Self Adaptive Support Vector Machine: A Multi-Agent Optimization Perspective

Nicolas Couellan, Tom Jorquera, Jean-Pierre Georgé, Sophie Jan

Self-adaptive Support Vector Machine: A multi-agent optimization perspective

Nicolas Couellan [a,*], Sophie Jan [a], Tom Jorquera [b], Jean-Pierre Georgé [b]

[a] Institut de Mathématiques de Toulouse, Université de Toulouse, UPS IMT, F-31062 Toulouse Cedex 9, France
[b] Institut de Recherche en Informatique de Toulouse, Université de Toulouse, UPS IRIT, F-31062 Toulouse Cedex 9, France

Keywords:
Support Vector Machine
Classification
Model selection
Multi-agent systems
Collaborative process
Complex systems optimization

A B S T R A C T

Support Vector Machines (SVM) have been in the forefront of machine learning research for many years now. They have very nice theoretical properties and have proven to be efficient in many real life applications but the design of SVM training algorithms often gives rise to challenging optimization issues. We propose here to review the basics of Support Vector Machine learning from a multi-agent optimization perspective. Multi-agents systems break down complex optimization problems into elementary "oracle" tasks and perform a collaborative solving process resulting in a self-organized solution of the complex problems. We show how the SVM training problem can also be "tackled" from this point of view and provide several perspectives for binary classification, hyperparameters selection, multiclass learning as well as unsupervised learning. This conceptual work is illustrated through simple examples in order to convey the ideas and understand the behavior of agent cooperation. The proposed models provide simple formulations of complex learning tasks that are sometimes very difficult to solve with classical optimization strategies. The ideas that are discussed open up perspectives for the design of new distributed cooperative learning systems.

1. Introduction

SVMs are known as powerful mathematical tools for classification as well as regression tasks (Cristianini & Shawe-Taylor, 2001; Scholkopf & Smola, 2001). They have proven good capabilities for the classification of complex and large datasets. Many successful implementations have been developed for various types of applications: medical diagnosis (Fung & Mangasarian, 2006; Lee, Mangasarian, & Wolberg, 1999), manufacturing (Balakrishna, Raman, Santosa, & Trafalis, 2008; Gilbert, Raman, Trafalis, Obeidat, & Aguirre-Cruz, 2009), meteorology (Trafalis, Adrianto, & Richman, 2007), hand digits recognition (Decoste & Schölkopf, 2002), fraud detection (Chan & Stolfo, 1998), and many others.

The underlying concepts are based on empirical risk theory (Vapnik, 1998) and the available algorithms make use of convex optimization techniques (Boyd & Vandenberghe, 2004). A strong focus is now put on the ever increasing size of datasets and new algorithms based on first order stochastic optimization have now emerged (Bottou, 1997; Bousquet & Bottou, 2007).

Training speed has therefore been greatly reduced by combining "cheap" first order optimization techniques with stochastic frameworks that process only samples of data at a time. The increasing dimensions of datasets and the emergence of online applications where data is only available dynamically bring new and great challenges to the machine learning and optimization communities. Additionally, most learning algorithms require the selection of hyperparameters. These parameters are usually problem dependent and control for example the trade-off between training accuracy and generalization performance (ability to generalize prediction to unseen data) or some mapping function (or usually its corresponding kernel function) that will transform the problem into a linear problem. Selecting the optimal parameters is known as *model selection*. It is often critical in real life applications and no matter how effective and fast the training procedure is, if the hyperparameters are not tuned in a proper way, the resulting model will not generalize well to new data (Hastie, Rosset, Tibshirani, & Zhu, 2003/04). Model selection is usually done through the so-called *k-fold Cross Validation* (CV) where the data is split in k subsets and k training procedures are performed with $k-1$ subsets as training data and the remaining subset as validation data (swapping the training and validation subsets for each training procedure) (Hastie, Tibshirani, & Friedman, 2009). The k-fold cross validation is combined with a

* Corresponding author.
E-mail addresses: nicolas.couellan@math.univ-toulouse.fr (N. Couellan), sophie.jan@math.univ-toulouse.fr (S. Jan), tom.jorquera@irit.fr (T. Jorquera), jean-pierre.george@irit.fr (J.-P. Georgé).

grid search method to estimate the optimal hyperparameters. The CV is statistically valid but has two major drawbacks: (1) if datasets are large, it can become extremely time expensive and in practice unrealistic, (2) the grid search will only go over a very limited and discrete set of values of the hyperparameters while the optimal parameters could lie in between grid points.

To overcome these drawbacks, some techniques making use of bi-level SVM formulations have been investigated. The idea is to perform descent techniques with respect to the hyperparameters while trying to find the optimal separating hyperplane (Couellan & Wang, 2014; Du, Peng, & Terlaky, 2009; Kunapuli, Bennett, Hu, & Pang, 2008). These techniques address successfully the problem of auto-selecting the trade-off (training accuracy/generalization performance) parameter but do not extend to the problem of tuning the kernel function parameter that arises in the case of nonlinear learning systems.

Alternatively, to design learning systems that are able to adjust dynamically to online data as well as being able to self-adjust the problem hyperparameters, we investigate a novel approach. We propose to look at the SVM model and its parameter selection as a whole and perform optimization on a system involving various natures of variables. These variables are interconnected in a calculus network so that it can be considered as a complex system. To solve these types of naturally complex tasks, one way is to make use of an *Adaptive Multi-Agent System (AMAS)* where autonomous agents are each given part of the optimization problem and cooperation between them takes place to solve the overall problem.

A multi-agent system (MAS) (Weiss, 1999) is a system composed of several autonomous software entities (the agents), interacting among each others (usually by sending information and request messages) and with their environment (by observing and modifying it). The *autonomy* of an agent is the fundamental characteristic that differentiates it from, for example, the computer science concept of object. While an object is a passive entity encapsulating some data and functions, waiting to be solicited, an agent is capable of reacting to its environment and displaying pro-activity (activity originating from its own decision). From this comparison it should be clear that the concept of agent is, like the concept of object, the building brick of a paradigm which can be used to model a complex reality. And indeed, agents have been used in a great variety of fields, a fact which can contribute to explain the difficulty to produce a unified definition of the concept.

While it is not true for all MAS, some interesting properties can be achieved when taking advantage of the autonomy of the agents. This autonomy, coupled with an adequate behavior of the agents, can lead to systems able to adjust, organize, react to changes, etc. without the need for an external authority to guide them. These properties are gathered under the term self-* capabilities (Di Marzo Serugendo et al., 2011) (self-*tuning*, self-*organizing*, self-*healing*, self-*evolving*...). Not all MAS necessarily present all of these self-* capabilities but, as a result of building a system from autonomous and locally situated agents, many MAS will exhibit them to some degree. Consequently, MAS are often relevant for dynamically taking into account changes in their environment. For example, a MAS in charge of regulating the traffic of packets in a computer network could be able to react efficiently to the disappearance of some of the relay nodes.

MAS have been applied to a great variety of fields: social simulation, biological modeling, systems control, robotics, etc. and agent-oriented modeling can be seen as a programming paradigm in general, facilitating the representation of a problem.

A particular approach to MAS relying strongly on self-* properties is the *AMAS* technology and underlying theory (Georgé, Edmonds, & Glize, 2004). A designer following this approach focuses on giving the agent a local view of its environment, means to detect problematic situations and guidelines to act in a *cooperative* way, meaning that the agents will try to achieve their goals while respecting and helping the other agents around them as best as they can. The fact that the agents do not follow a global directive towards the solving of the problem but collectively build this solving, produces an *emergent problem solving process* that explores the search space of the problem in original ways.

Modeling SVMs as AMAS has several advantages over more classical mathematical strategies. It avoids running unnecessary training procedures for non-optimal regions of the hyperparameters space. The selected optimal hyperparameters values are more accurate as they are not constrained on a grid but can take freely any value of the space. Finally, the use of AMAS opens the door to parallelization, decomposition and distributed computation. The work presented here can be seen as preliminary work to illustrate the possible perspectives that further research along this area could give. Current research in SVM has generated a great deal of work on model selection for binary classification and single kernel techniques with possible but sometimes expensive (complexity wise) extensions to multi-class and multiple kernel variants. Clearly the use of AMAS gives more flexible and more natural ways to extend models to more complicated contexts.

The article is organized as follows. Section 2 recalls the basic mathematics of classification with SVMs, Section 3 describes the principles of AMAS. In Section 4 we propose models to perform training tasks with AMAS and in Section 5 we incorporate the model selection concepts into our models. Finally, in Section 6 we provide several numerical examples on simple illustrative problems. Section 7 concludes the paper.

2. SVM classification

2.1. Linear classification

Consider a set of training vectors $\{x_i \in \mathbb{R}^n,\ i = 1,\dots,L\}$ and its corresponding set of labels $\{y_i \in \{-1,1\},\ i = 1,\dots,L\}$, where L is the number of training points and n is the number of attributes of each training point.

The soft margin SVM training problem can be expressed as follows (see for example Cristianini & Shawe-Taylor (2001), Scholkopf & Smola (2001) for further details on the construction of the problem):

$$\begin{aligned} &\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L}\xi_i \\ &\text{subject to} \quad y_i(w^\top x_i + b) + \xi_i \geqslant 1, \quad i = 1,\dots,L, \\ &\xi_i \geqslant 0, \quad i = 1,\dots,L, \end{aligned} \tag{1}$$

where ξ_i is a slack variable associated to a penalty term in the objective with magnitude controlled by C, a problem specific parameter. The vector w is the normal vector to the separating hyperplane ($w^\top x + b = 0$) and b is its relative position to the origin.

Problem (1) maximizes the margin $\frac{2}{\|w\|}$ between the two separating hyperplanes $w^\top x + b = 1$ and $w^\top x + b = -1$. The use of slack variables ξ_i penalizes data points that would fall on the wrong side of the hyperplanes.

In the constraints, observe that $\xi_i \geqslant \max\{0, 1 - y_i(w^\top x_i + b)\}$, therefore at optimality we have the equality:

$$\xi_i = \max\{0, 1 - y_i(w^\top x_i + b)\}.$$

Indeed, the ith point is either correctly classified ($\xi_i = 0$) or penalized ($\xi_i = 1 - y_i(w^\top x_i + b)$). Consequently, we can reformulate Problem (1) as an unconstrained optimization problem:

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L}\max\{0, 1 - y_i(w^\top x_i + b)\}.$$

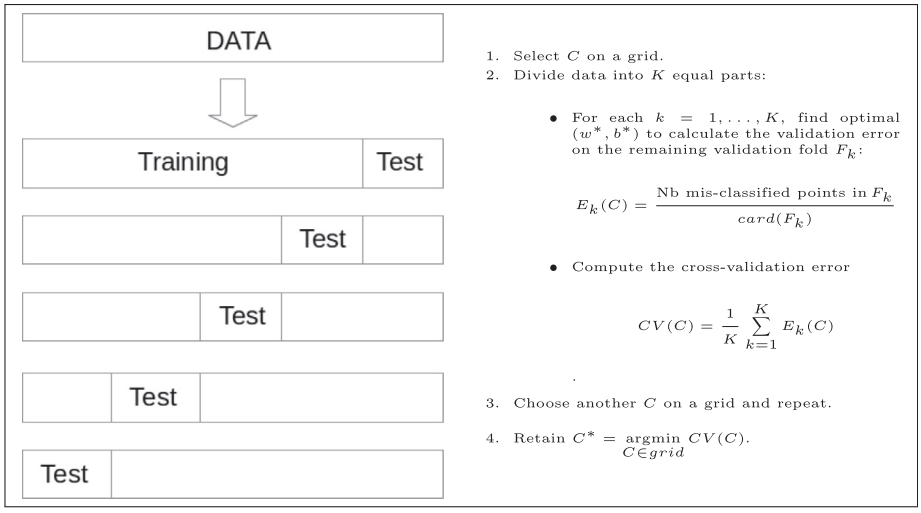

Fig. 1. K-fold cross-validation.

The term $\max\{0, 1 - y_i(w^\top x_i + b)\}$ is known in statistics as the "hinge loss". Other types of losses could be used and generally, we can write the problem as:

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L}\ell(y_i, w^\top x_i + b)$$

where ℓ is the loss function.

2.2. Nonlinear classification

In real life situations, the data is usually not linearly separable and the idea is to map the data from the input space into a higher dimensional space F by setting:

$$w = \sum_{i=1}^{L}\alpha_i\varphi(x_i),$$

where $\alpha_i \in \mathbb{R}$, $i = 1, \ldots, L$ and φ is the map from the input space to F:

$\varphi : \mathbb{R}^n \to \mathbb{R}^{n'}$ and $n' \gg n$.

By replacing w in Problem (1) formulated in the F space, we obtain the following "kernelized" primal problem:

$$\begin{aligned} &\min_{\alpha,b,\xi} \quad \frac{1}{2}\alpha^T\bar{K}\alpha + C\sum_{i=1}^{L}\xi_i \\ &\text{subject to} \quad y_i\left(\sum_{j=1}^{L}\alpha_j\bar{k}(x_i, x_j) + b\right) + \xi_i \geqslant 1, \quad i = 1, \ldots, L, \\ &\xi_i \geqslant 0, \quad i = 1, \ldots, L, \end{aligned} \tag{2}$$

where $\alpha \in \mathbb{R}^L$ and K is the kernel matrix defined as follows:

$$\bar{k}_{ij} = \varphi(x_i)^T\varphi(x_j) = \bar{k}(x_i, x_j)$$

with $\bar{k}$ being a kernel function, commonly chosen as a polynomial, a sigmoid function or a Radial Basis Function (RBF). In this work, the Gaussian RBF function is chosen and the elements of matrix $\bar{K}$ will be calculated as follows:

$$\bar{k}_{ij} = \bar{k}(x_i, x_j) = e^{\left(-\frac{\|x_i - x_j\|^2}{2\sigma}\right)}, \quad i, j = 1, \ldots, L, \text{ with } \sigma > 0.$$

2.3. Model selection and k-fold cross-validation

Commonly, the selection of C (for linear classifier) or (C, σ) (for nonlinear classifier) is done using a statistical technique known as k-fold cross-validation (CV). The idea is to partition the data into k equal size parts (folds), and perform training with $k - 1$ folds and compute a testing error on the remaining fold (called the validation fold). The process is performed k times so as to choose each time a new validation fold. The error found each time we select a validation fold is averaged over the k-th runs to give the so-called k-fold CV error (see Fig. 1).

To select the optimal C parameter (or (C, σ)), one usually decides on a grid search for C (for example C taking the values $0.01, 0.1, 1, 10, 100, 1000, 10000, \ldots$.) and computes the CV error for each value on the grid (or on a 2-dimensional grid search for (C, σ)). The value of C that gives the best CV error will be selected. If N_G is the number of values on the grid, one will have to solve $k \times N_G$ training problems. If the size of the training problem is large, this may require a huge amount of time.

3. Adaptive Multi-agent System (AMAS)

In Jorquera et al. (2013), the development of new tools for solving complex continuous optimization problems is investigated.[1] In this research work, a new multi-agent based algorithm for distributed continuous problem solving is proposed. The main idea of the algorithm is to represent an optimization problem as a graph, where each node of the graph is handled by an agent with a local decision process. Each agent has the responsibility to correctly propagate the information to its upper or lower neighbors in order to ensure the convergence of the problem towards an optimum solution. As an illustration, consider the following general composite optimization problem:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} \quad f_p \circ f_{p-1} \circ \ldots \circ f_1(x) \\ &\text{subject to} \quad h_m \circ h_{m-1} \circ \ldots \circ h_1(x) \geqslant 0 \end{aligned} \tag{3}$$

where

$$f_1 : \mathbb{R}^n \to \mathbb{R}^{n_1}, \quad f_2 : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}, \ldots, f_p : \mathbb{R}^{n_{p-1}} \to \mathbb{R}^{n_p},$$

$$h_1 : \mathbb{R}^n \to \mathbb{R}^{l_1}, \quad h_2 : \mathbb{R}^{l_1} \to \mathbb{R}^{l_2}, \ldots, h_m : \mathbb{R}^{l_{m-1}} \to \mathbb{R}^{l_m}.$$

Note that most complex optimization problems can be cast into this very general form. The multi-agent graph associated to Problem (3) is shown in Fig. 2. In this graph, there are 5 types of agents:

[1] Research project funded by the ANR (French National Research Agency) under the name of Integrated Design for Complex Systems (ID4CS), see also ID4CS (2013).

(1) the decision variable x, (2) intermediate variables c_i and z_j ($i = 1, \ldots, m$ and $j = 1, \ldots, p$), (3) the model agents calculating the various function values, (4) the constraint agent ensuring that the constraint is satisfied and finally (5) the objective agent that will force the system to minimize the objective. Even though the general composite formulation is rather complex in Problem (3) and its solving through classical optimization techniques would probably require complex chain rule derivation for gradient based descent, the AMAS system can be expressed in a very simple manner. Each agent is only executing one elementary task: one function evaluation, one objective or one constraint check. The complex composite structure is broken down naturally among the agents.

Thus the MAS is the representation of the problem to be solved, with the relationships between agents reflecting the natural structure of the problem. It is worth underlining the fact that this transformation (i.e. the agentification) is completely automatic, as it is fully derived from the expression of the problem. As a result, and contrary to most of the classical optimization techniques, no expert knowledge is required, neither in MAS nor on the optimization problem that is to be solved.

Algorithm 1 is a quick overview of the behavior of each agent type (*model*, *variable*, *output*, *constraint* and *objective* agents), which are further described in the following sub-sections. This is a synthetic description as it is not the focus of this article, a comprehensive and detailed description can be found in Jorquera et al. (2013).

Algorithm 1: Agents Behaviors Synthesis

```
behavior of Model Agent
    repeat
        analyze received messages
        if received new information messages then
            recalculate outputs
            inform depending agents
        end
        if received new requests then
            use optimizer to find adequate inputs
            propagate requests to input agents
        end
    until resolution end
behavior of Variable Agent
    repeat
        analyze received messages
        if received new requests then
            adjust value
            inform depending agents
        end
    until resolution end
behavior of Output Agent
    repeat
        analyze received messages
        if received new information messages then
            update its value
            inform depending agents
        end
        if received new requests then
            transmit requests to model agent
        end
    until resolution end
behavior of Constraint/Objective Agent
    repeat
        analyze received messages
        if received new information messages then
            update its value
            use optimizer to find adequate inputs
            send new requests to variable/output agents
        end
    until resolution end
```

3.1. Information exchange

The solving process —constituted by the collective behavior of the agents— basically relies on two types of messages: *inform* and *request* messages exchanged between connected agents and propagated as required.

Exchange of *inform* messages can be seen as a *simulation* mode where the main concern of the agents is to ensure consistency between the values of the design variables and the produced outputs. To this end, the agents will propagate *Inform* messages through the system that carry new values v. The exact semantic of this information slightly changes depending on which agents are involved:

- If the message is sent from a value agent (variable or output) to a model or criterion agent, it indicates to the receiving agent that the sending agent has changed its value.
- If the message is sent from a model agent to an output agent, it indicates to the receiving agent that the model has calculated its new value (or values) because of changes on its inputs.

Exchange of *change-value requests* sent by the criteria agents, resulting in cooperatively decided adjustments done by the *design variables*, constitutes the *optimization* mode. These adjustments lead to new values computed by the models, resulting in the satisfaction or dissatisfaction of the criteria agents. Basically, during solving, the criteria agents try to improve their local goals. That is, the constraint agents try to keep their constraint satisfied, while the objective agents try to improve their objective. To this end, they send *Request* messages to the agents controlling their inputs, asking them to change value by sending them a new target value v, which is more an indication of a *direction* than a strict target.

Depending of the agent types, this information is either forwarded after processing or actually treated:

- If the receiving agent is *not* a variable agent, the agent has to propagate these requests toward the variable agents in the most adequate way (making changes depending on the models, choosing the relevant inputs, solving conflicts, etc.).
- If the receiving agent is a variable agent, it has to take into account the request and try to satisfy it as best as it can, by managing conflicts and making the changes with specific dynamics so as to ensure efficient convergence of the solving process.

3.2. Model agent

A *model agent* takes charge of a model of the problem. It interacts with the agents handling its inputs (which can be *variable* or *output agents*) and the *output agents* handling its outputs. Its individual goal is to maintain the consistency between its inputs and its outputs. To this end, when it receives a message from one of its inputs informing it of a value change, a *model agent* recalculates the output values of its model and informs its *output agents* of their new value. On the other part, when a *model agent* receives a message from one of its *output agents* it translates and transmits the request to its inputs. To find the input values corresponding to a specific desired output value, the *model agent* uses an external optimizer. This optimizer is provided by the engineer based on expert domain-dependent knowledge regarding the structure of the model itself. It is important to underline that the optimizer is used only to solve the local problem of the *model agent*, and is not used to solve the global problem.

3.3. Variable agent

This agent represents a *design variable* of the problem. Its individual goal is to find a value which is the best equilibrium among

all the requests it can receive (from models and criteria for which it is an input). The agents using the variable as input can request a value change to the input variable agent. When changing value, the agent informs all connected agents of its new value. To find its new value, the *variable agent* uses an exploration strategy based on *Adaptive Value Trackers* (AVT) (Lemouzy, Camps, & Glize, 2011). The AVT can be seen as an adaptation of dichotomous search for dynamic values. The main idea is to change value according to a search direction which is currently requested and directions previously chosen. As long as the value varies in the same direction, the magnitude of the variation is increased otherwise it is reduced. This capability to take into account a changing solution allows the *variable agent* to continuously search for an unknown dynamic target value. This is also a requirement for the system to be able to adapt to changes made during the solving process.

3.4. Output agent

The *output agent* takes charge of an output of a model. *Output agent* and *variable agents* have similar roles, except *output agents* cannot directly change their value. Instead they send a request to the *model agent* they depend on. In this regard, the *output agent* acts as a filter for the *model agent* it depends on, selecting among the different requests the ones it then transmits. The *output agent* is distinct from the *variable agent* in the way that it can be involved in cycles. A cycle is a situation of interdependent models (that is, models which depend of each other to calculate their outputs).

3.5. Constraint agent

The *constraint agent* has the responsibility for handling a constraint of the problem. When receiving a message from one of its inputs, the agent recalculates its constraint and checks its satisfaction. If the constraint is not satisfied, the agent sends *change value* requests to its inputs. It should be noted that, to estimate the input values required to satisfy the constraint, this agent employs the same technique as the *model agent* (i.e. an external optimizer).

3.6. Objective agent

The *objective agent* is in charge of an objective of the problem. This agent sends requests to its inputs aiming to improve its objective, and recalculates the objective when receiving *value changed* messages from its inputs. This agent uses an external optimizer to estimate input values which would improve the objective, as the model and constraint agents.

3.7. Collective solving and non cooperative situations

The complete collective solving relies on several local algorithms and techniques the agents use to choose between conflicting situations and can not be detailed here (a detailed description can be found in Jorquera et al. (2013)). But a good understanding can be achieved by taking into account the following points.

- **Local solving.** During solving, the criteria agents try to improve their local goals and the constraint agents try to keep their constraint satisfied. To this end, they send request messages to the agents controlling their inputs, asking them to change value. The other agents have to propagate these requests toward the variable agents in the most adequate way. An important point is that each agent only has a partial knowledge and local strategy. No agent is in charge of the optimization of the system as a whole, or even of a subset of the other agents. Contrary to classical approaches, the solving of the problem is not directed by a predefined methodology, but by the structure of the problem itself. The emerging global strategy is thus unique and adapted to the problem.
- **Criticality heuristic.** In order to guide the decision of the agents, the multi-agent algorithm introduces several heuristics to be used in order to converge toward a global optimum. The first of these heuristics is called the criticality. The criticality of an agent represents its distance to its local goal, and is transmitted to the neighbors in order to help them select the adequate action. When faced with contradictory requests, an agent will solve the conflict by favoring the most critical neighbor.
- **Non Cooperative Situations.** When represented as graphs, complex optimization problems often exhibit specific topological properties which impede the correct propagation of information between agents. These configurations patterns have been formalized as Non-Cooperative Situations (NCS). A part of the agents decision process consists in identifying such possible NCS, and to apply a specific resolution procedure in order to maintain a correct information propagation in the system. The different NCSs are summarized in Table 1 and the different mechanisms used by the agents to solve them are summarized in Table 2.

Table 1
Non cooperative situations summary.

NCS	Description	Solving mechanisms
Conflicting requests	An agent receives several incompatible requests	Criticality
Cooperative trajectories	An agent receives seemingly incompatible requests, which can each be satisfied	Participation
Cycle solving	Several agents are involved in a diverging cycle	Signature
Hidden dependencies	An agent sends seemingly independent requests to dependent agents	Signature, influence
Asynchronous messages	Agents receive messages in untimely manner	Influence

Table 2
Non cooperative situations solving mechanisms.

Mechanism	Description	Properties
Criticality	An aggregated measure to indicate the state of the agent	Comparable between different agents
Signature	A unique signature composed of the unique id of the sender/origin of the message and a (local) timestamp	Comparable, allows a partial ordering of the messages by sender/origin
Influence	An indicator of the impact value of the receiver on the origin	Comparable by origin
Participation	An indicator of the relative impact of the receiver on the origin relative to the rest of the system	Comparable between different senders

4. AMAS-based SVM training

Let us first consider the problem of finding the optimal (w, b) defining the optimal separating hyperplane (the so-called *training phase*) in linear classification as discussed in Section 2 and the optimal (α, b) for the nonlinear case. We propose simple multi-agent schemes that will achieve training by performing cooperation between margin optimization and loss minimization (training classification error minimization).

4.1. Linear classifier

Consider the following general unconstrained version of Problem (1):

$$\min_{w\in\mathbb{R}^n, b\in\mathbb{R}} m(w) + Cp_S(w, b), \tag{4}$$

where $m(w)$ is a term that takes into account the margin and $p_S(w, b)$ is a penalization term for the constraints

$$y_i(w^\top x_i + b) + \xi_i \geqslant 1 \text{ and } \xi_i \geqslant 0 \quad \forall i \in S,$$

S being the training set.

In Fig. 3, we propose an AMAS counterpart of this problem. One agent will perform margin calculation while another agent will compute the classification error (loss). A third agent combines both values as a weighted sum that is later minimized as a single objective.

As mentioned above, classic choices are:

$$m(w) = \frac{1}{2}\|w\|_2^2$$
$$p_S(w, b) = \sum_{i\in S} \max\left(0, 1 - y_i(w^\top x_i + b)\right).$$

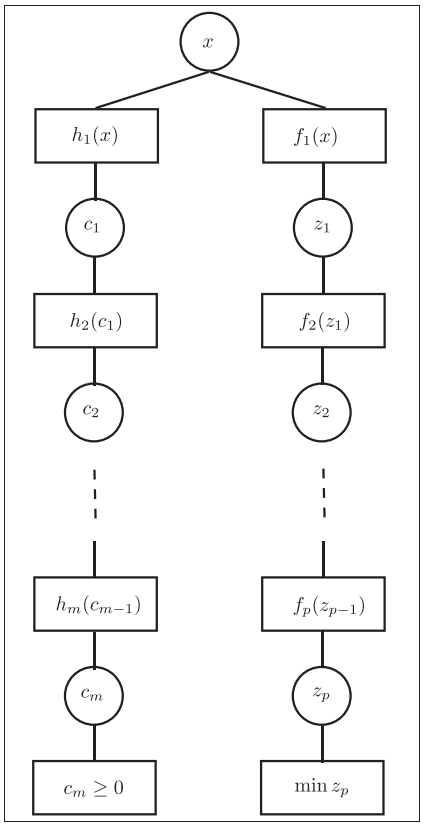

Fig. 2. Multi-agent graph for Problem (3).

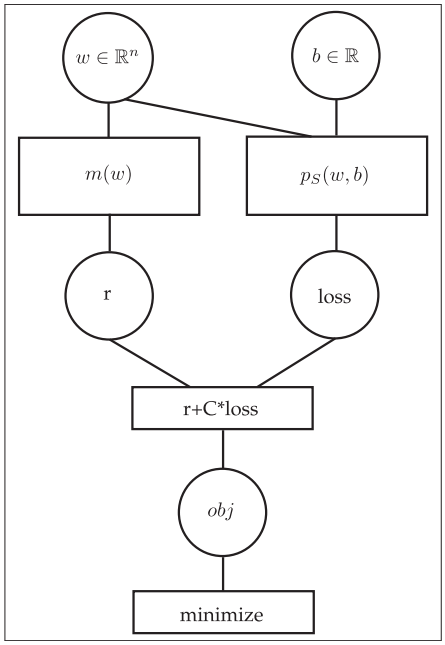

Fig. 3. Multi-agent graph for the linear classifier: $(w, b, obj) = m_L(S, C)$.

In Fig. 3, cooperation will take place at the expense of a model selection procedure to estimate the optimal C parameter for a given problem. Therefore one could also consider a multiobjective variant AMAS system where the tradeoff between margin and loss is naturally achieved through agent cooperation (see Fig. 4 where the variable agent "r + C*loss" is replaced by two variables agents "r" and "loss", both connected to their own objective agent "minimize". Both minimizations are therefore carried out via agent collaboration in a multi-objective setup).

To increase the generalization capabilities of the classifier, the optimal trade-off between margin and loss should be tuned on data that was not used for training. This is why k-fold cross-validation procedures are usually applied to select C with the drawback of being expensive. In Section 5, we propose new AMAS frameworks to overcome such difficulties and find the optimal hyperplane while performing model selection simultaneously.

An important challenge in SVM learning is the increasing sizes of datasets and learning algorithms are required to remain tractable in this context. One way to deal with huge datasets is to perform stochastic optimization in the sense that not all training

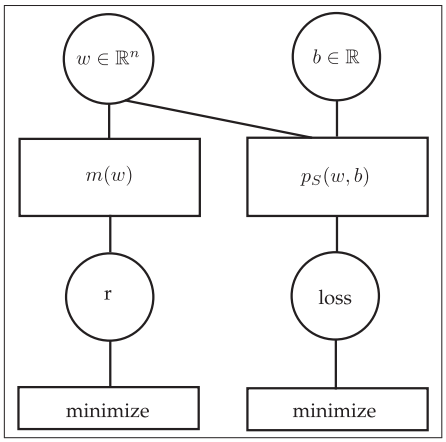

Fig. 4. Multiobjective multi-agent graph for the linear classifier.

datapoints are fed to the system at one time but multiple passes through randomly picked samples are performed instead. Therefore, in the context of large datasets, one could consider replacing the single model $p_S(w,b)$ by several models $p_k(w,b) = \max(0, 1 - y_k(w^\top x_k + b))$, where k is randomly chosen in S each time the AMAS system requires its evaluation.

4.2. Nonlinear classifier

Consider now the general unconstrained version of Problem (2):

$$\min_{\alpha\in\mathbb{R}^L, b\in\mathbb{R}} m_{S,\sigma}(\alpha) + C p_{S,\sigma}(\alpha, b), \tag{5}$$

where the classic choices are

$$m_{S,\sigma}(\alpha) = \frac{1}{2}\alpha^\top \bar{K}(\sigma)\alpha$$
$$p_{S,\sigma}(\alpha, b) = \sum_{i\in S}\left(\max\left(0, 1 - y_i((\bar{K}(\sigma)\alpha)_i + b)\right)\right).$$

As for the linear case, an AMAS counterpart of this problem can be constructed. It has the exact same structure as in Fig. 3 where w is now replaced by $\alpha \in \mathbb{R}^L$ with L the number of data points.

The margin and loss agents are now dependent on the choice of kernel functions we decide to use for the specific learning task. If general multi-purposes kernels such as Gaussian Radial Basis Functions are used, the agents will depend on the kernel parameter σ as described in Section 2.

4.3. Unsupervised learning

Consider now a set of L unlabeled training vectors $\{x_i \in \mathbb{R}^n,\ i = 1,\dots,L\}$ and the problem of assigning one of two classes to each training vector so as to maximize the margin between the resulting classes. This problem can be formulated as:

$$\begin{aligned} \min_{y\in\{-1,1\}^L} \quad & \left(\min_{(w,b,\xi)\in\mathbb{R}^n\times\mathbb{R}\times\mathbb{R}^L} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L}\xi_i\right) \\ \text{s.t.} \quad & \left|\sum_{i=1}^{L} y_i\right| \leqslant L-1 \qquad \begin{array}{ll} \text{s.t.} & y_i(w^\top x_i + b) + \xi_i \geqslant 1 \quad i = 1,\dots,L, \\ & \xi_i \geqslant 0, \qquad\qquad i = 1,\dots,L. \end{array} \end{aligned} \tag{6}$$

Problem (6) expresses that the labels are now unknown and should be selected so as to maximize the margin and minimize the mis-classification penalties. The constraint $\left|\sum_{i=1}^{L} y_i\right| \leqslant L-1$ ensures that the points are separated into two classes and avoids the trivial solution that all points lie in only one class. This problem can also be expressed as:

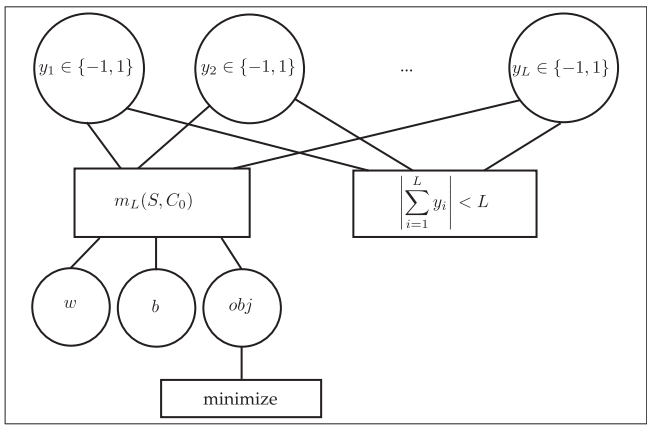

Fig. 5. Multi-agent graph for unsupervised classification.

$$\min_{y\in\{-1,1\}^L} \left|\sum_{i=1}^{L} y_i\right| \leqslant L-1 \left(\min_{(w,b,\xi)\in\mathbb{R}^n\times\mathbb{R}\times\mathbb{R}^L} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L}\max\{0, 1 - y_i(w^\top x_i + b)\}\right) \tag{7}$$

A multi-agent model of the above problem is proposed in Fig. 5. Some variable agents have the tasks to assign each data point to a class while the other model agents are dealing with the search for (w,b) for the given class assignment. The cooperation between agents is attempting to reach an equilibrium where each point is assigned to a class -1 or 1 and where the classes are separated with the largest possible margin. The optimization problems (6) and (7) are usually hard to solve with classical optimization techniques as they combine two levels of optimization with mixed integer variables and nonlinear objectives.

5. Model selection

In the previous section, SVM training with AMAS requires prior knowledge of optimal C or optimal (C,σ) for nonlinear classifiers. As already discussed, acquiring such knowledge via k-fold cross validation is time expensive, more so if datasets are large.

For this reason, in this section, we investigate AMAS system able to self-adjust the hyperparameters while training. Doing so requires that the AMAS system seek the optimal hyperplane on training data while adjusting the hyperparameters on validation data (test data, unused for training). As before, we proceed step by step and investigate first self-adaptive linear classifier and later extend the idea to nonlinear classifiers.

5.1. Penalty parameter selection for linear classifier

Let us denote $(w,b,obj) = m_L(S,C)$ the multi-agent graph displayed in Fig. 3. Given the training set S and the parameter C, it gives the characteristics (w,b) of the classifier and the corresponding value of the objective function. Following the classical setup of a k-fold cross-validation procedure for model selection, we define K training folds $\bar{F}_K$, $j = 1,\dots,K$ and their corresponding testing fold F_k (we have $\bar{F}_k = \{x_1,\dots,x_L\} \setminus F_k$). In the linear case C must be selected so as to minimize the mean of the testing errors on the sets F_k obtained after training on the sets $\bar{F}_k$.

The proposed AMAS for such penalty parameter selection is summarized on the graph in Fig. 6, where $E(F_k, w, b)$ gives the testing error on the set F_k when calculated using the parameters (w,b).

The resulting AMAS is a complex cooperative system that involves training agents providing an optimal hyperplane for each training set and a given C and other agents that are at the same time attempting to minimize the test classification error on the validation sets. Therefore, when a choice for a C value is made, some agents give feedback on its capability for training and others give feedback on its capability for testing. The system will auto-adjust to eventually reach a stable state that is the best compromise between training and testing error. Rather than seeking this compromise sequentially via grid search procedure, we expect that the system will follow an optimization path avoiding ineffective (non-optimal) regions of the feasible space and therefore saving a great amount of computing time. Our expectation will be later confirmed in the numerical examples (see Section 6).

5.2. Penalty and kernel parameters selection for nonlinear classifier

Let us denote $(w,b,obj) = m_{NL}(\bar{F}, C, \sigma)$ the multi-agent graph discussed in Section 4.2. Given the training set $\bar{F}$, the penalty

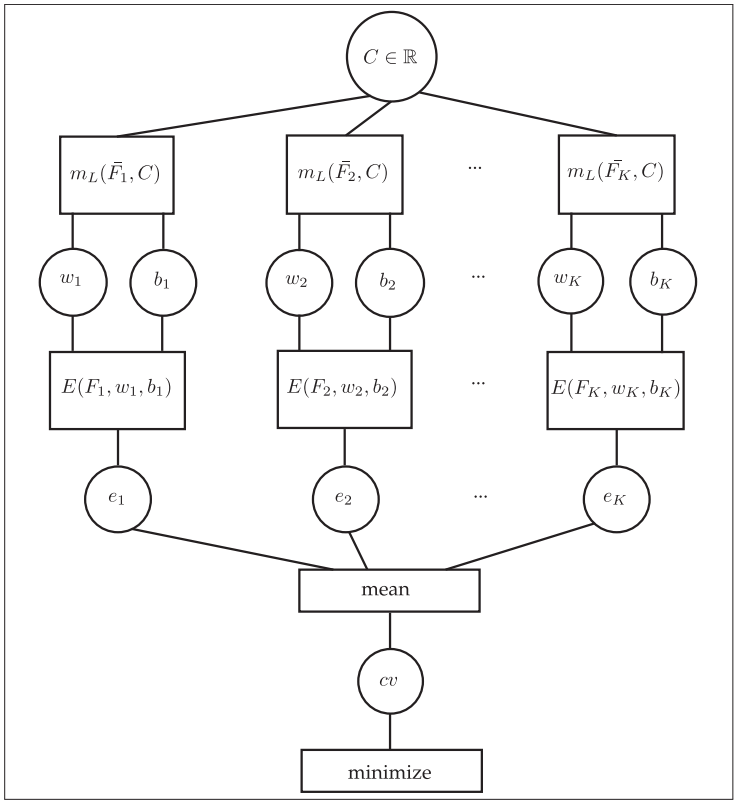

Fig. 6. Multi-agent graph for the penalty parameter selection for linear classifier.

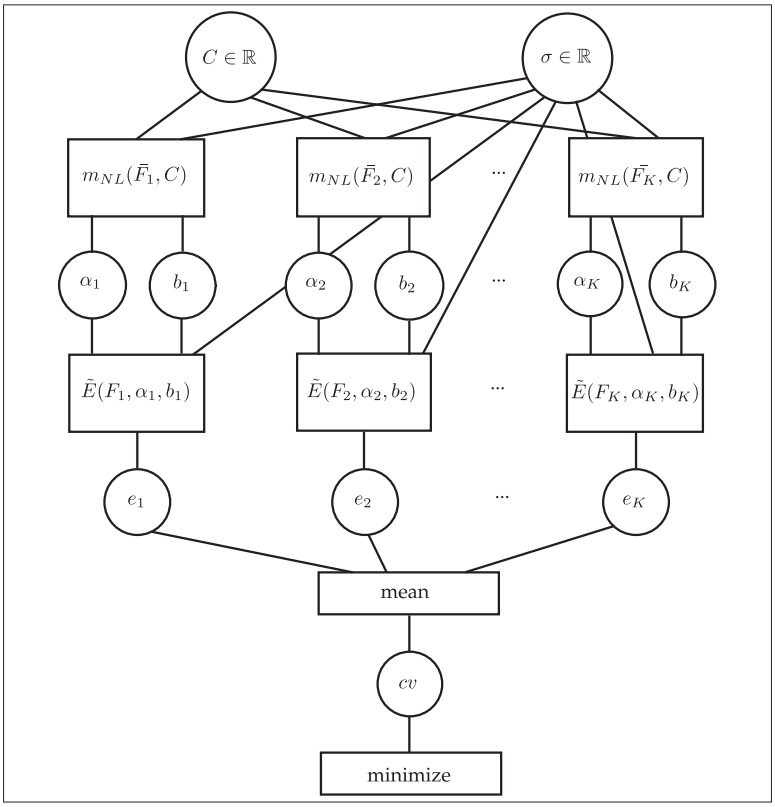

Fig. 7. Multi-agent graph for the penalty and kernel parameters selection for nonlinear classifier.

parameter C and the kernel parameter σ, it gives the characteristics (α, b) of the nonlinear classifier.

Following a similar idea as before and using the same notation for the k training folds $\bar{F}_k,\ k = 1, \ldots, K$ and their corresponding testing folds F_k, we propose an AMAS system for automatic selection of C and σ. The classification error computed with (α, b) is now denoted $\tilde{E}(F_k, \alpha, b)$. The idea is summarized on the graph in Fig. 7. In this system, during the optimization process, each training agent and each validation agent give feedback on the choice on both C and σ. Cooperation between agents will take place dynamically in order to select the best C and the best kernel function to achieve a compromise between training classification error and validation error. Again and even more so in the nonlinear context, the system will seek optimal region of the (C, σ) space avoiding non-optimal regions where computing time would be wasted. In Section 6, we illustrate this behavior.

5.3. Extension to nonlinear multi-class SVM learning

Consider now a set of training vectors $\{x_i \in \mathbb{R}^n,\ i = 1, \ldots, L\}$ and its corresponding set of labels $\{y_i \in \{1, \ldots, M\},\ i = 1, \ldots, L\}$, where L is again the number of training points, n the number of attributes of each training point and M the number of classes.

Among the most common strategies and mathematical formulations for the multi-class problem, we can refer to the following two methods:

- *One-against-all:* M binary SVM models are constructed by taking the class m on one side and the other classes together as the opposite class. The resulting decision function will be of the form $y = \underset{m=1,\ldots,M}{\operatorname{argmax}}\ w_m^\top x + b_m$ where (w_m, b_m) defines the optimal hyperplane computed by the mth binary model.

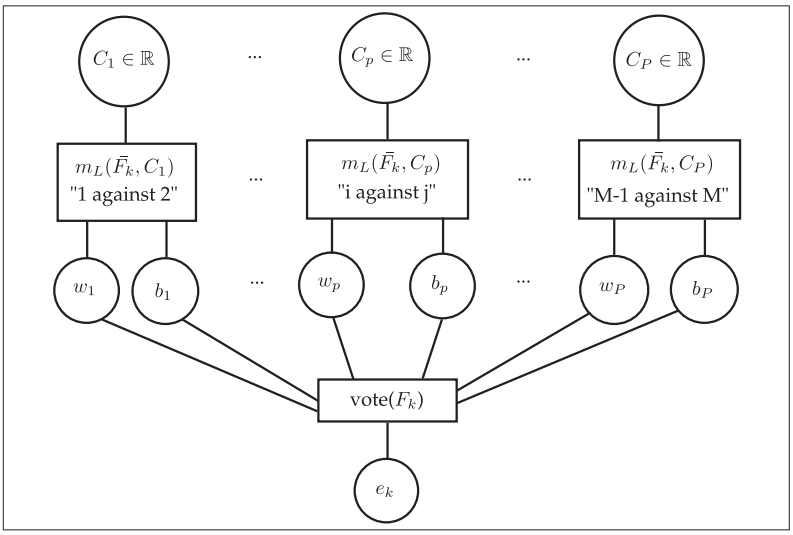

Fig. 8. Multi-agent graph for multiclass training on $\bar{F}_k$ and validation on $F_k : \hat{m}_L(F_k, C)$ where $C = (C_1, C_2, \ldots, C_P)$.

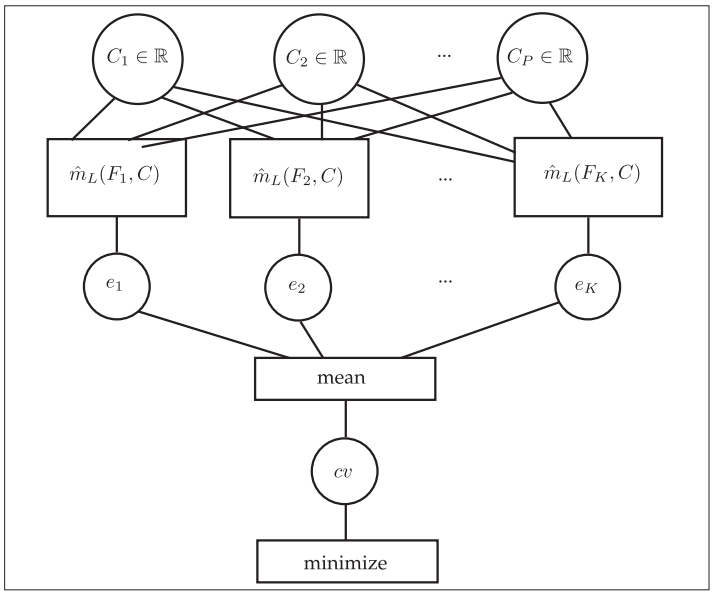

Fig. 9. Multi-agent graph for the penalty parameters selection for multiclass training.

- *One-against-one:* $P = \frac{M(M-1)}{2}$ binary SVM models are constructed by taking each pair of classes in $\{1 \ldots, M\}$. The resulting decision function is obtained by a majority vote meaning that a point gets one vote for class m if the p-th pair of classes $(p = 1, \ldots, P)$ assigns x to class m. The class with the highest total vote numbers will finally be assigned to x.

In these two methods, one can clearly see that model selection is a critical issue if the number of classes M is high. In the *One-against-all* technique, M training procedures have to be performed, each one of them involving the selection of $C_m,\ m = 1, \ldots, M$. In the *One-against-one* technique, P training procedures are required and therefore P model selection are to be executed. In both cases, classic k-fold cross-validation procedures would be extremely expensive. The number of binary classification tasks is greater for the *One-against-one* method, however, each task is easier to execute than for the *One-against-all* method. This setup (more agents but less and simpler agent decisions) is usually preferred in AMAS as it provides a greater breakdown of complexity. We first propose an AMAS model to train multi-class data in Fig. 8. The P agents will execute each a binary classification on the fold $\bar{F}_k$ and return the corresponding hyperplanes. Another model agent will perform the majority vote to calculate the test error for a final variable agent e_k. In Fig. 9, we show how this agent-based training procedure can be embedded in a higher level agent graph that will perform model selection on k folds and return the cross-validation error via the *cv* variable agent.

6. Some numerical examples

6.1. Experimental setup

We provide several examples to illustrate the behavior of the AMAS when performing SVM training as described in Section 4 as well as model selection as explained in Section 5. We also show some simple applications to multi-class training and unsupervised learning. All agents models used in the examples are implemented using the ID4CS multi-agent system environment (ID4CS, 2013).

6.2. Example 1: linear classifier

This first simple example illustrates the calculation of the separation hyperplane between two classes using agents. We randomly and uniformly generate samples ($x_i \in \mathbb{R}^2$ for $i = 1, \ldots, 126$) in the box $[-1, 1]^2$ and define their corresponding labels $y_i \in \{-1, +1\}$ as:

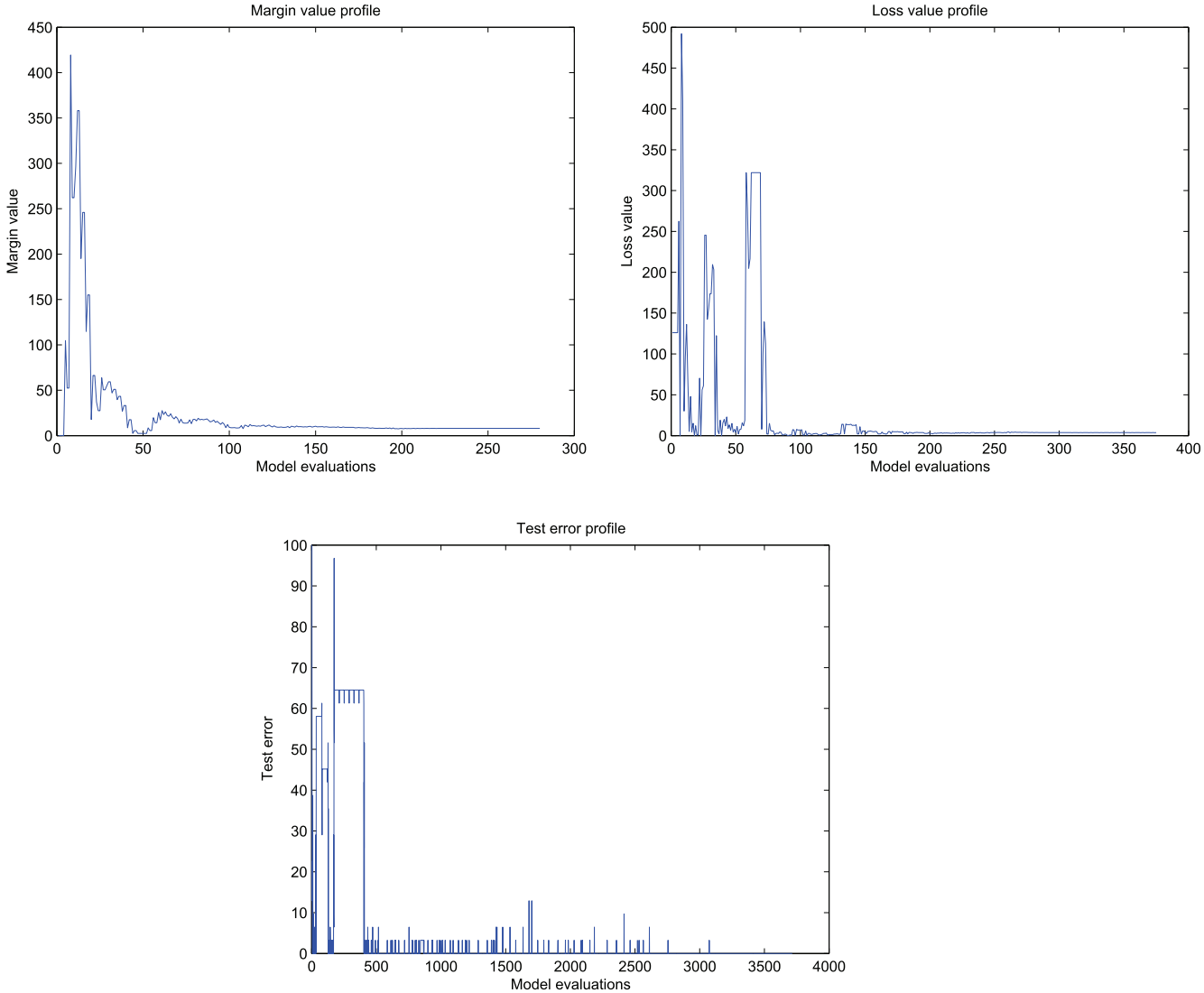

Fig. 10. Profiles for linear classification.

$$y_i = \begin{cases} -1 & \text{if } x_1 + x_2 \leqslant 0 \\ +1 & \text{otherwise.} \end{cases} \tag{8}$$

We also randomly generate $N\,(N = 31)$ test data points uniformly in the box $[-1, 1]^2$ and the desired test labels are defined as in (8).

The AMAS has the task to learn the data on one fold and return the optimal hyperplane defined by (w, b). There is no model selection to perform. The C parameter is here taken as $C = 1$. We evaluate the performance of the training by checking the classification error on the test data. We record the test error profile as well as margin and loss profiles during the learning process. The results are given in Fig. 10. In Fig. 11, we plot the hyperplane (w, b) found by the AMAS and check whether both training and testing data are well separated. The AMAS was able to achieve zero training and testing error, with complete separation of the two classes of points for the training set but also for the testing set.

6.3. Example 2: penalty and kernel parameters selection for nonlinear classifier

As a second example, we now simulate the nonlinear AMAS model for parameters selection (see Fig. 7). We use the same synthetic example as for Example 1 but perturb randomly 10% of the points by assigning them to the opposite class to ensure non linear separability. The dataset is randomly divided into 5 folds. The objective is to achieve agent cooperation in order to find the optimal (C, σ)-pair of hyperparameters over the 5 folds.

We report the results on Fig. 12. Starting from 100%, the cross-validation error rapidly drops down to a small value after approximately 25 model agent evaluations. The left figure in Fig. 12 shows the plane (C, σ). It is interesting to see which values of (C, σ) were evaluated by the agents during the cooperation process. If one had to run a classic cross-validation procedure combined with a grid search, the complete space would have to be explored. Here, the AMAS cooperation process is only going over specific regions of the space where better compromises between training error and generalization error can be found. Specifically, a first region around the starting values of $(C, \sigma) = (1, 0.01)$ was explored and rapidly the system "jumped" into another region (central region of the figure) where a better optimum was found. Attempts were made to seek other regions but cooperation was clearly bringing back the values into the central region. Rather than a blind grid search, the agents are able to focus only on optimal regions of the space, leading to savings in computing time.

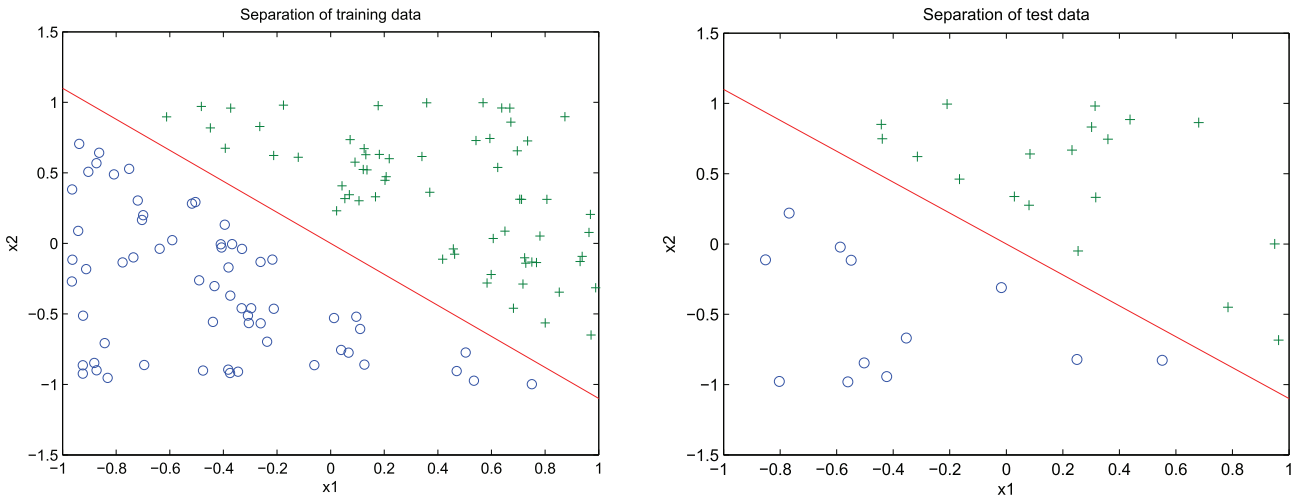

Fig. 11. Separation of data with linear classifier.

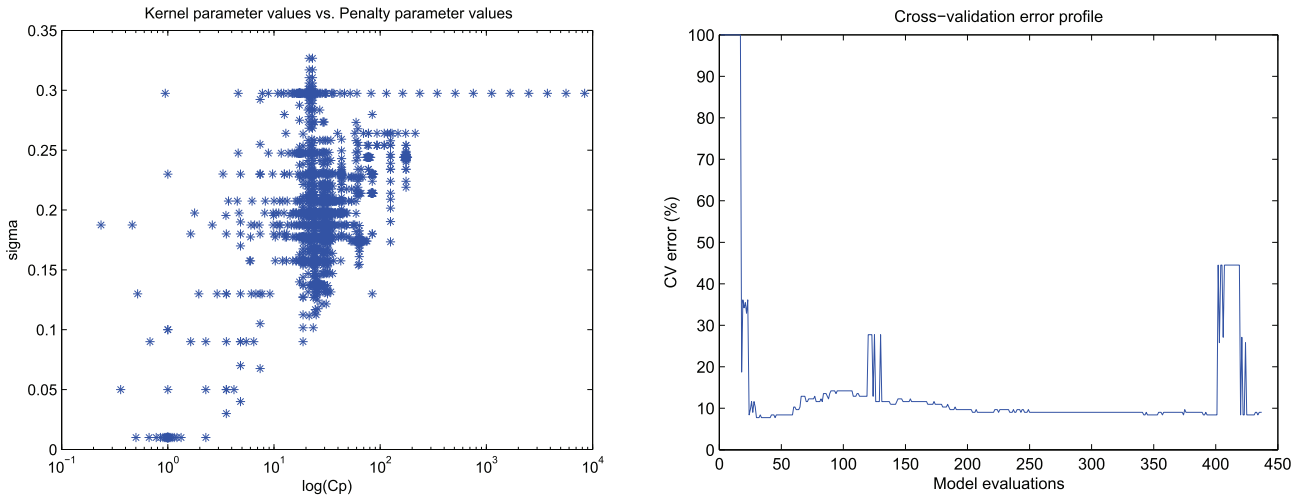

Fig. 12. Separation of data with nonlinear classifier.

6.4. Example 3: penalty parameters selection for multi-class learning

In Example 3, we focus on multi-class learning. We randomly and uniformly generate samples ($x_i \in \mathbb{R}^2$ for $i = 1, \ldots, 100$) in the box $[-1, 1]^2$ and define their corresponding labels $y_i \in \{-1, +1\}$ as:

$$y_i = \begin{cases} 1 & \text{if} \quad x_1 \geqslant 0 \quad \text{and} \quad x_1 + x_2 \geqslant 0, \\ 2 & \text{if} \quad x_1 - x_2 \geqslant 0 \quad \text{and} \quad x_1 + x_2 < 0, \\ 3 & \text{if} \quad x_1 - x_2 < 0 \quad \text{and} \quad x_1 < 0. \end{cases} \tag{9}$$

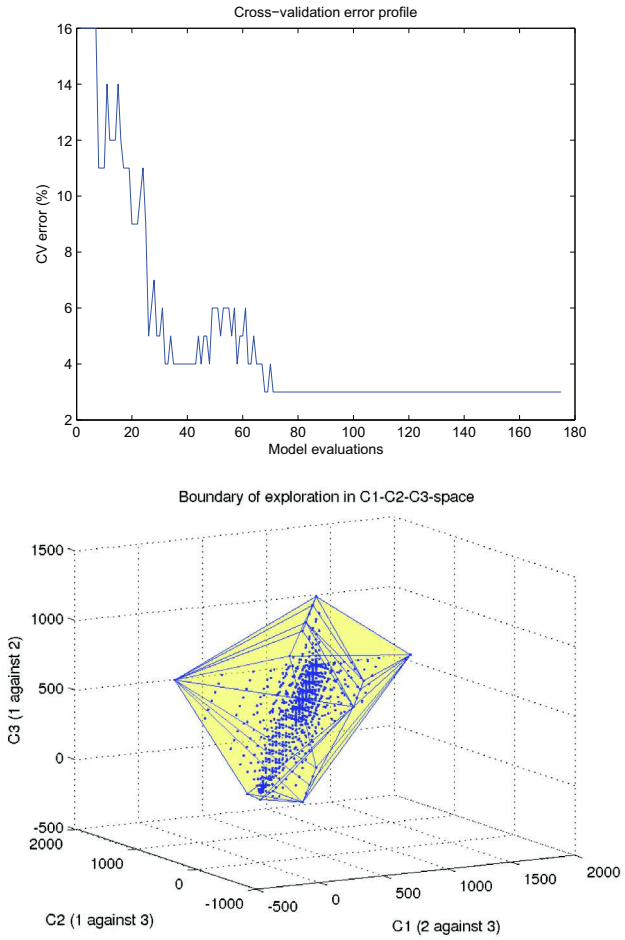

Fig. 13. Penalty parameters selection for multi-class learning.

Again, we divide the data into 5 folds and model the AMAS presented in Fig. 9 to automatically select the P hyperparameters $C_1, C_2, \ldots, C_P$ corresponding to the P *one-against-one* binary agent classification tasks. In this example $P = 3$. We present the results in Figs. 13 and 14. On the top graph in Fig. 13, one can see the profile of the cross-validation error over the 5 folds. The learning process clearly takes place to eventually reach a CV-error of about 3%. On the bottom graph in Fig. 13, we plot in the (C_1, C_2, C_3)-space, the values of the hyperparameters that were evaluated by the variable agents and draw their convex-hull to emphasize the complete volume that was explored by the AMAS: it only focuses on a small part $(C_1, C_2, C_3) \in \{1, \ldots, 1500\}^3$ of the complete (C_1, C_2, C_3)-cube which was set to $\{1, \ldots, 10^5\}^3$. Again, when compared to a classical cross-validation procedure (combined with a grid search) that would require exploring the complete cube values, the AMAS system is able to limit the search space to a smaller sub-volume. Less computational effort is spent as non optimal regions are not explored and better accuracies should be achieved since (C_1, C_2, C_3) can take any continuous values in the cube instead of discrete grid values for the classic grid search. To check whether the (C_1, C_2, C_3)-values found by the AMAS are appropriate for learning the 3 given classes, we show in Fig. 14 the resulting separation for the training data as well as for some testing set. The achieved separation is correct.

6.5. Example 4: unsupervised classification

In this last example, we propose to illustrate the AMAS for unsupervised classification (see Fig. 5). For this purpose, we draw randomly and uniformly 10 samples $x_i \in \mathbb{R}^2$ in the box $[-1, 1]^2$. The objective is to assign each sample into one of the two classes -1 and 1 such that the separation margin between the two classes is maximum. The resulting class assignment and separation achieved by the AMAS model is shown in Fig. 15. In this figure, 6 subplots are given. Each one of them represents a distinct situation during the learning process. Subplot (1) is the starting situation with no separation and random class assignment. Subplot (6) gives the final situation after learning. The AMAS has found that the

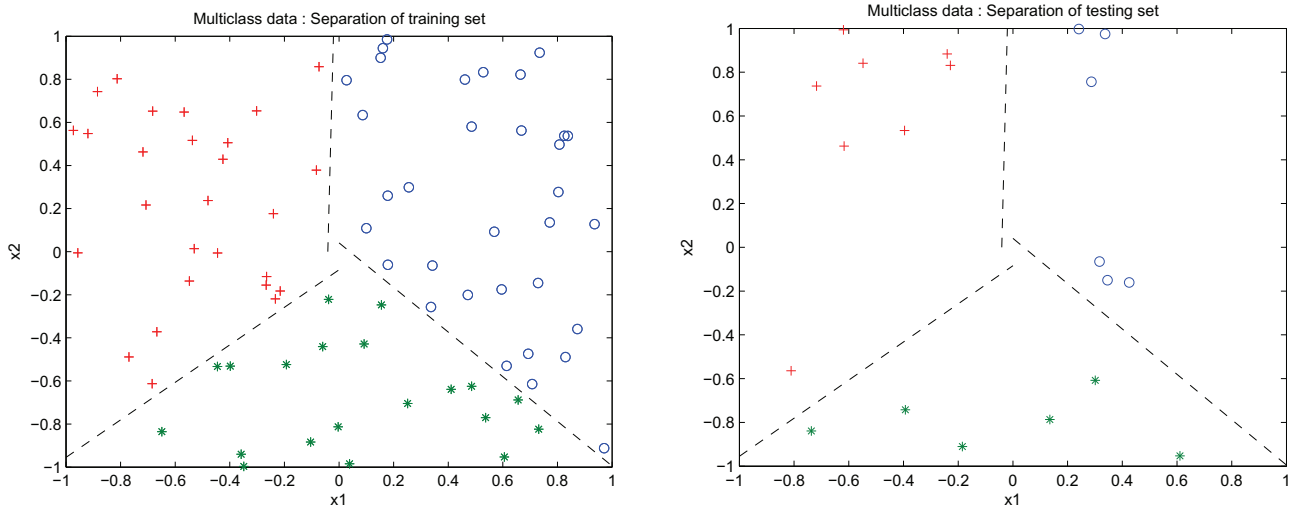

Fig. 14. Separation of multi-class data.

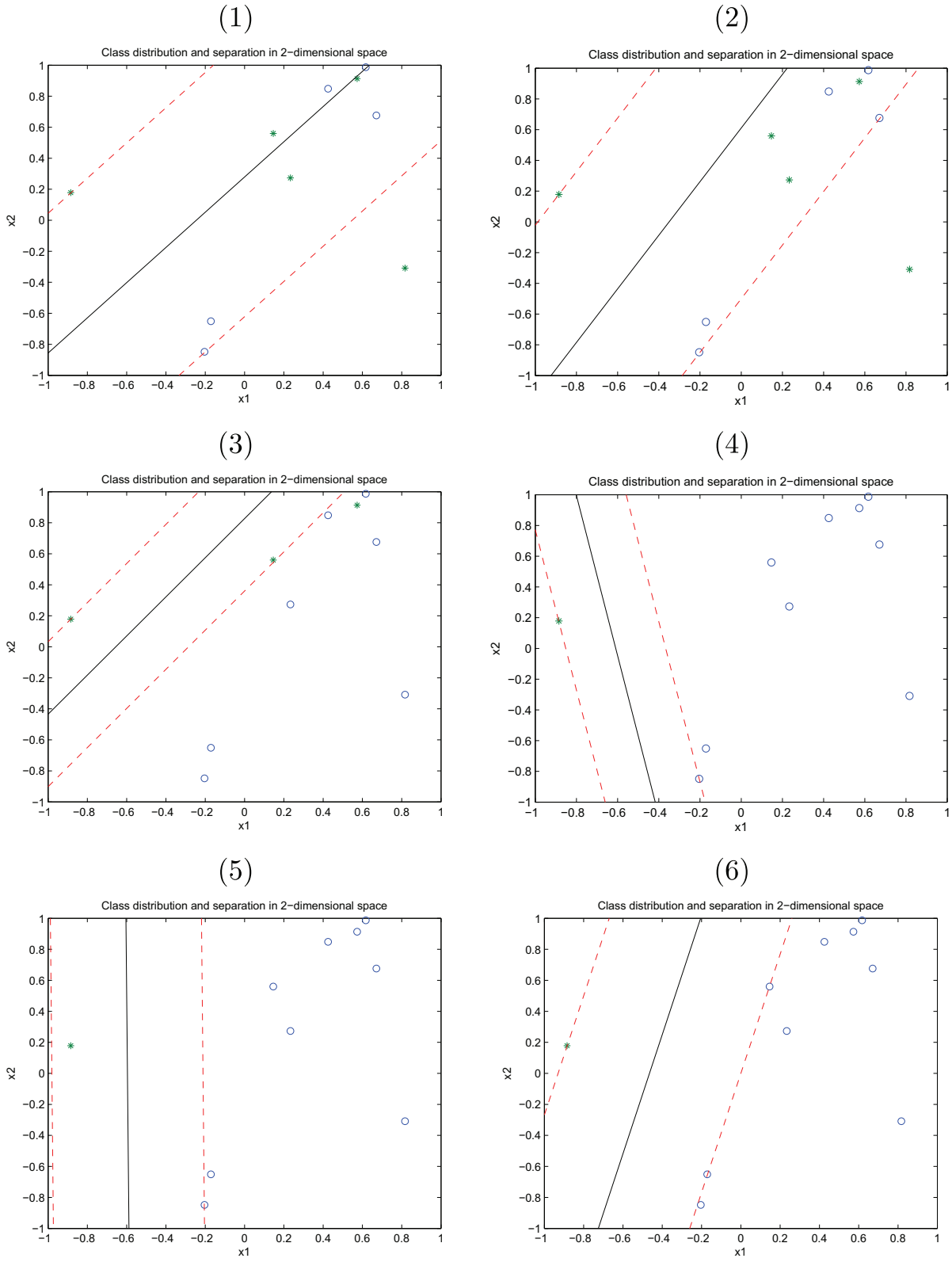

Fig. 15. Class assignment and class separation during unsupervised learning.

maximum margin separation into two classes is achieved when only one data point belongs to one class (the datapoint around the coordinates $(-0.9, 0.2)$) and the other points in the other class. The subplots (2)–(5) show the intermediate situations and the rotation and displacement of the hyperplane during the process of agent collaboration.

7. Discussions and conclusions

We have discussed and proposed a new class of learning methods making use of multi-agent optimization. Focusing on Support Vector Machine concepts, we have proposed and illustrated how multi-agents systems can perform linear or nonlinear

classification, model selection as well as multi-class or unsupervised learning.

Using AMAS, our objective is not only to distribute computational effort. We aim at distributing the complexity of the model selection issue across agents. As explained in the introduction, multi-level optimization formulations that are alternatives for such problems are difficult to solve and actually do not address the kernel parameter selection (the task is too complex for classical optimization methods). AMAS provide a natural way to implement multi-level optimization. This explains why we are not providing comparison with other efficient techniques such as parallel SMO (Cao et al., 2006) or other large scale stochastic subgradient methods (Shalev-Shwartz, Singer, Srebro, & Cotter, 2011). These methods do not provide numerical methodologies to perform automatic parameter selection. They require cross-validation or other parameter estimation techniques (see for example Guyon (2009), Guyon, Saffari, Dror, & Cawley (2010) for reviews of practical parameter selection techniques).

The AMAS cooperation strategy is building knowledge of the problem during the solving process. Complex problems are distributed across agents and each agent gains local knowledge of the problem with respect to the task it is given. Therefore, the global strategy of the AMAS could be compared as a "complex" descent technique towards the optimal solution. One could legitimately raise the question of comparison with other approaches based on heuristics such as Genetic Algorithms (GA) or Swarm intelligence. In fact, these strategies would explore solutions without building any knowledge of the problem and could be seen as intelligent random search rather than "descent" technique. Such heuristics require themselves problem specific parameter tuning which makes them poor candidate for parameter selection issues. In the case of SVM, C and σ selection would be replaced by other problem specific parameter selection (ex: population size in the case of GA) of the same order of complexity. For these reasons, we believe that AMAS based models are better suited for model selection issues.

In machine learning, complexity arises with the dimension of problems, for example, with large datasets or with high number of classes. Learning problems usually possess a decomposable structure and AMAS gives a natural way to distribute the learning task among agents in order to breakdown the complexity. There are other important issues of learning that could be addressed in such a way. Multiple kernel learning (MKL) is for example one of the challenging issue that AMAS could nicely model avoiding the inherent complexity of classical optimization models (Gönen & Alpaydın, 2011).

In MKL, one makes use of convex combinations of kernels

$$k(x_i, x_j) = \sum_{q=1}^{Q} \beta_q k_q(x_i, x_j)$$

with $\sum_{q=1}^{Q} \beta_q = 1$ and $\beta_q \geqslant 0$ for all $q = 1, \ldots, Q$ to find the maximum margin hyperplane. Therefore, the problem is to find the best convex combination coefficients β_q, in order to compute the separating hyperplane in the Reproducing Kernel Hilbert Space (i.e. the mapped space as mentioned above) associated to the resulting composite kernel function. This problem is a multi-level optimization that is usually difficult to solve. By setting variable agents to decide on the values of the coefficients $\beta_q \geqslant 0$ and adding specific constraint agents, the agent graph for training can naturally be defined. It is also simple to include heterogeneous kernel functions (ex: a mixture of gaussian kernels, polynomials or other specific-purposed kernels) and let the agents decide on the best combination as well as the best kernel parameters (flatness for gaussian kernels, degree for polynomials,...). Such setup could be efficient on very complex learning tasks as it is known that MKL can outperform single kernel techniques.

Clearly, AMAS provides a natural formalism to express learning problems. Further research and implementations should confirm that this framework could be an alternative solution for building learning systems that require high level of adaptability especially for complex learning tasks where distribution of problem solving among agents is necessary. Furthermore, high degree of parallelization is possible since most of the time all elementary agent decisions are independent and asynchronized. AMAS are therefore suited to high dimensional problems arising in many applications such as in genomic data analysis in biology or in webdata analysis from the internet environment. Even though we did not detail the idea of online learning with data being fed to the system one at a time, it is also important to mention that all concepts that were discussed can be extended to such situations as long as the AMAS system implementation allows online generation and integration of new agents.

References

Balakrishna, P., Raman, S., Santosa, B., & Trafalis, T. (2008). Support vector regression for determining the minimum zone sphericity. *International Journal of Advanced Manufacturing Technology, 35*, 916–923.

Bottou, L. (1997). Online learning and stochastic approximations. In *On-line learning in neural networks. Papers from the workshop, Cambridge, GB, November 17–21, 1997* (pp. 9–42). Cambridge: Cambridge University Press.

Bousquet, O., & Bottou, L. (2007). The tradeoffs of large scale learning. In J. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.), *Advances in neural information processing systems 20* (pp. 161–168). Cambridge, MA: MIT Press. http://books.nips.cc/papers/files/nips20/NIPS2007_0726.pdf.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press [<http://dx.doi.org/10.1017/CBO9780511804441>. doi: 10.1017/CBO9780511804441].

Cao, L. J., Keerthi, S. S., Ong, C.-J., Zhang, J. Q., Periyathamby, U., Fu, X. J., et al. (2006). Parallel sequential minimal optimization for the training of support vector machines. *Transactions on Neural Networks, 17*, 1039–1049 [<http://dx.doi.org/10.1109/TNN.2006.875989>. doi:10.1109/TNN.2006.875989].

Chan, P., & Stolfo, S. (1998). Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *Proceedings of the fourth international conference on knowledge discovery and data mining* (pp. 164–168). Springer.

Couellan, N., & Wang, W. (2014). Bi-level stochastic gradient for large scale support vector machine. *Neurocomputing*. <http://www.sciencedirect.com/science/article/pii/S0925231214015586>. doi: http://dx.doi.org/10.1016/j.neucom.2014.11.025.

Cristianini, N., & Shawe-Taylor, J. (2001). An introduction to support vector machines and other kernel-based learning methods. *Repr. (repr. ed.)*. Cambridge: Cambridge University Press.

Decoste, D., & Schölkopf, B. (2002). Training invariant support vector machines. *Machine Learning, 46*, 161–190. http://dx.doi.org/10.1023/A:1012454411458.

Di Marzo Serugendo, G., Gleizes, M.-P., & Karageorgos, A. (Eds.). (2011). *Self-organizing software. From natural to artificial adaptation*. http://dx.doi.org/10.1007/978-3-642-17348-6.

Du, P., Peng, J., & Terlaky, T. (2009). Self-adaptive support vector machines: Modelling and experiments. *Computational Management Science, 6*, 41–51 [<http://dx.doi.org/10.1007/s10287-008-0071-6>. doi:10.1007/s10287-008-0071-6].

Fung, G., & Mangasarian, O. L. (2006). Breast tumor susceptibility to chemotherapy via support vector machines. *Computational Management Science, 3*, 103–112 [<http://dx.doi.org/10.1007/s10287-005-0002-8>. doi:10.1007/s10287-005-0002-8].

Georgé, J.-P., Edmonds, B., & Glize, P. (2004). Making self-organising adaptive multiagent systems work. In *Methodologies and software engineering for agent systems*. US: Springer.

Gilbert, R., Raman, S., Trafalis, T., Obeidat, S., & Aguirre-Cruz, J. (2009). Mathematical foundations for form inspection and adaptive sampling. *Journal of Manufacturing Science and Engineering*, 131.

Gönen, M., & Alpaydın, E. (2011). Multiple kernel learning algorithms. *Journal of Machine Learning Research, 12*, 2211–2268 [<http://dx.doi.org/10.1016/j.patcog.2012.09.002>. doi:10.1016/j.patcog.2012.09.002].

Guyon, I. (2009). *A practical guide to model selection. Proceedings of the machine learning summer school springer text in statistics*. Springer.

Guyon, I., Saffari, A., Dror, G., & Cawley, G. (2010). Model selection: Beyond the bayesian/frequentist divide. *Journal of Machine Learning Research, 11*, 61–87. <http://www.jmlr.org/papers/volume11/guyon10a/guyon10a.pdf>.

Hastie, T., Rosset, S., Tibshirani, R., & Zhu, J. (2003/04). The entire regularization path for the support vector machine. *Journal of Machine Learning Research, 5*, 1391–1415.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning. Springer series in statistics* (2nd ed.,). New York: Springer [<http://dx.doi.org/10.1007/978-0-387-84858-7>. doi:10.1007/978-0-387-84858-7 data mining, inference, and prediction].

ID4CS (2013). Integrative design for complex systems. webpage: <http://www.irit.fr/id4cs>.

Jorquera, T. (2013). An adaptive multi-agent system for self-organizing continuous optimization. Thèse de doctorat Université de Toulouse, Toulouse, France. <https://websecu.irit.fr/ajaxplorer/data/public/be000eca75d16e31ad56e126fc25344b.php?lang=fr>.

Jorquera, T., Georgé, J.- P., Gleizes, M.- P., Couellan, N., Noel, V., & Régis, C. (2013). A natural formalism and a multi-agent algorithm for integrative multidisciplinary design optimization. In *International workshop on optimisation in multi-agent systems at the twelfth international conference on autonomous agents and multiagent systems (AAMAS)*. Saint Paul, Minnesota, USA.

Kunapuli, G., Bennett, K. P., Hu, J., & Pang, J.-S. (2008). Bilevel model selection for support vector machines. In *Data mining and mathematical programming. CRM proc. lecture notes* (Vol. 45, pp. 129–158). Providence, RI: Amer. Math. Soc..

Lee, Y.-J., Mangasarian, O. L., & Wolberg, W. H. (1999). Breast cancer survival and chemotherapy: A support vector machine analysis. In *Discrete mathematical problems with medical applications (New Brunswick, NJ, 1999). DIMACS ser. discrete math. theoret. comput. sci.* (Vol. 55, pp. 1–10). Providence, RI: Amer. Math. Soc..

Lemouzy, S., Camps, V., & Glize, P. (2011). Principles and properties of a MAS learning algorithm: A comparison with standard learning algorithms applied to implicit feedback assessment. In *2011 IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology (WI-IAT)*, (pp. 228–235). Vol. 2. doi:10.1109/WI-IAT.2011.190.

Scholkopf, B., & Smola, A. J. (2001). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. Cambridge, MA, USA: MIT Press.

Shalev-Shwartz, S., Singer, Y., Srebro, N., & Cotter, A. (2011). Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical Programming, 127*, 3–30 [<http://dx.doi.org/10.1007/s10107-010-0420-4>. doi:10.1007/s10107-010-0420-4].

Trafalis, T., Adrianto, I., & Richman, M. (2007). Active learning with support vector machines for tornado prediction. *Computational science–ICCS* 2007 (pp. 1130–1137).

Vapnik, V. N. (1998). *Statistical learning theory. Adaptive and learning systems for signal processing, communications, and control*. John Wiley & Sons.

Weiss, G. (1999). *Multiagent systems. A modern approach to distributed artificial systems*. MIT Press.