Incremental Learning Algorithms for Classification and Regression: local strategies

Florence d'Alché-Buc* and Liva Ralaivola*

*LIP6, UMR CNRS 7606, Université P. et M. Curie, F-75252 Paris cedex 05, France

Abstract. We present a new local strategy to solve incremental learning tasks. Applied to Support Vector Machines based on local kernel, it allows to avoid re-learning of all the parameters by selecting a working subset where the incremental learning is performed. Automatic selection procedure is based on the estimation of generalization error by using theoretical bounds that involve the margin notion. Experimental simulation on three typical datasets of machine learning give promising results.

INTRODUCTION

Robotics, system monitoring and user-modeling in real time require adaptive systems whose parameters change with their environments. We consider an incremental learning task for which the learning system has continuously to adjust its parameters and its structure to new patterns. This can be seen as a first step towards the acquisition of drifting concepts that evolve with time. While recent advances in machine learning have mainly concerned the statistical framework ([19],[11]) and batch learning tasks for which the whole training set is available at the beginning of the learning process, incremental learning has given rise to few developments. Like batch learning, incremental learning aims at minimizing the generalization error but with a growing training set ([13]). It can be summarized by a two-step procedure that is applied at each arrival of new observations: first, modify the parameters of the existing learning system, and second, if the new data are not correctly predicted, add new parameters and learn the new system.

In this paper, we introduce a new strategy, the *local strategy*, based on locality considerations, that avoids the re-learning process of all the parameters. Only the most responsible parameters of the system are considered for the re-learning process. In order to select the responsible subset, we propose to use theoretical bounds that allow to give an estimate of generalization error.

Some specific neural networks architectures seem especially well adapted to the *local strategy* such as hybrid trees whose leaves are learning systems and intermediate nodes segment the input space or as kernel-based machines like Radial Basis Function Network or Support Vector Machines (SVM). Here we derive it for the SVMs ([7, 19]), one of the most accurate learning system developed so far. Support Vector Machines present several advantages: first, they act as a committee machine combining the votes of experts based on training data (the so-called support vectors) that lie on the boundary of the classes (for classification tasks). Second, they use the kernel trick that allows to solve a non linear separation problem as a linear one in a high dimensional *feature space* without directly working in it. Third, they minimize a margin criterion that forces the produced decision surface to be as far as possible to the data. Fourth, powerful theoretical bounds involving margins are available for SVM. However, whereas SVM have been largely developed for different batch learning tasks, few works tackle the issue of incremental learning of SVM. This can be explained by the nature of the optimization problem involved by SVM learning that must be solved globally by quadratic programming and which thus does not seem very well suited to incremental learning. Although there exist some very recent works that propose ways to update SVM each time new data are available [5, 14, 18], they generally imply to re-learn the whole machine.

The work presented here exploits the fact that parameters of SVM correspond to the examples themselves. It is thus possible to focus learning on a working set restricted to the neighbourhood of the new data and thus update only the weights of concerned training data. The proposed algorithm for SVM, called ALISVM, improves first implementations of LISVM ([20]) drawn with a fixed number of neighbors and early stopping.

The paper is structured as follows. The machine learning framework is first presented. Then, the local strategy is developed. Then, after briefly describing SVM, we present the local incremental algorithm devoted to this machine and discuss the model selection method to determine the size of the neighbourhood to be used at each step. Numerical simulations on IDA benchmark datasets [17] are presented and analyzed. Extensions to regression and to other machines (hybrid decision trees) are also proposed. Finally, we conclude and give perspectives.

STATISTICAL MACHINE LEARNING AND INCREMENTAL LEARNING

In this section, the framework of statistical machine learning restricted to the supervised classification task with two classes is recalled. The extension to the regression case is straightforward ([19]).

Let p be a fixed but unknown distribution over $(\mathbf{x}, y)$ pairs, $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y} = \{-1, 1\}$. Let $\mathcal{S} = \{\mathbf{z}_i = (\mathbf{x}_i, y_i),\ i = 1...\ell\}$ be a i.i.d. sample drawn from p. The goal of *supervised learning* is to define the parameters of a classification function f defined as $f(\mathbf{x}) = \mathrm{sign}(h(\mathbf{x}))$ with $h \in \mathcal{H}$, a family of functions from $\mathcal{X}$ to $\mathcal{R}$ that tends to minimize the generalization error defined by

$$\varepsilon_G = \int_{(\mathbf{X},\mathbf{Y})} 1_{f(\mathbf{x}) \neq y} p(\mathbf{x}, y) d\mathbf{x} dy.$$

As this quantity cannot be calculated since $p(\mathbf{x}, y)$ i unknown, machine learning methods use to minimize

$$\widehat{\varepsilon}_G = \frac{1}{\ell} \sum_{i=1}^{\ell} f(\mathbf{x}_i) y_i$$

known as the *empirical error* plus a term controlling the Vapnik-Chervonenkis complexity of the family $\mathcal{H}$. The theoretical bounds on generalization error provide such criteria.

In incremental learning, the sample size increases throughout the training phase. We suppose that data items arrive one after the other and that the whole dataset is not fully available at the beginning of the learning process. This means that the system keeps on learning and adapts itself to new data. For sake of simplicity, we suppose that at each time t, a new training pattern $\mathbf{z}_t = (\mathbf{x}_t, y_t)$ is observed. The assumption that this pattern is drawn randomly from $p(\mathbf{x}, y)$ remains. $\mathcal{S}_t$ the current observed sample at time t is thus $\mathcal{S}_t = \mathcal{S}_{t-1} \cup \{\mathbf{z}_t\}$. We note $\ell_t = t$ the number of examples observed until time t. The *incremental supervised learning* consists in updating the previous hypothesis h_{t-1} into $h_t \in \mathcal{H}$ using $\mathcal{S}_t$ in order to minimize generalization error. Let us call Θ_{t-1} the parameters set of h_{t-1} and Θ_t the parmeters set of h_t. The updating process usually involves two main operations, *modify* the existing parameters of Θ_{t-1} and *add* new parameters if needed and learn the new set Θ_t.

Incremental learning should be differentiated from *online learning* and *constructive learning* [21] which convey other meanings in the machine learning community: in opposition to batch learning algorithms, online learning algorithms try to learn and update the current classifier using only the new available data, that is, without using any past observed data; constructive learning aims at automatically designing a neural network of appropiate size, given the full training dataset. However, in both of these domains, automatic growing of the system (network) and adding new units have been studied. Especially, the strategies referred by Kwok et al. in their survey about constructive algorithms ([21]) consists in re-learning the whole parameters set or only the added unit when adding a new unit.

A natural extension to the incremental learning of a static concept is to consider that the target concept or function changes with times. In this case, $p(\mathbf{x}, y)$ is time-dependent and can be written $p_t(\mathbf{x}, y)$. A third operation can be added to the previous ones which consist in *forgetting* old examples and put emphasis on recent ones. This task identified as incremental learning of drifting concepts is considered in the discussion while most of the paper is devoted to incremental learning of static targets.

THE LOCAL STRATEGY

Principle

One method for incremental learning consists in a two-step procedure: first, apply a complete re-learning process of all the parameters using the training data observed so far when a new example fails to be correctly classified and second, evaluate if the prediction for the new example is correct and if not, add some new parameters to be learnt with the whole network (for instance, a neuron is created, if the classifier is a neural network, [13]). We refer to this method as "re-batch learning" since it makes a batch learning pass each time a new data item arrives. While this algorithm ensures that the optimization process is correclty driven, the computational cost is prohibitive since a complete re-learning of all the parameters is needed.

Therefore, we propose to speed up step 1 by deriving a local approach that selects at each arrival of new data, the minimal subset of parameters Θ_t that need to be modified. It seems rather natural to focus on parameters that are the most responsible for the classifier output given the input pattern and do not modify the other parameters. This means that the decision frontier in the classification case will only be modified in a neighborhood of the new data. For regression, the outputs of the predicted function will be changed only for inputs localized in the neighborhood of the data.

In both cases, the amount of responsibility of a parameter θ of h can be captured by the gradient $\frac{\partial h(\mathbf{x})}{\partial \theta}$. However, the cardinal $\widehat{n}_t$ of the working parameters subset, Θ_t must be determined according to the minimization of some instantaneous objective function $J_{\mathcal{S}_t}(n)$ that reflects the goal of the task. First of all, we want to minimize the generalization error $\varepsilon_G(t)$ as in batch learning. For this purpose, we can use theoretical bounds for the functions family to which h belongs and consider an estimate $\widehat{\varepsilon}_G(t)$ of $\varepsilon_G(t)$. But we also would like to increase the efficiency of the learning algorithm and therefore, reduce the time-complexity of the method. The objective function should thus convey the care that must be given to either the generalization error and the time-complexity. In order to illustrate this scheme, we describe in the following a detailed implementation of the approach together with the adequate objective function for the Support Vector Machine.

SUPPORT VECTOR MACHINES

Support Vector Machines belong to the neural networks family and kernel-based methods [4] [8] [19], and they have been largely developed for many application domains. As this, the SVM output function is a mapping from $\mathcal{X}$ (the input space) to $\mathcal{R}$ which is of the following form[1]:

$$h(\mathbf{x}) = \sum_{i}^{\ell} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b. \tag{1}$$

The vector α and the real threshold value b are the parameters characterizing the machine. The function k is a *Mercer kernel* which is used as an inner product in a high dimensional (even infinite) space Ψ known as a *feature space*. The use of an inner product naturally arises from the SVM learning as the basic idea of such a machine is to find a separating hyperplane of the transformed input data $(\phi(\mathbf{x})_1, y_1), \ldots, (\phi(\mathbf{x})_\ell, y_\ell)$ in Ψ with $\phi : \mathcal{X} \to \Psi$. The idea is to find a hyperplane of equation $\mathbf{w} \cdot \phi(\mathbf{x}) + b = 0$ doing such a separation, the introduction of the non-linear transformation ϕ allowing to deal with non-linearly separable data (for more details, see [4]). If k meets Mercer's condition there then exists a mapping ϕ from $\mathcal{X}$ to a space Ψ such that for any couple $(\mathbf{u}, \mathbf{v})$ in $\mathcal{X}^2$, $k(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u}) \cdot \phi(\mathbf{v})$. One can then focus on the search of an adequate mapping ϕ or a kernel k. It turns out that the easier function to find is the kernel function as some of them are well known and studied such as the gaussian kernel (also called Radial Basis kernel):

$$k(\mathbf{u}, \mathbf{v}) = \exp(-\gamma \|\mathbf{u} - \mathbf{v}\|^2) \tag{2}$$

This is the kernel we choose to work with.

Once k has been chosen, the vector $\mathbf{w}$ and the threshold b are obtained from the quadratic problem:

$$\max_{\alpha} \quad \mathbf{W}(\alpha) = \alpha' \mathbf{1}_\ell - \frac{1}{2} \alpha' \mathbf{K} \alpha \tag{3}$$

$$\text{s.t.} \quad \begin{cases} \alpha' \mathbf{y} = 0 \\ 0 \leq \alpha \leq C \end{cases} \tag{4}$$

In those equations, $'$ denotes transposition, bold letters indicate column vector, $\mathbf{1}_d$ is the column vector of dimension d filled with 1 and K the kernel matrix $K = (y_i y_j k_{ij})_{1 \leq i,j \leq \ell}$, k_{ij} shorthanding $k(\mathbf{x}_i, \mathbf{x}_j)$. C is a user-defined constant which

[1] We use the same notations as those seen earlier

represents the trade-off between the model complexity and the approximation error. This problem can be efficiently solved with Platt's SMO implementation [16]; its solution and the optimality conditions give $\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \phi(\mathbf{x}_i)$, b and the so-called *support vectors* which are training points $\mathbf{x}_i$ with strictly positive α_i.

LOCAL INCREMENTAL LEARNING OF A SUPPORT VECTOR MACHINE

State of the art in SVM area

The Kernel-Adatron algorithm [10] is a very fast approach to approximate the solution of the support vector learning and can be seen as a component-wise optimization algorithm. It has been succesfully tested by their authors to dynamically adapt the kernel parameters of the machine, doing model selection in the learning stage. Nevertheless, the main drawback of this work is that it should not be straightforward to extend this work to deal with drifting concepts.

Another approach, proposed in [18], consists in learning new data by discarding all past examples except support vectors. The proposed framework thus relies on the property that support vectors summarize well the data and has been tested against some standard learning machine datasets to evaluate some goodness criteria such as *stability*, *improvement* and *recoverability*.

Finally, a very recent work [5] proposes a way to incrementally solve the global optimization problem in order to find the exact solution with the help of algebraic reformulation of the problem (3) and bookkeeping stratgey. Its reversible aspect allows to make "decremental" unlearning and to efficiently compute leave-one-out estimations.

Local approach for incremental SVM

With SVM, the local approach is especially made easier by two typical properties of this machine. Indeed, measuring the degree of responsibility of a given parameter $\theta_i = \alpha_i$ is given by :

$$\frac{\partial h(\mathbf{x}_t)}{\partial \alpha_i} = y_i k(\mathbf{x}_t, \mathbf{x}_i) \tag{5}$$

Thus, using the following metric $\left|\frac{\partial h(\mathbf{x}_t)}{\partial \alpha_i}\right|$ with a RBF kernel k to determine how responsible α_i is for the prediction concerning $\mathbf{x}_t$ is equivalent to use the euclidean L_2 metric in the input space directly. This forms the first attractive property of SVM. Second, as the machine acts as a voting machine that combines the outputs of experts based on training data, observing a new data automatically leads to add an expert linked with this data and therefore, a new scalar parameter α_t. In other words, this means that the SVM algorithm is constructive by definition and the addition step systematically preceeds the modification step.

Applying the local strategy implies that we define which experts we are going to ask advice. Table 1 sketches the updating procedure to build h^t from h^{t-1} when the incoming instance $(\mathbf{x}_t, y_t)$ is to be learned, $\mathcal{S}_{t-1}$ being the set of instances learned so far, and $\mathcal{S}_t = \mathcal{S}_{t-1} \cup \{(\mathbf{x}_t, y_t)\}$. The strategy to take a new point $\mathbf{x}_t$ into account consists in considering growing – with respect to the number of points – neighborhoods around $\mathbf{x}_t$ in the input space, and, for each of them, to re-adapt the candidate hypothesis until a stopping rule is verified. We discuss and define the stopping rule in subsection

TABLE 1. Core algorithm

1. if $y_t h^{t-1}(\mathbf{x}_t) > 1$ then stop (the current hypothesis classifies $\mathbf{x}_t$ with enough security)
2. $n \leftarrow 1$
3. repeat
   - $n \leftarrow n+1$
   - Add to the working subset (working neighborhood) the n-nearest neighbors of $\mathbf{x}_t$
   - Learn a candidate hypothesis h_n^t by optimizing the quadratic problem on examples in the working subset.
4. until a stopping criterion $C_t(n)$ is verified.

Automatic selection of a working neighborhood

Fixing *a priori* a number n of neighbors at each new arrival of data. Obviously, it presents two drawbacks: first we do not know which value to choose and second, there is no reason that each pattern requires the same number of neighbors. It depends both on the localization of the pattern with respect to the hyperplane in the feature space and on the statistics of the patterns already observed. This pleads for a method that adapts itself to the data. We thus need to define the goal of local incremental learning and translate it into a cost funtion. Our strategy aims at finding a balance between good ability to generalize and a small number of neighbors to be re-considered. However, as the examples come online, the only generalization error that we can estimate is the instantaneous generalization error $\varepsilon_G^n(t)$ that the SVM from n neighbors would produce at step t. We thus propose two strategies based on the minimization of an estimation of the generalization error: one can be considered as *early stopping* and the other corresponds to *weight-decay* where n, the number of neighbors is the "weight".

Early stopping: The first strategy comes directly from the observation that the local learning algorithm cannot do better than the global one: at the beginning of the training process, one may choose the accuracy δ that the learnt hypothesis is supposed to reach: $C_t(n) : \widehat{\varepsilon}_G^n < \delta$

Weight-decay: The second strategy defines a cost function that measures how the parameter k verifies the double goal of a reasonable time-complexity and a low generalization error. The criterion $J_t(n)$ is to be minimized at each time t, giving the stopping rule: $C_t(n) : (J_t(n+1) - J_t(n)) > 0$

$$J_t(n) = \widehat{\varepsilon}_G^n(t) + \lambda \left(\frac{n}{t}\right)^2 \tag{6}$$

λ defines the importance of the time-complexity with respect to the generalization error.

One way to do that is to estimate error on an additionnal data (a validation set) set and thus keeping n that mnimizes this estimation ε_{Val}. However it does not seem to be realistic to get a validation set during the incremental learning process. Elsewhere, there exist some analytical expression of Leave-One-Out estimates of SVMs generalization error such as those recalled in [6]. However, in order to use these estimates, one has to ensure that the margin optimization problem has been solved exactly. The same holds for the $\xi\alpha-$estimators of Joachims [12, 14]. This restriction prevents from using these estimates as we only do a partial local optimization.

To circumvent the problem, we use the bound on generalization provided by a result of Cristianini and Shawe-Taylor [8] for thresholded linear real-valued functions. The theorem follows:

Theorem 1 *Consider thresholding real-valued linear functions $\mathcal{L}$ with unit weight vectors on an inner product space X and fix $\gamma \in \mathcal{R}^+$. There is a constant c, such that for any probability distribution $\mathcal{D}$ on $X \times \{-1,1\}$ with support in a ball of radius R around the origin, with probability $1-\eta$ over ℓ random examples S, any hypothesis $g \in \mathcal{L}$ has error no more than*

$$\varepsilon_G = err_{\mathcal{D}}(g) \le B = \frac{c}{\ell}\left(\frac{R^2 + \|\xi\|^2}{\gamma^2}\log^2 \ell + \log\frac{1}{\eta}\right) \tag{7}$$

where $\xi = \xi(\gamma, h, S) = (\xi_1, \xi_2, \ldots, \xi_\ell)$ is the margin slack vector with respect to f and γ defined as $\xi_i = \max(0, \gamma - y_i g(\mathbf{x}_i))$.

We notice that once the kernel parameter σ is fixed, this theorem, directly applied in the feature space Ψ_σ defined by the kernel K_σ, provides an estimate of generalization error for the machines we work on. This estimate is expressed in terms of a margin value γ, the norm of the slack margin vector ξ and the radius of the ball containing the data.

In order to use this theorem, we consider the feature space of dimension $d(K)$ defined by the Gaussian kernel with a fixed value for σ. In this space, we consider $\mathcal{L}$ with unit weight vectors. At step t, different functions h_n^t can be learnt with $n = 1, \ldots, N_{\max}$. For each n, we get a function g_n^t by normalizing the weight vector of h_n^t. h_n^t belongs to $\mathcal{L}$ and when thresholded provides the same outputs than h_n^t does. The theorem is then applied to $g = g_n^t$ and data of $\mathcal{S}_t$. It ensures that:

$$\varepsilon \le B(c(\gamma, \mathcal{L}), R_g, \xi_g, \gamma, t). \tag{8}$$

Hence, for each $n = 1 \ldots N_{\max}$, we can use this bound as a test error estimate. However, as R_g is the radius of the ball containing the examples in the feature space, it only depends on the chosen kernel and not on k. On the contrary, ξ_g, defined as: $\xi_{i,k}^t = \max(0, \gamma - y_i g_n^t(\mathbf{x}_i))$ is the unique quantity which differs among functions f_k. The slack margin vectors ξ_n^t are thus sufficient to compare g_n functions. Now we are able to simplify the criterion of equation (6) to be minimized: $J_t(n) = \frac{\left(\xi_{i,n}^t\right)^2}{\gamma^2 t} + \lambda \left(\frac{n}{t}\right)^2$. Moreover, at each time t, a value of γ must be chosen: in order to do that, we take a time-varying value defined as $\gamma_t = 1/\|\mathbf{w}_{t-1}\|$.

NUMERICAL RESULTS

Experiments have been conducted on three benchmarks *Banana*, *Ringnorm* and *Diabetes*, available at *www.first.gmd.de/~raetsch/*. All these batch problems have been transformed into online problems with one presentation for each pattern. 20 samples have been used to get average results and standard deviation error.

We compared four methods on each problem (see tables 2, 3):

- *Rebatch* that consists in re-learning the whole set of example at each step t
- *k-LISVM* that uses a constant fixed value k of n during all the learning process
- δ-*LISVM* that finds the minimal n_t that allows to reach a fixed accuracy (implementing early-stopping)
- λ-*ALISVM* that finds the first minimum of the criterion $J_t(n)$ involving a weight decay term.

We choose to systematically use the three same values for the varying parameter of each algorithm in order to test the sensitivity of the algorithm.

For *Banana* problem, we ran more extensive tests. The full database has been separated into three sets: a training set, a validation set and a test set. This allowed us to compare the proposed method (use of slack margin vector) to choose the appropriate k at eatch step t with the expensive method that makes the selection according to the validation error.

TABLE 2. Results for the different strategies on the Banana dataset. $C = 100.0$, $\sigma = 1.0$. The first line corresponds to the batch algorithm. The second line corresponds to the expensive strategy that selects $\hat{k}_t$ according the minimal validation error

		Test Error rate (%)	$\mathbf{w}^2$	Number of Svs	$\|\|\xi\|\|^2/\ell$
Rebatch		10.6 ± 0.5	879 ± 184	94 ± 10	0.298 ± 0.042
Best validation error		11.3 ± 0.9	625 ± 244	42 ± 18	0.76 ± 0.30
K	10	18.8 ± 5.4	603 ± 223	104 ± 12	0.929 ± 0.637
	20	18.6 ± 5.3	566 ± 154	100 ± 12	0.868 ± 0.587
	50	16.9 ± 4.9	530 ± 151	78 ± 12	0.531 ± 0.268
δ	0.001	11.1 ± 0.6	995 ± 218	87 ± 13	0.283 ± 0.04
	0.01	11 ± 0.6	1050 ± 151	91 ± 9	0.291 ± 0.049
	0.1	11.2 ± 0.8	982 ± 190	90 ± 11	0.296 ± 0.036
λ	0.1	12 ± 1.4	771 ± 291	58 ± 14	0.347 ± 0.05
	0.5	14.5 ± 2.5	489 ± 284	35 ± 13	0.452 ± 0.113
	1.0	15.9 ± 2.4	340 ± 250	27 ± 12	0.537 ± 0.168

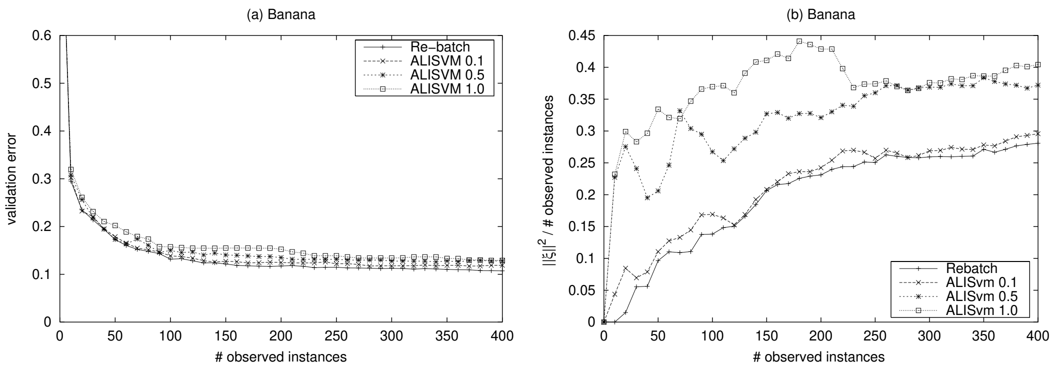

FIGURE 1. Evolution of Test Error and $\|\xi_{t,\hat{n}_t}\|^2/\ell_t$ during ALISVM processing for *Banana*

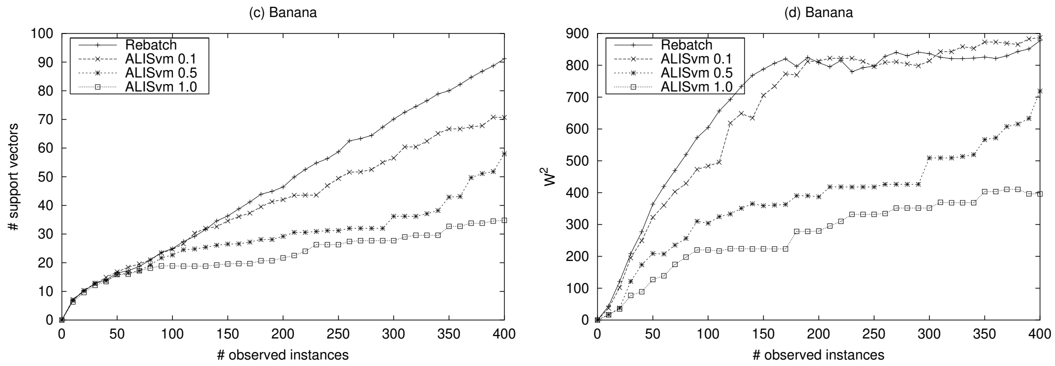

FIGURE 2. Number of SVs and squared norm of **w** during ALISVM processing for *Banana*.

TABLE 3. Test error rates (%) for Ringnorm and Diabetes problems

		Ringnorm	Diabetes
Rebatch		2.52 ± 0.41	23.3 ± 1.6
K	10	4.2 ± 1.6	39 ± 13.8
	20	4 ± 1	32 ± 10
	50	3.2 ± 0.7	29.6 ± 4.8
δ	0.001	2.8 ± 0.6	23.6 ± 1.6
	0.01	3.8 ± 1	23.8 ± 1.8
	0.1	7.4 ± 1.4	23.5 ± 1.7
λ	0.1	3.3 ± 0.6	26.7 ± 2.5
	0.5	6.5 ± 1.8	26.6 ± 2.6
	1.0	6.5 ± 1.3	27.3 ± 2.8

First, in table 2,the k-LISVM is not efficient while δ-LISVM has a very low error rate but at a high complexity cost. This means that the δ has been chosen too small. On the contrary, λ-ALISVM degrades slowly for increasing λ with a low complexity (norm of **w**).

The study of Figure 1 shows that for a low value of λ, ALISVM behaves like the re-batch algorithm that re-learns all the current training set at each time t for a low value of λ. Moreover, for the highest value of λ the ratio $\#SVs/\ell$ is drastically reduced and quasi stabilizes at the end of new data arrival (Figure 2). An interesting remark concerns the adequation between the behaviour of the squared norm of the slack margin vector and the test error (almost linear in table 1). The same remarks stand for Ringnorm and Diabetes problems. Finally, CPU-time together with

TABLE 4. CPU times in seconds for the three different problems

		Ringnorm	Banana	Diabetes
Rebatch		42.1 ± 7.3	58.3 ± 12.1	2.5 ± 1
K	10	1.87 ± 0.34	0.68 ± 0.15	0.17 ± 0.05
	20	9.06 ± 1.75	2.15 ± 0.6	0.21 ± 0.05
	50	27.6 ± 4.6	8.07 ± 2.52	0.34 ± 0.04
δ	0.001	33.3 ± 9.4	350 ± 168	120 ± 35
	0.01	9.4 ± 3.6	300 ± 150	130 ± 37
	0.1	2.1 ± 1.6	146 ± 128	111 ± 24
λ	0.1	60.8 ± 40	21.5 ± 16.6	0.16 ± 0.1
	0.5	10 ± 7.9	12.6 ± 11.3	0.2 ± 0.14
	1.0	10.6 ± 11	13.9 ± 21.2	0.273 ± 0.034

the complexity of the machines provide a criterion to measure the best algorithm: λ-ALISVM offers a good balance between a reduction in time and good generalization capabilities.

EXTENSIONS

ALISVM for regression task

We presented extensively the details of the local approach applied to the framework of incremental supervised classification and SVM. For SVM, there also exists a usable theorem also proven by Cristianini and Shawe-Taylor [8] that suits the regression case. The theorem states as follows:

Theorem 2 *Consider performing regression with linear functions $\mathcal{G}$ on an inner product space $\mathcal{X}$ and fix $\gamma \leq \theta$ with $\theta > 0$. There is a constant c, such that for any probability distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{R}$ with support in a ball of radius R around the origin, with probability $1-\eta$ over ℓ random examples S, any hypothesis $g \in \mathcal{L}$* MORE THAN θ AWAY FROM ITS TRUE VALUE IS BOUNDED BY AS*:*

$$\varepsilon(g) = err_{\mathcal{D}}(g) \leq B = \frac{c}{\ell}\left(\frac{||\mathbf{w}||_2^2 R^2 + ||\xi||_2^2}{\gamma^2}\log^2 \ell + \log\frac{1}{\eta}\right) \tag{9}$$

where $\xi = \xi((\mathbf{x}_i, y_i), g, \theta, \gamma) = (\xi_1, \xi_2, \ldots, \xi_\ell)$ is the margin slack variable with respect to g, to target accuracy θ and loss margin γ defined as $\xi_i = \max(0, |y_i - g(\mathbf{x}_i)| - (\theta - \gamma))$.

This result can be injected in the algorithm 1 and used to automatically choose the number of neighbors to be considered.

Drifting concepts

When the concept to be learnt changes with time, a temporal window can be aplied to the training data observed so far to select the most recent ones. This temporal window also called forgetting factor or scheduling factor by several authors can be automatically moved.

Local approach for Hybrid decision trees

Hybrid decision trees [1] and general regression trees [3] are characterized by the following definition :

$$h(\mathbf{x}) = \sum_{k=1}^{n_l} \alpha_k(\mathbf{x}) h_k(\mathbf{x}) \tag{10}$$

where n_l is the number of the leaves in the tree $\alpha_k(\mathbf{x})$ can be decomposed as a product of functions $a_{anc(i)}$ which are functions associated to intermediate nodes in the tree and $h_k(\mathbf{x})$ are classifiers or regressors associated with leaves of the tree.

$$\alpha_k(\mathbf{x}) = \Pi_{j=0}^{father(k)} a_j(\mathbf{x}) \tag{11}$$

Functions h_k have their outputs in the same set (space) that the function h. If function a_j have their outputs in $\{0,1\}$ then the tree becomes a soft mixture of classifiers. In general, one prefers the crisp outputs with the interval $[0,1]$ with the benefits of reduced time-complexity for the calculations of the output. We first note that the local strategy is facilitated by the fact that the parameters responsible for the ouput of the tree can be easily ordered by the means of the tree depth. For the explanation, we suppose that the nodes of the tree are numbered according to their location in the binary tree. Let us denote $l(t)$ the number of the leaf where the pattern $\mathbf{x}_t$ falls and $d(t)$ the depth of this leaf. Let us define the increasing inclusion: $\Theta_t(0) \subset \Theta_t(1) ... \subset \Theta_t(d(t))$ IS THE SUBSET OF PARAMETERS IN THE TREE THAT ARE THE MOST RESPONSIBLE FOR THE OUTPUT OF THE TREE, GIVEN $\mathbf{x}_t$.

Therefore the automatic selection of the working subset of parameters given an observation is equivalent at which level the tree should be re-considered. Moreover, like for SVM, this means that only a localized part of the decision surface will be modified, while most of the decision surface remains fixed.

CONCLUSION

We proposed to enhance general incremental learning method by reducing the time-complexity of the updating step with a local approach. The principle of the method consists in automatically choosing which of existing parameters need to be re-optimized and whether if adding new parameters is necessary. Some learning systems such as SVM, Radial Basis Functions or hybrid decision trees are especially well suited for this approach. Indeed, in the case of SVM based on Gaussian kernels, the metric used to measure the most useful working subset of parameters is exactly the input space metric. Moreover, the observation of a pattern directly provides a new parameter to add. The local strategy has just to concentrate on the choice of the existing parameters that are concerned with this new pattern. In this paper, we focus on the Support Vector machine and present for this learner, a detailed version of the local approach. We especially define an objective function that measures how efficient is the algorithm in terms of time-complexity and how well the classifier obtained generalizes. The generalization capabilities are measured through a quantity that is involved in theoretical bounds for generalization error: the squared norm of the slack margin vector. Numerical simulations show interest of the method in the case of classification. Incremental Regression can also be solved using equivalent theoretical bounds but for regression.

The proposed algorithm can also be extended to deal with drifting concepts using a working neighborhood that falls also within a temporal window. Moreover, if some of the observed data are unlabeled, the local approach could be mixed with semi-supervised learning methods devoted to SVM [9] or to large margin classifiers [2].

REFERENCES

1. F. d'Alche-Buc, D. Zwierski and J.-P. Nadal. Hybrid trio-learning,Int. J. of Neural Systems, december, 1994.
2. F. d'Alche-Buc,Y. Grandvalet and C. Ambroise, Semi-Supervised MarginBoost, submitted to NIPS'01, Vancouver, 2001.
3. L. Breiman and, J. H. Friedman, Classification and regression trees, 1983.
4. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):955–974, 1998.
5. G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Adv. Neural Information Processing*, volume 13. MIT Press, 2001.
6. O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing kernel parameters for support vector machines. Technical report, AT&T Labs, March 2000.
7. C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:1–25, 1995.
8. Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based methods* Cambridge University Press, 2000.
9. K. P. Bennet and A. Demiriz. Semi-supervised support vector machines. In *Advances in Neural Information Processing Systems*, D. Cohn and M. Kearns and S. Solla (eds),pp. 368-374,1999.
10. T. Friess, F. Cristianini, and N. Campbell. The kernel-adatron algorithm: a fast and simple learning procedure for support vector machines. In J. Shavlik, editor, *Machine Learning: Proc. of the* 15^{th} *Int. Conf.* Morgan Kaufmann Publishers, 1998.
11. L. Devroye, L. Gyorfi and G. Lugosi. A Probabilistic Theory of Pattern Recognition ., SPringer Verlag, New York, 1997.
12. T. Joachims. Estimating the generalization performance of a svm efficiently. In *Proc. of the* 17^{th} *ICML*. Morgan Kaufmann, 2000.
13. C. Jutten and R. Chentouf, Neural processing Letters, 2,1,pp. 1-4, 1995.
14. R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *Proc. of the* 17^{th} *ICML*. Morgan Kaufmann, 2000.
15. E. Osuna, R. Freund, and F. Girosi. Improved training algorithm for support vector machines. In *Proc. IEEE Workshop on Neural Networks for Signal Processing*, pages 276–285, 1997.
16. J.C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report 98-14, Microsof Research, April 1998.
17. G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. Technical Report NC-TR-1998-021, Department of Computer Science, Royal Holloway, University of London, Egham, UK, 1998.
18. N. Syed, H. Liu, and K. Sung. Incremental learning with support vector machines. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 1999.
19. V. Vapnik. *The nature of statistical learning theory*. Springer, New York, 1995.
20. L. Ralaivola and F. d'Alché-Buc. Incremental Support Vector Machine Learning: a Local Approach. To appear in *Proc. of ICANN'01*, Austria, 2001.
21. T. -Y. Kwok and D. -Y. Yeung. Objective Functions for Training New Hidden Units in Constructive Neural Networks. IEEE Transactions on Neural Networks, vol. 8, number 5, pp. 1131–1148, `http://citeseer.nj.nec.com/kwok99objective.html`, 1997.