.g., *logitboost, rf* or *gkridge*). Instead, *PSMS* attempts to find the best full model for each data set.

6. We do not show those preferred by *PATSMS* because the methods selected with this strategy are, most of the times, those considered in the initial solution. In our experiments we found that for each replication *PATSMS* used the same classifier and feature selection method for about 95% of the search iterations. The selection of these methods depended on the initial solution instead of the way the search space is explored or the individual performance of the methods. Therefore, no useful information can be obtained about why some methods are preferred over others, even when in average (over the 90,900 solutions tried) the histograms may look similar to those shown in Figure 2.

k	CV-BER	test-BER	test-BER*	Time
2	25.36$^+_-$0.56	22.99$^+_-$1.18	25.79$^+_-$0.59	4166.85
5	25.18$^+_-$0.59	20.50$^+_-$1.53	26.30$^+_-$0.46	6062.56
10	24.54$^+_-$0.63	20.82$^+_-$1.74	26.03$^+_-$0.51	7737.11

Table 5: Average *CV-BER* (average CV result over all particles and iterations), *test-BER* (test result corresponding to the best CV result), *test-BER** (average test result over all particles and iterations) and processing time (in seconds) for different values of k in the $k-$fold *CV*. Results are averaged over a single replication of each data set described in Table 3.

4.2 Parameter Selection for *PSMS*

In this section we analyze the performance of *PSMS* under different settings. The goal is to identify mechanisms in *PSMS* that allow us obtaining competitive models and, to some extent, avoiding overfitting. For the experiments described in this section we consider a single replication for each data set. As before, we show the average *CV* error of the solutions considered during the search as well as the error of the selected model in the test set. We also show the average test set error of all solutions tried through the search *test-BER** (averaging performed over all particles and all iterations), providing information about the generalization performance of the models considered throughout the search.

4.2.1 VALUE OF K IN K-FOLD CV

First we analyze the behavior of *PSMS* for different values of k in the *CV* for computing the fitness function. We consider the values $k = [2,5,10]$. Average results of this experiment are shown in Table 5.

From this table, we can see that the performance is similar for the different values of k considered. The best results at the end of the search are obtained with $k = 5$ (column 3 in Table 5), while the best generalization performance is obtained with $k = 2$ (column 4 in Table 5). However, these differences are not statistically significant. Therefore, the null hypothesis that models selected with $k = [2,5,10]$ perform equally in *test-BER* and *test-BER** cannot be rejected. The latter is an important result because by using $k = 2$ the processing time of *PSMS* is considerably reduced. Note that for models of quadratic (or higher order) complexity, computing the fitness function with $2-$fold *CV* is even more efficient than computing the fitness function using the full training set. It is not surprising that processing time increases as we increase k (column 5 in Table 5). Although it is worth mentioning that the variance in processing time was very large (e.g., for $k = 2$ it took 7 minutes applying *PSMS* to the *titanic* data set and about five hours for the *image* data set).

4.2.2 NUMBER I OF ITERATIONS

Next we performed experiments varying the number of iterations (I), using $2-fold\ CV$ for computing the fitness function and a swarm size of $m = 10$. We considered the values $I = [10,25,50,100]$. Averaged results for this experiment are shown in Table 6 (rows 2 - 5). As we can see the best results were obtained by running *PSMS* for 50 iterations. Interestingly, models selected by running *PSMS* for 10 iterations outperformed those selected after 100 iterations in *test-BER*, though the difference

Setting	CV-BER	test-BER	test-BER*
I=10	25.33$\pm$0.33	22.17$\pm$1.81	27.64$\pm$0.52
I=25	25.29$\pm$0.33	21.88$\pm$1.68	27.59$\pm$0.49
I=50	**24.02$\pm$0.38**	**21.12$\pm$1.74**	**26.72$\pm$0.65**
I=100	24.57$\pm$0.37	22.81$\pm$1.44	27.27$\pm$0.48
m=5	**24.27$\pm$0.49**	**20.81$\pm$1.50**	**25.01$\pm$0.85**
m=10	25.07$\pm$0.34	21.64$\pm$2.04	25.99$\pm$0.74
m=20	25.09$\pm$0.34	21.76$\pm$1.84	26.00$\pm$0.64
m=40	24.82$\pm$0.43	21.45$\pm$2.13	25.96$\pm$0.78
m=50	25.32$\pm$0.44	22.54$\pm$1.65	26.11$\pm$0.77
W=(0,0,0)	**23.86$\pm$0.71**	20.40$\pm$1.71	**22.46$\pm$1.32**
W=(1.2,0.5,0.5)	24.22$\pm$0.76	**19.41$\pm$1.37**	23.38$\pm$1.34
W=(1,1,1)	27.62$\pm$0.30	21.88$\pm$1.68	27.13$\pm$0.53

Table 6: Average *CV-BER*, *test-BER* and *test-BER** for different settings of *m*, *I* and **W**. The best results are shown in **bold**. Results are averaged over a single replication of each data set described in Table 3.

was not statistically significant. The difference in performance between models selected after 50 and 100 iterations was statistically significant. This result shows the fast convergence property of *PSMS* and that early stopping could be an useful mechanism to avoid overfitting, see Section 5.

4.2.3 SWARM SIZE M

In the next experiment we fixed the number of iterations to $I = 50$ and varied the swarm size as follows, $m = [5,10,20,40,50]$. Results of this experiment are shown in rows 6-10 in Table 6. This time the best result was obtained by using a swarm size of 5, however there is a statistically-significant difference only between $m = 5$ and $m = 50$. Therefore, models of comparable performance can be obtained by using $m = [5,10,20,40]$. This is another interesting result because using a small swarm size reduces the number of fitness function evaluations for *PSMS* and therefore makes it more practical. An interesting result is that by using 50 iterations with any swarm size the *test-BER* is very close to the *CV-BER* estimate. Again, this provides evidence that early stopping can improve the average generalization performance of the models.

4.2.4 INERTIA WEIGHT W

We also performed experiments with different configurations for **W**, the adaptive inertia weight. Each configuration is defined by a triplet $\mathbf{W} = (w_{start}, w_f, w_{end})$, whose elements indicate the starting value for W, the proportion of iterations to vary it and its final value, respectively, see Section 2. Three configurations were tried with the goal of evaluating the advantages of using an adaptive inertia weight instead of constant values. Results of this experiment are shown in rows 11-13 in Table 6. It can be seen that the best results in *CV-BER* and *Test-BER** were obtained with $\mathbf{W} = (0,0,0)$; the differences with the other results are statistically-significant. Under this configuration *PSMS* is not taking into account information from past velocities for updating solutions; which causes *PSMS* to converge quickly into a local minimum and refining the search around this point.

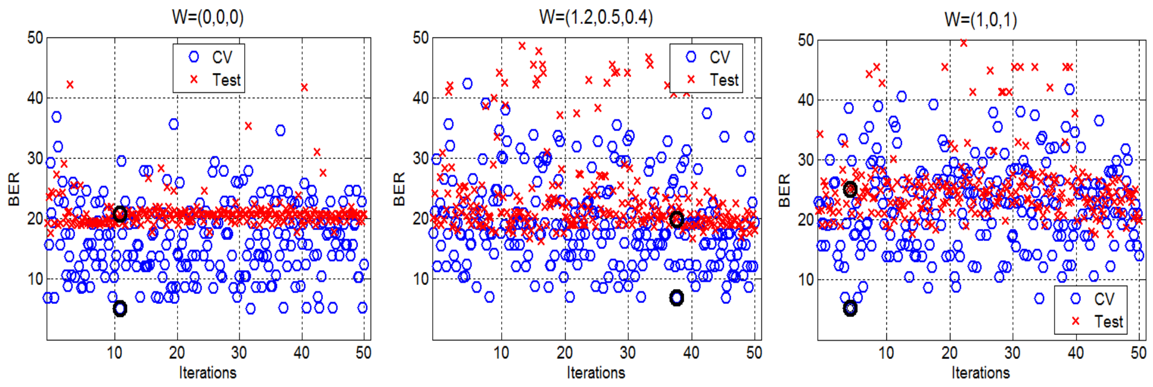

Figure 3: Algorithm performance as a function of number of iterations for different configurations of $\mathbf{W}$. *CV-BER* (circles) and *test-BER** (crosses) for the *Heart* data set. We are displaying the test and CV BER values for each particle at every time step. Since each time step involves m=5 particles, then for each iteration are displayed m=5 crosses and m=5 circles. We consider the following configurations: $\mathbf{W} = (0,0,0)$ (left), $\mathbf{W} = (1.2,0.5,0.4)$ (middle) and $\mathbf{W} = (1,0,1)$ (right). The *CV-BER* and *test-BER* of the best solution found by *PSMS* are enclosed within a bold circle.

The best result at the end of the search (column 3 in Table 6) was obtained with $\mathbf{W} = (1.2,0.5,0.4)$, the difference with the other results is statistically-significant. Under this configuration both global and local search is performed during the *PSMS* iterations; which caused higher *CV-BER* and *test-BER** than that of the first configuration, however, the generalization performance of the final model was better. The configuration $\mathbf{W} = (1,0,1)$ obtained the worst results in all of the measures; this is because under this configuration the search is never refined, since *PSMS* always takes into account the past velocity for updating solutions. The latter configuration could be a better choice for *FMS* because this way *PSMS* does not over-search around any solution; however, local search is also an important component of any search algorithm. In Figure 3 we show the *CV* and test *BER* of solutions tried during the search for the *Heart* data set under the different configurations tried. From this figure we can appreciate the fact that using constant values for W results in more local (when $\mathbf{W} = (0,0,0)$) or global search (when $\mathbf{W} = (1,0,1)$). An adaptive inertia weight, on the other hand, aims to control the tradeoff between global and local search, which results in a model with lower variance for this example. Therefore, an adaptive inertia weight seems to be a better option for *FMS*; this is because it prevents, to some extend, *PSMS* to overfit the data. However, further experiments need to be performed in order to select the best configuration for $\mathbf{W}$.

4.2.5 INDIVIDUAL (c_1) AND GLOBAL (c_2) WEIGHTS

We now analyze the performance of *PSMS* under different settings of the individual (c_1) and global (c_2) weights. We considered three configurations for c_1 and c_2; in the first one both weights have the same influence in the search (i.e., $c_1 = 2; c_2 = 2$), this was the setting used for all of the experiments reported in Section 4. In the second setting the local weight has no influence in the search (i.e., $c_1 = 0; c_2 = 2$), while in the third configuration the global weight is not considered in the search (i.e., $c_1 = 2; c_2 = 0$). We ran *PSMS* for $I = 50$ iterations with a swarm size of $m = 5$, using a single

ID	**Setting**	**CV-BER**	**test-BER**	**test-BER***
1	$c_1 = 2; c_2 = 2$	**23.69$^+_-$0.68**	**19.72$^+_-$1.45**	**23.92$^+_-$1.16**
2	$c_1 = 0; c_2 = 2$	26.87$^+_-$0.43	22.42$^+_-$1.32	27.13$^+_-$0.55
3	$c_1 = 2; c_2 = 0$	24.99$^+_-$0.41	21.73$^+_-$1.42	25.59$^+_-$0.66

Table 7: Average *CV-BER*, *test-BER* and *test-BER** for different settings of c_1 and c_2 in Equation (1). The best results are shown in **bold**. Results are averaged over a single replication of each data set described in Table 3.

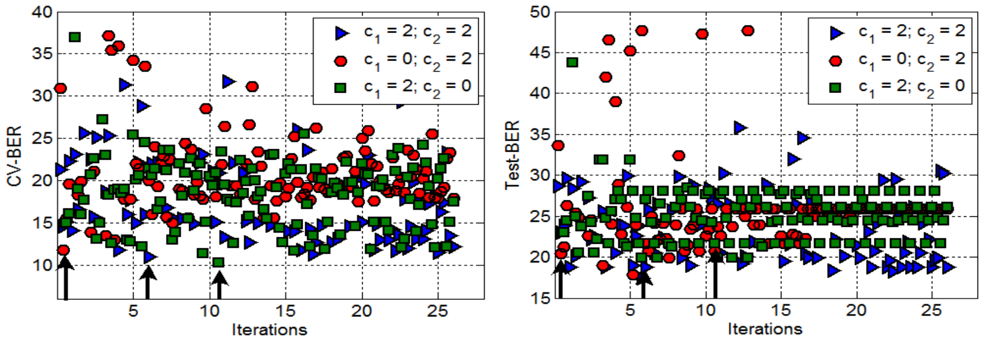

Figure 4: Performance of *PSMS* as a function of number of iterations using different settings for c_1 and c_2, see Table 7. We show the *CV-BER* (left) and *Test-BER* (right) for a single replication of the *Heart* data set. *PSMS* was ran for $I = 25$ iterations in this experiment. The models selected with each configuration are indicated with arrows.

replication for each data set described in Table 3; for the three configurations considered it was used the same adaptive inertia weight $\mathbf{W} = (1.2, 0.5, 0.4)$; averaged results of this experiment are shown in Table 7.

From this table we can see that the best performance is obtained by assigning equal weights to both factors. The difference in performance is statistically-significant over all measures with respect to the other two configurations. Therefore, by using the first configuration we can obtain solutions of better performance through and at the end of the search. More importantly, solutions of better generalization performance can be obtained with this configuration as well. The difference in performance is higher with the second configuration, where the individual-best solutions have no influence in the search; therefore, *PSMS* is searching locally around the global best solution. In the third configuration the global-best solution has no influence in the search; in consequence, the search is guided according the $m-$individual-best solutions. For illustration in Figures 4 and 5 we show the performance of *PSMS* as a function of the number of iterations for a single replication of the *Heart* data set. In Figure 4 we show the performance of *PSMS* for $I = 25$ iterations and in Figure 5 for $I = 100$ iterations.

From these figures we can see that the *CV* estimate is very similar for the different settings we considered (left plots in Figures 4 and 5). However, by looking at the performance of the solutions in the test set (right plots in Figures 4 and 5), we can appreciate that configurations 2 and 3 overfit

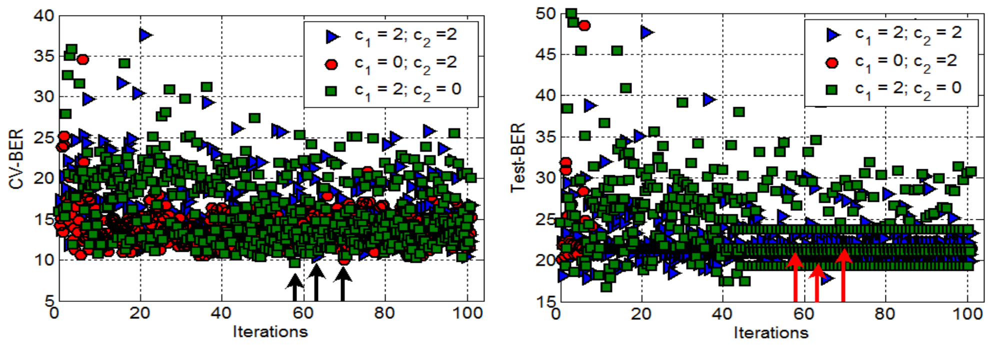

Figure 5: Performance of *PSMS* as a function of number of iterations using different settings for c_1 and c_2, see Table 7. We show the *CV-BER* (left) and *Test-BER* (right) for a single replication of the *Heart* data set. *PSMS* was ran for $I = 100$ iterations in this experiment. The models selected with each configuration are indicated with arrows.

the data (red circles and green squares). With $c_1 = 0$ we have that the $m = 5$ particles converge to single local minima, performing a fine grained search over this solution (red circles). With $c_2 = 0$ each of the $m-$particles converge to different local minima, overdoing the search over each of the m solutions (green squares). On the other hand, with the configuration $c_1 = 2$, $c_2 = 2$ *PSMS* is not trapped into a local minimum (blue triangles); searching around promising solutions, but without doing a fine grained search over any of them.

Better models (indicated by arrows) are selected by *PSMS* with the first configuration, even when their *CV* is higher than that of the models selected with the other configurations. This result confirms that *PSMS* is overfitting the data with the configurations 2 and 3. Note that with $I = 25$ iterations (Figure 4) the first configuration is not converging to a local minimum yet; while with $I = 100$ iterations (Figure 5) it looks like *PSMS* starts searching locally at the last iterations. This result illustrates why early stopping can be useful for obtaining better models with *PSMS*.

In order to better appreciate the generalization performance for the different configurations, in Figure 6 we plot the *CV-BER* as a function of *test-BER* for the run of *PSMS* with $I = 25$, we plot each particle with a different color.

From this figure we can see that the best model is obtained with the first configuration; for the configurations 2 and 3 the particles obtain the same *test-BER* for different solutions (middle and right plots in Figure 6). Despite the *CV* estimate is being minimized for these configurations, the *test-BER* performance of models does not improve. It is clear from the right plot in Figure 6 that with $c_2 = 0$ each particle is trapped in different local minima, doing a fine grained search over them that causes *PSMS* to overfit the data. It also can be seen from the middle plot that with $c_1 = 0$ the search is biased towards a single global-best solution (magenta circle), again, causing *PSMS* to overfit the data. On the other hand, results with the first configuration (left plot in Figure 6) show that particles do not oversearch at any solution.

4.3 Results on the Model Selection Challenge

In this section we describe experimental results of *PSMS* in the framework of a model selection competition called *agnostic learning vs. prior knowledge challenge* (*ALvsPK*) (Guyon et al., 2007,

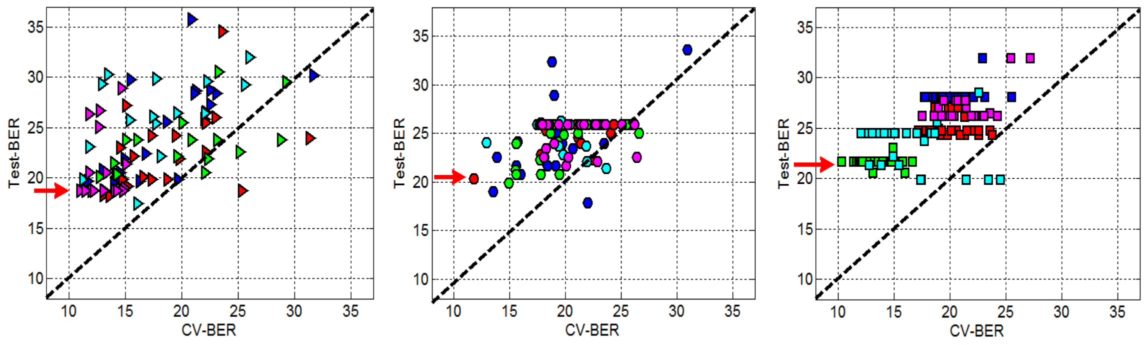

Figure 6: *Test-BER* as a function of *CV-BER* for a run of *PSMS* for $I = 25$ iterations in the *Heart* data set. Results with different configurations for c_1 and c_2 are shown. Left: $c_1 = 2$, $c_2 = 2$. Middle: $c_1 = 0$, $c_2 = 2$. Right: $c_1 = 2$, $c_2 = 0$. In each plot each particle is shown with a different color. The selected model with each configuration is indicated with an arrow.

2008). The goal of these experiments is to compare the performance of *PSMS* against other model selection strategies that work for a single algorithm or that use domain knowledge for this task. Through its different stages, the *ALvsPK* competition evaluated novel strategies for model selection as well as the added value of using prior knowledge for improving classification accuracy (Guyon et al., 2008). This sort of competitions are very useful because through them the real effectiveness of methods can be evaluated; motivating further research in the field and collaborations among participants.

4.3.1 CHALLENGE PROTOCOL AND CLOP

The rules of the challenge were quite simple, the organizers provided five data sets for binary classification together with the *CLOP* toolbox (Saffari and Guyon, 2006). The task was to obtain the model with the lowest *BER* over the five data sets on unseen data. Participants were free to elect using *CLOP* or their own learning machine implementations. The challenge is over now, although the challenge website[7] still remains open, allowing the evaluation of learning techniques and model selection methods. A complete description of the challenge and a comprehensive analysis of the results are described by Guyon et al. (2006c, 2007, 2008).

The competition was divided into two stages. The first stage, called *the model selection game* (Guyon et al., 2006c), was focused on the evaluation of pure model selection strategies. In the second stage, the goal was to evaluate the gain we can have by introducing prior knowledge into the model selection process (Guyon et al., 2007, 2008). In the latter stage participants could introduce knowledge of the data domain into the model selection process (*prior knowledge track*). Also, participants could use agnostic methods in which no domain knowledge is considered in the selection process (*agnostic track*).

The data sets used in the agnostic track of the *ALvsPK* challenge are described in Table 8, these data sets come from real domains. Data sets used for the agnostic and prior knowledge tracks were

7. See `http://www.agnostic.inf.ethz.ch/`.

different. For the agnostic track the data were preprocessed and dummy features were introduced, while for the prior knowledge track raw data were used, together with a description of the domain. We should emphasize that, although all of the approaches evaluated in the *ALvsPK* competition faced the same problem (that of choosing a model that obtains the lowest classification error for the data), such methods did not adopt the *FMS* interpretation. Most of the proposed approaches focused on a fixed machine learning technique like tree-based classifiers (Lutz, 2006), or kernel-based methods (Cawley, 2006; Pranckeviciene et al., 2007; Guyon et al., 2008), and did not take into account feature selection methods. Participants in the prior knowledge track could introduce domain knowledge. Furthermore, most of participants used their own implementations, instead of the *CLOP* toolbox.

After the challenge, the CLOP toolkit was augmented with methods, which performed well in the challenge (Cawley, 2006; Cawley and Talbot, 2007a; Lutz, 2006). These include Logitboost (Friedman et al., 2000), LSSVM (Suykens and Vandewalle, 1999), and kernel ridge regression (Saunders et al., 1998; Hastie et al., 2001).

4.3.2 COMPETITIVENESS OF PSMS

In both stages of the competition we evaluated models obtained with *PSMS* under different settings. Models obtained by *PSMS* were ranked high in the participants list, showing the competitiveness of *PSMS* for model selection (Guyon et al., 2006c, 2007, 2008; Escalante et al., 2007). Furthermore, the difference with methods that used prior knowledge was relatively small, showing that *FMS* can be a viable solution for the model selection problem without the need of investing time in introducing domain knowledge, and by considering a wide variety of methods.

The results of *PSMS* in the *ALvsPK* challenge have been partially analyzed and discussed elsewhere (Escalante et al., 2007; Guyon et al., 2007, 2008). During the challenge, our best entry (called *Corrida-final*) was ranked[8] 8^{th} over all ranked participants, 5^{th} among the methods that did not use domain knowledge and 2^{nd} among the methods that used the software provided by the organizers (Guyon et al., 2006c, 2007, 2008). For *Corrida-final* we used $k = 5$ and the full training set for computing the fitness function; we ran *PSMS* for 500 iterations for the *Ada* data set and 100 iterations for *Hiva*, *Gina* and *Sylva*. We did not applied *PSMS* to the *Nova* data set in that entry, instead we selected a model for *Nova* by trial and error. For such entry we used a version of *CLOP* where only there were available the following classifiers *zarbi, naive, neural* and *svc* (Escalante et al., 2007); also, only four feature selection methods were considered.

4.3.3 POST-CHALLENGE EXPERIMENTS

In the rest of this section we present results of *PSMS* using the augmented toolkit, including all methods described in Tables 1 and 2. In these tables we consider implementations of *logitboost, lssvm and gkridge,* which are the classifiers that won the *ALvsPK* challenge (Cawley, 2006; Cawley and Talbot, 2007a; Lutz, 2006) and were added to CLOP after the end of the challenge.

In order to efficiently apply *PSMS* to the challenge data sets we adopted a subsample strategy in which, instead of using the full training set, small subsamples of the training data were used to compute the fitness function. Each time the fitness function is computed we obtain a different random sample of size $S_{sub} = \frac{N}{SF}$, where N is the number of instances and SF is a constant that specifies the proportion of samples to be used. Subsamples are only used for the search process. At

8. See `http://www.clopinet.com/isabelle/Projects/agnostic/Results.html`.

Data set	Domain	Type	Features	Training	Validation	Testing
Ada	Marketing	Dense	48	4174	415	41471
Gina	Digits	Dense	970	3153	315	31532
Hiva	Drug discovery	Dense	1617	3845	384	38449
Nova	Text classification	Sparse binary	16969	1754	175	17537
Sylva	Ecology	Dense	216	13086	1309	130857

Table 8: Benchmark data sets used for the model selection challenges (Guyon et al., 2006c, 2007, 2008).

Data	SF	Model	Time (m)	Test-BER
Ada	1	chain({logitboost(units=469,shrinkage=0.4,depth=1),bias}	368.12	16.86
Gina	2	chain({sns(1),relief(fmax=487),gkridge,bias}	482.23	2.41
Hiva	3	chain({norm(1),rffs(fmax=1001),lssvm(gamma=0.096),bias}	124.54	28.01
Nova	1	chain({rffs(fmax=338),norm(1),std(1),sns(1),gkridge,bias}	82.12	5.27
Sylva	10	chain({sns(1),odds-ratio(fmax=60),gkridge,bias}	787.58	0.62

Table 9: Models selected with *PSMS* for the data sets of the *ALvsPK* challenge. For each data set we show the subsampling factor used (**SF**), the selected model (**Model**, some hyperparameters are omitted for clarity), the processing **Time** in minutes and the **test-BER** obtained. *sns* is for *shit-scale*, *std* is for *standardize* and *norm* is for *normalize*. See Tables 1 and 2 for a description of methods and their hyperparameters.

the end of the search the selected model is trained using the full training set (for the experiments reported in this paper we considered as training set the union of the training and validation data sets, see Table 8). Due to the dimensionality of the *Nova* data set we applied principal component analysis to this data set. Then we used the first 400 components for applying *PSMS*. We fixed $k = 2$, $I = 50$ and $m = 5$ for our experiments based on the results from previous sections. Then we ran *PSMS* for all of the data sets under the above described settings using different values for *SF*. The predictions of the resultant models were uploaded to the challenge website in order to evaluate them. Our best ranked entry in the *ALvsPK* challenge website (called *psmsx_jmlr_run_I*) is described in Table 9, and a comparison of it with the currently best-ranked entries is shown in Table 10.

We can see from Table 9 that very different models were selected by *PSMS* for each data set. This is the main advantage the *FMS* because it allows us selecting an ad-hoc model for each data set by considering different model types and a wide diversity of methods. With exception of *Ada*, the selected models included a feature selection method; this result shows the importance of feature selection methods and that some of them are more compatible than others with some classifiers; note that different numbers of features were selected for each data set. In all but the *Nova* data set preprocessing was performed before feature selection. This can be due to the fact that for *Nova* we used principal components instead of the original features. For *Ada* it was selected a *logitboost* classifier, while for *Hiva* it was selected a *lssvm* classifier with gaussian kernel. For *Gina*, *Nova* and *Sylva* it was selected the *gkridge* classifier; this classifier performs virtual leave-one out model selection each time it is trained (Cawley et al., 2007). Note that both *gkridge* and *logitboost* were the classifiers that best performed during the challenge (Guyon et al., 2007, 2008); this result gives evidence that *PSMS* can obtain similar and even better models, without spending time on ad-hoc modifications for each data set and without using domain knowledge.

The use of the subsampling strategy allowed to efficiently apply *PSMS* to all of the data sets. About six hours were required to obtain a competitive model for *Ada*, while about only two for the *Hiva* data set. Note that applying *PSMS* for *Hiva* using the entire training set (*Corrida-final*) took about 80 hours in the same machine (Escalante et al., 2007). During our experiments we found that the larger the subsamples the better the performance of the selected model. However, the selection of *SF* also depends on the available computer resources and time restrictions. One should note that we can increase speed of *PSMS* by caching intermediate results that can be reused (e.g., preprocessing the data sets off-line).

Despite the use of the subsampling technique the models selected with *PSMS* resulted very competitive as shown in Table 10. Currently, the *PSMS* run is the top-ranked agnostic entry in the challenge website; furthermore, the performance of this entry is superior than all but one prior-knowledge entry: *Interim-all-prior*; which is the best ranked entry overall, using “prior knowledge” or “domain knowledge”. Note that the latter entry is formed of models that were designed ad-hoc for each data set, requiring much more time, effort and knowledge than *PSMS*. In average *PSMS* outperforms the best results presented in the *ALvsPK* challenge: *IJCNN07AL*, row 4 in Table 10 (Guyon et al., 2007); and the best-ranked agnostic entry (after *PSMS*): *Logitboost-with-trees*, row 5 in Table 10 (Lutz, 2006).

The performance of *PSMS* is very close to that of *IJCNN07AL*, tieing in *Sylva* and outperforming one to the other in two data sets. *PSMS* outperforms *Logitboost-with-trees* in three out of the five data sets and achieves very close performance for the other two data sets. The latter entry is ranked 10th in the challenge website. It is very interesting that for the *Ada* data set the best model so far is a *logitboost* classifier with about 1000 trees (Lutz, 2006); while with *PSMS* we were able to achieve almost the same performance by using a half that number of trees, see row 2 in Table 9. This is important because simpler models of similar performance can be obtained with *PSMS*.

PSMS clearly outperformed our best-ranked entry during the challenge (row 6 in Table 10); this result gives evidence that we obtained better results by using more and better classifiers; also, the use of subsamples instead of the entire training set (when computing the fitness function), does not damage the performance of *PSMS*, although the reduction in processing time is very important. Note that for *Nova* the *PSMS* entry obtained a slightly worse result than that of *Corrida-final*; however, the model for *Nova* in *Corrida-final* was selected by trial and error which required of much more effort and time.

Results reported in this section show the efficacy of *PSMS* for model selection. Despite its simplicity it has shown comparable and even superior performance to those obtained by other model selection strategies that focused on a single learning algorithm and to methods that used prior domain knowledge for guiding the model selection process (Lutz, 2006; Reunanen, 2007; Boullé, 2007; Pranckeviciene et al., 2007; Wichard, 2007). Models selected with *PSMS* are simple and yet very competitive; furthermore, with *PSMS* no knowledge is needed on the methods to choose from, nor on the domain. In consequence, it is very easy to obtain classifiers that can achieve state-of-the-art performance without spending time on designing, developing and optimizing an ad-hoc model. Even though *PSMS* can be applied to any binary classification problem, we are not claiming it will obtain satisfactory results in every domain; however, it can be considered as a first option when dealing with binary classification tasks. It is expected that this will further improve the performance of models selected with *PSMS* if domain knowledge is used.

Entry	Description	Ada	Gina	Hiva	Nova	Sylva	Overall	Rank
Interim-all-prior	Best-PK	17.0	2.33	27.1	4.71	0.59	10.35	1^{th}
psmsx_jmlr_run_I	PSMS	16.86	**2.41**	**28.01**	5.27	**0.62**	10.63	2^{nd}
IJCNN07AL	Best-AL	**16.60**	3.39	28.27	**4.56**	**0.62**	10.68	4^{th}
Logitboost-with-trees	Best-AL	**16.60**	3.53	30.18	4.69	0.78	11.15	10^{th}
Corrida-final	Best-PSMS-ALvsPK	18.27	6.14	28.54	5.11	1.22	11.86	42^{th}

Table 10: Comparison of models selected with *PSMS* and the best entries in the *ALvsPK* challenge data sets. We show, for reference, the best prior-knowledge entry (*Interim-all-prior*); the entry formed by the models described in Table 9 (*psmx_jmlr_run_I*); the best individual entries for each data set in the *ALvsPK* challenge(*IJCNN07AL*) (Guyon et al., 2007, 2008); the second-best entry of the agnostic track (*Logitboost-with-trees*) and our best ranked entry evaluated during the challenge *Corrida-final*. The best results in the agnostic track are shown in **bold**.

5. Discussion

In this section, we discuss the advantages and disadvantages of *PSMS* and perform a synthesis of our experiments aiming at better understanding how *PSMS* performs intensive search in hyperparameter space without overfitting the data.

5.1 Robust and Computationally Tractable Intensive Search

In Section 4 we reported experimental results that give evidence of the validity of the *PSMS* approach, demonstrating in particular that *PSMS* can outperform *PATSMS* through and at the end of the search, showing better convergence behavior and generalization performance. This is obtained at the expense of moderate additional computational complexity. This claim is supported by the theoretical analysis of the computational complexity (Section 3.4), indicating that computations are dominated by the number of learning machine trainings, and by the experiments (Section 4.2), indicating as few as 5 particles (learning machines) and 10 iterations (i.e., 50 trainings) are needed to attain the best performance. The efficiency of *PSMS* can be improved, for instance by preferably exploring learning machines, which have a lower computational cost of training. We explored successfully other heuristics, including subsampling training data, which reduced computations, at the expense of no performance degradation.

An analysis of the diversity of models tried by *PSMS* shows that this method is not biased towards models that tend to perform well individually. Investigating the operation of *PSMS* under different settings we found that the performance of *PSMS* is not significantly affected by modifying its hyperparameters. However, experimental results indicate that the use of an adaptive inertia weight may be helpful to explore the search space better.We also observed that certain parameter configurations allow the selection of competitive models, while reducing processing time. Section 5.4 provides a final set of practical recommendations.

Results of international competitions suggest that *PSMS* is competitive with model selection strategies specific to single algorithms and to strategies using prior domain knowledge. The latter is an important result because it shows that we can obtain competitive models without the need of an expert on the domain, a careful analysis to the data, or even machine learning knowledge.

5.2 Intensive Search without Overfitting

The findings summarized above provide empirical evidence suggesting *PSMS* is a reliable strategy for agnostic model selection. However, it is not obvious why *PSMS* succeeds in selecting competitive models without significantly overfitting data. Our hypothesis is that *PSMS* is able to avoid overfitting because of the way the search is guided: *PSMS* performs a broad search around promising solutions without focusing on reaching local minima. Solutions in *PSMS* are updated by taking into account information from both: the global-best solution ($\mathbf{p}_g$, weighted by c_2) and the individual-best solutions of each particle ($\mathbf{p}_{1,\dots,m}$, weighted by c_1), see Equations (1) and (2). The latter combined with an adaptive inertia weight and early stopping cause do not exaggerate the search at any local minima. Methods like *PATSMS*, on the contrary, update solutions by moving the pattern towards the local minimum nearest the initial solution, see Algorithm 2. Reaching exactly a local minimum causes *PATSMS* to learn peculiarities in the data set and does not necessarily result in obtaining a better predictive model.

The experiments we performed in Section 4.2.5 support the above conjecture. The results from Table 7 and Figures 4, 5 and 6 indicate that *PSMS* is able to avoid overfitting (to some extend) because of the way the search is guided. *PSMS* searches around good solutions without overdoing in terms of really fine-grained optimization. This sort of search can be considered as suboptimal in Dietterich's sense: *"In machine learning it is optimal to be suboptimal!"* (Dietterich, 1995). The latter statement makes reference to the well known fact that oversearching (i.e., trying to be optimal) in model selection can lead to select models that fit very well the peculiarities of the considered data set without deriving a general predictive rule (Dietterich, 1995; Jensen and Cohen, 2000; Quinlan and Cameron-Jones, 1995; Hastie et al., 2001; Loughrey and Cunningham, 2005). On the contrary, *PSMS* is able to *undercompute* because for updating solutions it considers local and global knowledge, information from past solutions (weighted by the inertia term) and randomness; note that the latter holds when a reasonable small number of iterations is performed. Furthermore, our experimental results provide empirical evidence that agrees with recent, yet traditional, explanations about why and how *PSO* works (Kennedy, 2008; Kennedy and Eberhart, 2001). Kennedy has argued the success of *PSO* is due to the fact that it performs a *"collaborative trial and error"* search (Kennedy, 2008). That is, *PSO* obtains good results mainly because the search is directed according both individual and social knowledge; the same conclusion derived from experimental results in this section. It is not surprising that distributed and collaborative computing can improve results of centralized (individualized) methods. For *FMS*, however, it is very interesting that updating solutions in a heterogeneous way allows us to avoid oversearching and, in consequence, overfitting; even when the *FMS* search space is infinite and has many local minima solutions.

5.3 Comparison with Related Work

A variety of approaches have been proposed to select parameters of specific methods for regression, classification, feature selection, discretization etcetera (Hastie et al., 2001; Bishop, 2006; Voss and Feng, 2002; Nelles, 2001; Guyon et al., 2006b; Kim et al., 2002; Hastie et al., 2001; Cawley and Talbot, 2007b; Boullé, 2007; Hue and Boullé, 2007). However, despite the potential advantages of *FMS* (namely generality, simplicity and competitiveness), this problem has been little studied because of the huge search space involved and because intensive search methods are prone to overfit the data (Gorissen et al., 2008; Escalante et al., 2007). Nevertheless, in the rest of this section we outline techniques used to avoid oversearching/overfitting in search that are applicable/related to

FMS. One should note that traditional model selection techniques just like Akaike and Bayesian information criteria, the minimum description length principle and the *VC*-dimension are not directly applicable to *FMS* and therefore they are excluded of analysis.

Grid search (with *CV*) is the widely used search-approach to model selection in practical applications (Momma and Bennett, 2002; Hsu et al., 2003). This method consists of defining a uniform grid over the search space where each point in the grid defines a solution; every point in the grid is evaluated and the point of lowest error is selected. The granularity of the grid determines both the performance of the selected solution and the efficiency of the search. A fine-grained grid may be inefficient and can lead to oversearching; while a sparse grid will result in low performance models. Note that the heterogeneousness of models and the variety of ranges for the models parameters make very difficult the application of grid search to the *FMS* problem; furthermore, the choice of an adequate granularity can be a serious problem. Other methods already used for parameter optimization that can be applied to *FMS* include: greedy search (Dietterich, 1995), random search (e.g., the bumping technique) (Hastie et al., 2001), *PS* (Bi et al., 2003; Momma and Bennett, 2002), evolutionary computation approaches (Engelbrecht, 2006; Gorissen et al., 2008; Angeline, 1998), and other swarm-optimization techniques (Kennedy and Eberhart, 2001; Engelbrecht, 2006). Note that despite one may think that exhaustive search is the best search option in model selection, this approach is impractical for most real world problems and when applicable it suffers from the oversearching phenomenon (Dietterich, 1995).

Early stopping has been widely used to prevent overfitting in search methods (Hastie et al., 2001; Engelbrecht, 2006; Loughrey and Cunningham, 2005). The goal of this heuristic is to stop searching/learning when the model starts overfitting the data. There are several variants to stop the search: after a small number of iterations, when no improvement is found after a number of iterations, when a solution of acceptable performance has been found, etcetera. A problem with early stopping is that premature stopping the algorithm would lead to selecting a solution that has not converged yet, while a late stopping of the search will cause severely overfitting the data because of oversearching. For *PSMS* we found that a small number of iterations can be enough to obtain satisfactory results in benchmark data, although the number of iterations is problem dependent; therefore, we can adopt other stopping criteria for *FMS* in the future.

Randomness has bring into play in machine learning in order avoid overfitting and to escape from local minima in search (Hastie et al., 2001; Bishop, 2006; Kirkpatrick et al., 1983; Kennedy and Eberhart, 2001). In learning algorithms, it has been successfully used to prevent overfitting and to obtain better predictors; learning methods that use randomness include bagging classifiers (Hastie et al., 2001), neural and deep belief networks (Hastie et al., 2001; Hinton et al., 2006) and randomized decision-tree algorithms (Breiman, 2001; Geurts et al., 2006). Bootstrapping is a technique (used in bagging and random forest classifiers) based on random sampling that has been widely used to estimate the generalization performance of methods as an alternative to *CV* (Hastie et al., 2001). In *PSMS* randomness played an important role because it introduces diversity into the search process and allows *PSMS* to avoid local minima. Furthermore, the subsampling strategy we used to increase the speed of *PSMS* is related to bootstrapping; in future work on *PSMS* we will explicitly consider different subsampling estimations for the selection of the final model.

As the adaptive inertia weight in *PSO*, see Section 2, there are parameters in other algorithms that aim to avoid overfitting by exploring the search space both globally and locally; examples are the temperature parameter in simulated annealing (Kirkpatrick et al., 1983) and the momentum term in on-line gradient descend and backpropagation (Qian, 1999). The ridge in ridge-regression and weight decay in neural networks training are also related to the inertia weight. Model averaging and the use of ensembles have proved to be helpful to improve predictions and avoid overfitting; this is because different models have different biases that in average result in improved performance (Hastie et al., 2001; Bishop, 2006). Future work includes in *PSMS* consists of combining particles in order to improve the performance of the swarm strategy. Finally, adding noise to the training data is another overfitting avoidance mechanism in model selection that also can be used with *PSMS*.

5.4 A Practical Guide to PSMS

In this section we describe the way *PSMS* can be put in practice in any binary classification problem. Due to the simplicity and generality of the approach below we describe a practical guide to use the *Matlab*^{R} implementation of *PSMS* (included in the *CLOP* toolbox). It is assumed that the user has available a data set (in Matlab^R format) with N samples for binary classification: a matrix $\mathbf{X}_{N\times d}$ contains the N training samples of dimensionality d and a vector $\mathbf{Y}_{N\times 1}$ their respective labels ($y_i \in [-1,1]$). After downloading and installing *CLOP* (Saffari and Guyon, 2006), *PSMS* can be applied to any data set by typing the following *Matlab*^{R} code:

```
%% load your data into the MatlabR workspace
1: >> load train_data.mat;
%%  Create A Clop Data-Object
2: >> D = data(X,Y);
%% Create A CLOP PSMS-Object with default parameters
3: >> P = psmsx;
%% Perform PSMS
4: >> [Dat, Res] = train(P, D);
%% Train the selected model with the full training set
5: >> [Odat, TrM] = train(Res.Best_Model,D);
%% Create a test data set, note that Yt can be empty
6: >> load test_data; Dt = data(Xt,Yt);
%% Test the selected and trained model on unseen data Dt
7: >> [Pred] = test(TrM, Dt);
%% Estimate the model's performance on unseen data Dt, if Yt is available
8: >> [BER] = balanced_errate(Pred.X, Pred.Y);
%% Analyze the ROC performance of the selected model
9: >> roc(Pred);
```

Note that steps 1–2 and 5–9 are associated with loading the data and the evaluation of the selected model, respectively; which are operations not attained to *PSMS*. Steps 3 and 4 will create the *PSMS* object and will start the search, respectively. Besides the selected model, the output of the search (**Res**, line 4), is a structure with useful information about the search process, see the *PSMS* documentation (Escalante, In preparation, 2009).

In Section 4.2 were presented experimental results that suggest that there is not significant difference in performance by modifying most of the *PSMS* hyperparameters. Therefore, one can choose parameter settings for *PSMS* that make practical its application without a significant decrement of performance. Below are shown recommended values for the *PSMS* hyperparameters. These parameters and other options of the current implementation can be modified very simply (Escalante, In preparation, 2009).

Recommended PSMS parameters:

Weight for individual best solution	$c_1 = 2$
Weight for global best solution	$c_2 = 2$
Adaptive inertia weight	$\mathbf{W} = (1.2, 0.5, 0.4)$
Number of iterations	$I = 50$
Swarm size	$m = 5$
Folds in *CV*	$k = 2$
Subsampling factor	$SF = 1$ (as small as possible in large data sets)

6. Conclusions

In this paper we proposed Particle Swarm Model Selection (*PSMS*), that is, the application of Particle Swarm Optimization (*PSO*) to the problem of Full Model Selection (*FMS*). Given a data set, the *FMS* problem consists of selecting the combination of preprocessing, feature selection and learning methods that obtains the lowest classification error. *FMS* also includes hyperparameter optimization for the selected methods. This approach to the model selection problem has the following advantages. First, the generality of the approach allows us to consider different model types (for preprocessing, feature selection and learning) and a variety of methods. Second, *PSMS* can be applied to any data set, since neither domain knowledge nor machine learning knowledge is required, therefore it can be considered a black-box model selection method. Third, and most importantly, competitive and yet simple models can be obtained with *PSMS*. We provide empirical evidence that shows that *PSMS* can be applied efficiently, without a significant loss of accuracy, by using a subsampling heuristic and parameter settings that reduce the computational cost.

The simplicity of *PSO* and its proven performance, comparable to that of evolutionary algorithms, make this search algorithm well suited for *FMS*. However, the application of any other stochastic optimization strategy is also possible. The main advantage of *PSO* is that a single equation for updating solutions is needed, as opposed to evolutionary algorithms where methods for representation, mutation, cross-over, speciation and selection have to be considered. Interestingly, the way the search is guided in *PSMS* allows it obtaining competitive models without significantly overfitting. Experimental results in benchmark data show superior performance of *PSMS* when compared to Pattern Search Model Selection (*PATSMS*), a direct search method that constitutes a competitive baseline.

Results obtained by models selected with *PSMS* in the framework of a model selection challenge show that it is a very competitive model selection method, despite its simplicity and generality. In such competitions, models selected with *PSMS* were always among the top ranking models, together with methods performing solely hyperparameter selection in a given model family and methods relying on prior knowledge. This demonstrates that, via the use of *PSO*, *FMS* is a viable strategy for model selection. This is remarkable because we noted in previous competitions (Guyon et al., 2005; Guyon et al., 2006b) that each data set had a different best performing method, yet researchers performing *FMS* (in an effort to find the model family best suited to a given problem) were not successful. The participants, which obtained the best results on average over all data sets restricted themselves to hyperparameter selection in one given model family. In contrast, in this paper we demonstrated the viability of *FMS* using the *PSO* search strategy. Our work paves the way to the use of intensive search techniques to perform *FMS* in the entire model space of machine learning toolkits. With the increasing availability of diverse and sophisticated machine learning toolkits and the improvements in computing power, we foresee that *FMS* will become an effective methodology.

Current work includes the use of *PSMS* for the selection of model-members for ensembles and the hierarchical application of *PSO* for *FMS* and hyperparameter optimization. *PSMS* is currently being applied to different tasks, including galaxy classification, automatic image annotation, object recognition and text classification. Future work includes the introduction of a penalty-term into the fitness function; such that (computationally) inexpensive models be favored by PSMS. The extension of *PSMS* to the multi-class classification and regression problems is another future work direction.

Acknowledgments

We would like to thank the organizers and participants of the NIPS multi-level inference workshop and model selection game, and of the *IJCNN ALvsPK* challenge. The first author thanks the UNIPEN foundation and the US National Science Foundation for travel support. We also thank editors and anonymous reviewers for their useful comments that have help us to improve this paper. We thank Dr. Eduardo Morales for his useful suggestions about the content of the paper. Last, but not least, we thank Dr. Aurelio López and INAOE for the provided support.

References

P. J. Angeline. Evolutionary optimization vs particle swarm optimization: Philosophy and performance differences. In *Proceedings of the 7th Conference on Evolutionary Programming*, volume 1447 of *LNCS*, pages 601–610, San Diego, CA, March 1998. Springer.

Y. Bengio and N. Chapados. Extensions to metric-based model selection. *Journal of Machine Learning Research*, 3:1209–1227, 2003.

J. Bi, M. Embrechts K. P. Bennett, C. M. Breneman, and M. Song. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, Mar(3):1229–1243, Mar 2003.

C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

M. Boullé. Report on preliminary experiments with data grid models in the agnostic learning vs prior knowledge challgenge. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1802–1808, 2007.

L. Breiman. Random forest. *Machine Learning*, 45(1):5–32, 2001.

G. Cawley. Leave-one-out cross-validation based model selection criteria for weighted ls-svms. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2006)*, pages 2970–2977, Vancouver, Canada, July 2006.

G. Cawley and N. L. C. Talbot. Agnostic learning vs prior knowledge in the design of kernel machines. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1444–1450, Orlando, Florida, 2007a.

G. Cawley, G. Janacek, and N. L. C. Talbot. Generalised kernel machines. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1439–1445, Orlando, Florida, 2007.

G. C. Cawley and N. L. C. Talbot. Preventing over-fitting during model selection via bayesian regularisation of the hyper-parameters. *Journal of Machine Learning Research*, 8:841–861, April 2007b.

M. Clerc and J. Kennedy. The particle swarm: Explosion, stability and convergenge in a multi-dimensional complex space. *IEEE Transactions on on Evolutionary Computation*, 6(1):58–73, February 2002.

J. Demsar. Statistical comparisons of classfifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, January 2006.

J. E. Dennis and V. J. Torczon. Derivative-free pattern search methods for multidisciplinary design problems. In *Proceedings of the AIAA / USAF / NASA / ISSMO Symposium on Multidisciplinary Analysis and Optimizatino*, pages 922–932, 1994.

T. Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27(3): 326–327, 1995. ISSN 0360-0300.

A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2006.

H. J. Escalante. Particle swarm optimization for classifier selection: A practical guide to psms. `http://ccc.inaoep.mx/~hugojair/psms/psms_doc.pdf`, In preparation, 2009.

H. J. Escalante, M. Montes, and E. Sucar. Psms for neural networks on the ijcnn 2007 agnostic vs prior knowledge challenge. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1191–1197, Orlando, FL, USA., 2007.

V. Franc and V. Hlavac. The statistical pattern recognition toolbox. `http://cmp.felk.cvut.cz/cmp/software/stprtool/index.html`, 2004.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–407, 2000.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006. ISSN 0885-6125.

T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomeld, and E. S. Lander. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286: 531–537, October 1999.

D. Gorissen. Heterogeneous evolution of surrogate models. Master's thesis, Katholieke Universiteit Leuven, Belgium, June 2007.

D. Gorissen, L. De Tommasi, J. Croon, and T. Dhaene. Automatic model type selection with heterogeneous evolution: An application to rf circuit block modeling. In *IEEE Proceedings of WCCI 2008*, pages 989–996, 2008.

V.G. Gudise and G.K. Venayagamoorthy. Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium, 2003. (SIS03)*, pages 110–117, 2003.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182, 2003.

I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the nips 2003 feature selection challenge. In *Advances in Neural Information Processing Systems 17*, pages 545–552. MIT Press, Cambridge, MA, 2005.

I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature Extraction, Foundations and Applications*. Series Studies in Fuzziness and Soft Computing. Springer, 2006a.

I. Guyon, A. Saffari, G. Dror, and J. M. Buhmann. Performance prediction challenge. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2006)*, pages 2958–2965, Vancouver, Canada, July 2006b.

I. Guyon, A. Saffari, G. Dror, G. Cawley, and O. Guyon. Benchmark datasets and game result summary. In *NIPS Workshop on Multi-level Inference and the Model Selection Game*, Whistler, Canada, December 2006c.

I. Guyon, A. Saffari, G. Dror, and G. Cawley. Agnostic learning vs prior knowledge challenge. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1232–1238, Orlando, Florida, 2007.

I. Guyon, A. Saffari, G. Dror, and Gavin Cawley. Analysis of the ijcnn 2007 competition agnostic learning vs. prior knowledge. *Neural Networks*, 21(2–3):544–550, 2008.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Verlag, New York, 2001.

E. Hernández, C. Coello, and A. Hernández. On the use of a population-based particle swarm optimizer to design combinational logic circuits. In *Evolvable Hardware*, pages 183–190, 2004.

G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. ISSN 0899-7667.

C. W. Hsu, C. C. Chang, and C. J. Lin. A practical guide to support vector classification. Technical report, Taipei, 2003. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

C. Hue and M. Boullé. A new probabilistic approach in rank regression with optimal bayesian partitioning. *Journal of Machine Learning Research*, 8:2727–2754, December 2007.

D. Jensen and P Cohen. Multiple comparisons in induction algorithms. *Machine Learning*, 38(3): 309–338, 2000. ISSN 0885-6125.

J. Kennedy. How it works: Collaborative trial and error. *International Journal of Computational Intelligence Research*, 4(2):71–78, 2008.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the International Conference on Neural Networks*, volume IV, pages 1942–1948. IEEE, 1995.

J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, volume 2, pages 1671–1676, 2002.

Y. Kim, N. Street, and F. Menczer. Evolutionary model selection in unsupervised learning. *Intelligent Data Analysis*, 6:531–556, 2002.

S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598): 671–680, 1983.

J. Loughrey and P. Cunningham. Overfitting in wrapper-based feature subset selection: The harder you try the worse it gets. In F. Coenen M. Bramer and T. Allen, editors, *Proceedings of AI-2004, the Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, Research and Development in Intelligent Systems XXI, pages 33–43, 2005.

R. Lutz. Logitboost with trees applied to the wcci 2006 performance prediction challenge datasets. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2006)*, pages 1657– 1660, Vancouver, Canada, July 2006.

S. Mika, G. Rätsch, J. Weston, B. Schölkopf, A. J. Smola, and K.-R. Müller. Invariant feature extraction and classification in kernel spaces. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 526–532, Cambridge, MA, 2000. MIT Press.

M. Momma and K. Bennett. A pattern search method for model selection of support vector regression. In *Proceedings of SIAM Conference on Data Mining*, 2002.

O. Nelles. *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. Springer, 2001.

E. Ozcan and C. K. Mohan. Analysis of a simple particle swarm optimization system. In *Intelligent Engineering Systems Through Artificial Neural Networks*, pages 253–258, 1998.

E. Pranckeviciene, R. Somorjai, and M. N. Tran. Feature/model selection by the linear programming svm combined with state-of-art classifiers: What can we learn about the data. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1422–1428, 2007.

N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Netw.*, 12(1): 145–151, 1999. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/S0893-6080(98)00116-6.

J. R. Quinlan and R. M. Cameron-Jones. Oversearching and layered search in empirical learning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1019–1024, 1995.

G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for adaboost. *Mach. Learn.*, 42(3):287–320, 2001. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1007618119488.

J. Reunanen. Model selection and assessment using cross-indexing. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1674–1679, 2007.

M. Reyes and C. Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 3(2):287308, 2006.

Y. Robinson, J. Rahmat-Samii. Particle swarm optimization in electromagnetics. *IEEE Transactions on Antennas and Propagation*, 52(2):397– 407, February 2004.

A. Saffari and I. Guyon. Quickstart guide for clop. Technical report, Graz University of Technology and Clopinet, May 2006. http://www.ymer.org/research/files/clop/QuickStartV1.0.pdf.

J. Salerno. Using the particle swarm optimization technique to train a recurrent neural model. In *Proceedings of the Ninth International Conference on Tools with Artificial Intelligence*, pages 45–49, 1997.

C. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In Jude W. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning*, pages 515–521, Madison, WI, USA, 1998.

Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII*, pages 591–600, New York, 1998. Springer-Verlag.

Y. Shi and R. C. Eberhart. Emprirical study of particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, pages 1945–1949, Piscataway, NJ, USA, 1999. IEEE.

S. Sonnenburg. Nips workshop on machine learning open source software. http://www2.fml.tuebingen.mpg.de/raetsch/workshops/MLOSS06/, December 2006.

J.A.K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(1):293–300, 1999.

F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, Sudafrica, November 2001.

F. van der Heijden, R. P.W. Duin, D. de Ridder, and D. M.J. Tax. Prtools: a matlab based toolbox for pattern recognition. http://www.prtools.org/, 2004.

M. Voss and X. Feng. Arma model selection using particle swarm optimization and aic criteria. In *Proceedings of the 15th IFAC World Congress on Automatic Control*, 2002.

J. Weston, A. Elisseeff, G. BakIr, and F. Sinz. The spider machine learning toolbox. http://www.kyb.tuebingen.mpg.de/bs/people/spider/, 2005.

J. Wichard. Agnostic learning with ensembles of classifiers. In *Proceedings of the 20th International Joint Conference on Neural Networks*, pages 1753–1759, 2007.

J. Wichard and C. Merkwirth. Entool - a matlab toolbox for ensemble modeling. `http://www.j-wichard.de/entool/`, 2007.

I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 4:67–82, 1997.

H. Xiaohui, R. Eberhart, and Y. Shi. Engineering optimization with particle swarm. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium, 2003. (SIS03)*, pages 53–57, 2003.

H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE Transactions on Power Systems*, 15(4):1232–1239, Jan 2001.