8-2022

Weighted Incremental–Decremental Support Vector Machines for concept drift with shifting window

Honorius Gâlmeanu

Răzvan Andonie

Neural Networks 152 (2022) 528–541

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Weighted Incremental–Decremental Support Vector Machines for concept drift with shifting window

Honorius Gâlmeanu [a,*], Răzvan Andonie [b,c]

[a] Faculty of Mathematics and Computer Science, Transilvania University of Braşov, Braşov, Romania
[b] Computer Science Department, Central Washington University, Ellensburg, WA, USA
[c] Transilvania University of Braşov, Braşov, Romania

ARTICLE INFO

Article history:
Received 6 July 2021
Received in revised form 11 May 2022
Accepted 17 May 2022
Available online 27 May 2022

Keywords:
Support Vector Machines
Concept drift
Incremental learning
Shifting window

ABSTRACT

We study the problem of learning the data samples' distribution as it changes in time. This change, known as concept drift, complicates the task of training a model, as the predictions become less and less accurate. It is known that Support Vector Machines (SVMs) can learn weighted input instances and that they can also be trained online (incremental–decremental learning). Combining these two SVM properties, the open problem is to define an online SVM concept drift model with shifting weighted window. The classic SVM model should be retrained from scratch after each window shift. We introduce the Weighted Incremental–Decremental SVM (WIDSVM), a generalization of the incremental–decremental SVM for shifting windows. WIDSVM is capable of learning from data streams with concept drift, using the weighted shifting window technique. The soft margin constrained optimization problem imposed on the shifting window is reduced to an incremental–decremental SVM. At each window shift, we determine the exact conditions for vector migration during the incremental–decremental process. We perform experiments on artificial and real-world concept drift datasets; they show that the classification accuracy of WIDSVM significantly improves compared to a SVM with no shifting window. The WIDSVM training phase is fast, since it does not retrain from scratch after each window shift.

1. Introduction

Some machine learning tasks are based on datasets collected continuously over a long period of time. It may happen that, as we collect more and more data, historic data becomes less and less relevant for the task at hand. This is due to the intrinsic data distribution changing over time (i.e., the source generating the data is not stationary). For example, energy companies collect various characteristics regarding their customers, in order to compare their consumption habits. This leads to a better estimation of future customer energy requirements. However, the profiles of customer consumption tend to change over time, and the model should adapt dynamically to these changes.

Another example comes from climate prediction. For the past 40 years, computer models have delivered quite a consistent picture of how fast human carbon emissions might warm up the world. However, a host of global climate models developed for the United Nation's next major assessment of global warming, due in 2021, is now showing a new trend. They show higher temperatures than previous predictions (Voosen, 2019).

Such variations of a regular pattern of evolution over time are known as *concept drift*, meaning that the input data may shift from time to time, each time after some minimal persistence (Gama, 2010). Many machine learning algorithms can approximate stationary distributions with arbitrary accuracy (bounded by the Bayes error) whenever the number of training samples increases to infinite (Gama, 2010). Detecting concept drift in a non-stationary environment is more difficult. In this case, the distribution of the input data stream changes in time. It is hard to distinguish whether a new data sample represents *noise* or *change*. The difference between noise and change is given by *persistence*. For change, there should be a consistent set of samples from the new distribution. To detect change, we must combine robustness to noise/outliers with sensitivity to concept drift (Gama, 2010). Concept drift is better associated to gradual changes than to abrupt changes. For abrupt changes we use the term *concept shift* (Gama, 2010). According to Lazarescu, Venkatesh, and Bui (2004), concept drift is defined in terms of consistency and persistence. Noise and outliers are not consistent or persistent. Recent overviews of techniques, methods and applications of concept drift can be found in Gama, Žliobaitė, Bifet,

* Corresponding author.
E-mail addresses: galmeanu@unitbv.ro (H. Gâlmeanu), andonie@cwu.edu (R. Andonie).

https://doi.org/10.1016/j.neunet.2022.05.018

Pechenizkiy, and Bouchachia (2014), Iwashita and Papa (2019) and Lu et al. (2019).

Research on techniques that handle concept drift with machine learning methodologies is presently highly needed (Lu et al., 2019). An important research focus is how to embed concept drift detection in the learning phase of machine learning models. According to Farid et al. (2013), there are three main approaches: *(a)* instance selection or window-based approach, *(b)* weight-based approach, and *(c)* ensemble of classifiers.

Most algorithms adapted to concept drift, including our approach, consider online (incremental) learning algorithms, which are able to adapt to the evolution of the data generating process over time. A taxonomy of online algorithms that learn from evolving data is presented in Gama et al. (2014). In incremental machine learning, a common assumption is that the most recent training samples are the most relevant; a mechanism controls the gradual forgetting of the old information. Therefore, a natural concept drift handling technique is based on instance selection: a window moves over recently arrived instances and the learned concepts are used for prediction only in the immediate future (Tsymbal, 2004). The forgetting mechanism can be extended by weighting the samples in a time window — its size can be fixed or variable. However, the time window approach may not always be effective. For instance, when the data is noisy or when the change is slower than the window size, windowing may fail as well (Gama et al., 2014).

To implement an incremental concept drift model with a shifting (or sliding) window, we have to answer the following questions: *(i)* how the model can learn new information without corrupting/forgetting previously learned information (the stability-plasticity dilemma (Carpenter & Grossberg, 1988)), and *(ii)* how the model can forget obsolete information without corrupting/forgetting still valid, already learned information. In short, how can the model selectively learn and forget in a controlled and dynamic way?

In the following, we say that a concept drift model with shifting window has the Shifting Window (SW) Property if: *(a)* After each shift, it learns the new sample and forgets the oldest one without being retrained on the other samples; and *(b)* The drift model produces for each window shift exactly the same results as when it is trained from scratch on all samples within the window. In other words, in a model with the SW Property, shifting the window consists of two operations: adding a new sample to its head and removing the oldest sample from the tail. Retraining from scratch on the new configuration is not necessary. This implies the condition of incremental–decremental training, since the new configuration of the shifted window is the same as the one obtained by retraining the new configuration from scratch. Using the SW Property we can make a SVM learn faster than learning from scratch. This is a good reason to have SVMs fulfill the SW Property.

A SVM with the SW Property can be regarded as a generalization of the incremental–decremental SVM SMO algorithm (Cristianini & Shawe-Taylor, 2000). The incremental–decremental SVM training algorithm achieves the SW property by shifting between configurations for which Kuhn–Tucker conditions are always satisfied. Since this is a convex optimization problem, having a unique solution is the rule rather than the exception (Burges & Chrisp, 1999).

We introduce a SVM with the SW Property, capable of learning from data samples with concept drift using the shifting weighted window technique. We obtain the same classification accuracy as a model trained from scratch (non-incremental), but with significantly shorter training time. In our approach, the SW Property is a particular case of the weighted incremental–decremental algorithm, corresponding to a rectangular weight profile. However, our model can be applied to any arbitrary weight profile. In this case, the characteristic λ values that form the solution will satisfy a more constrained set of Kuhn–Tucker conditions.

1.1. Related work and motivation

Several concept drift models with shifting window approaches were proposed during the last years. Comprehensive compilations can be found in Iwashita and Papa (2019) and Lu et al. (2019).

The Learn++.NSE algorithm (Elwell & Polikar, 2009, 2011) and its fast version (Shen, Zhu, Du, & Chen, 2018) use an ensemble of classifiers for the incremental learning of concept drift. They are examples of non-stationary environments, where the underlying data distributions change over time. Learn ++.NSE trains one new classifier for each batch of data it receives, and combines these classifiers using a dynamically weighted majority voting. The adaptive random forest algorithm for the classification of evolving data streams, introduced in Gomes et al. (2017) combines batch algorithm traits with dynamic update methods, to deal with evolving data streams. Variable length windows have been employed in FLORA algorithms (Widmer & Kubat, 1996).

The problem of concept drift and incremental learning has already been approached. The Cauwenberghs and Poggio (2000) (CP) algorithm, later extended in Diehl and Cauwenberghs (2003), introduced incremental–decremental training of SVMs. An analysis of an efficient implementation for individual learning of the CP algorithm was presented in Laskov, Gehl, Krüger, and Müller (2006), along with a similar algorithm for one-class learning. Practical implementation issues of the incremental–decremental CP algorithm were discussed in Gâlmeanu and Andonie (2008, 2009). This algorithm was also adapted for regressions (Gâlmeanu, Sasu, & Andonie, 2016; Ma, Theiler, & Perkins, 2003; Martin, 2002). The CP algorithm is an exact online method to incrementally train an SVM, incorporating one example at a time and retaining the Kuhn–Tucker (KT) condition *on all previously seen data*.

Syed, Liu, and Sung (1999) introduced a concept drift SVM which could be trained incrementally on new data by discarding all previous data except their support vectors. An improved version of this method was provided by Rüping (2001). The incremental learning result is similar to the non-incremental result, but only if the last training set contains all examples that are also support vectors in the non-incremental case.

ZareMoodi, Siahroudi, and Beigy (2016) proposed a SVM classification model beyond the learned label space in data streams in the presence of concept-drift, where novel classes may emerge while processing the stream. For modeling intricate-shape class boundaries, they used data description methods based on support-vectors. Yalcin, Erdem, and Gurgen (2007) used SVMs in an ensemble-based incremental learning algorithm.

A weighted SVM for training shot-level representations of video sequences is described in Chang, Yu, Yang, and Xing (2017). Their method uses a linear SVM to learn the representation of samples, by building their relevance information into the architecture of the regularizer. This method, called *nearly isotonic SVM*, takes into account the pre-computed saliency of the input samples. On determining the input samples' weights that form the solution, it favors the configuration where the sorted order of the weights matches the pre-computed saliency order of input samples. This forces the important samples to weigh more in the expression of the solution (and the opposite for the weaker samples). There are some similarities with our proposed method, since we also impose additional constraints on the weights defined by the optimal solution. The nearly isotonic SVM encourages the input vectors' associated weights to take a certain precedence by using a carefully designed regularizer. However, our method imposes restrictions on the solution's weights by limiting their maximum allowed value — the limit is determined by the training profile. The nearly isotonic SVM uses the primal form expression

of the linear SVM and learns from scratch, using nearly isotonic regularization. In contrast, our method uses the dual formulation of the kernel-based SVM, with squared Frobenius norm as regularization, and does not require learning from scratch, being an incremental approach.

In order to deal with the accumulation of new samples, strategies for incremental learning using SVM have been recently proposed. Thus, Chitrakar and Huang (2014) use the Improved Concentric Circles method: the points in the ring region are kept for further training, while the others are discarded. This region is defined as the area between two circles: the inner one, determined by class center and the nearest support vector, and the outer one, tangent to the nearest point at the hyperplane. Only the points in this region are used for further incremental training. Chitrakar et al. also propose a Half-partition strategy, that considers as candidate support vectors all the vectors outside the inner circle, with the refinement of probabilistic weighting for those samples.

Other incremental learning strategies are described in Chen, Xiong, Xu, and Zuo (2019) and Wang and Xing (2019). In Chen et al. (2019), the Variable SVM speeds up the updates by using pre-calculated information and an incremental matrix update derived from the Sherman–Morrison–Woodbury formula (Laskov et al., 2006). Incremental cost-sensitive learning, a related method also based on the CP algorithm, is employed in Ma, Zhao, Wang, and Tian (2020) to deal with the class-imbalance problem in online situations. High-cost is implemented for minority class and low cost for majority class, using a linear-exponential function for the constraint in the expression of the Lagrangian. The incremental–decremental migrations of vectors across the sets are regulated by specific relations. Such relations were first defined in Laskov et al. (2006) and Ma et al. (2003), and later extended for a variable regularization parameter in Gâlmeanu and Andonie (2008) and Gâlmeanu et al. (2016).

Instance weighting and learning windows of variable length also appear in the papers of Klinkenberg (2004) and Klinkenberg and Joachims (2000). The authors introduce a method for recognizing and handling concept changes with SVMs. This method uses a variable size window of training samples. The window size is adjusted based on the estimate of the generalization error. At each time step, the algorithm builds several SVM models with various window sizes, then selects the one minimizing the error estimate. Klinkenberg's method shows how to select the appropriate window size, sample selection, and sample weighting for a SVM drift model with shifting window. While Klinkenberg et al. methods can be used online in applications, they are not incremental due to the following reasons (Klinkenberg, 2004): *(a)* incremental SVM approaches do not address the problem of concept drift, *(b)* the potential speed-up gained by using an incremental technique did not seem very significant, and *(c)* incremental learning sometimes leads to reduced accuracy compared to non-incremental learning. In our approach, we address all these questions: our proposed method is an incremental concept drift SVM method, exhibits significant speed-up gain, and does not sacrifice any accuracy compared to non-incremental learning.

SVMs have the ability to process weighted instances (Yang, Song, & Cao, 2007). In addition, SVMs can perform incremental–decremental learning (e.g., the CP algorithm). Combining these two properties, our goal is to define a SVM concept drift model with shifted weighted window and the SW Property.

None of the above SVM algorithms have the SW Property. Neither the CP method nor its successive developments take the problem of concept drift into consideration. The CP algorithm does not lead to accuracy loss compared to the non-incremental approach, but it can be computationally more expensive (Klinkenberg, 2004). There are also technical difficulties related to dealing with the support vectors representing the samples inside a shifting window. This may explain why the CP algorithm has not yet been used for concept drift and this gave us the motivation for the current work.

1.2. Our contribution

Our main contribution is the generalization of the CP algorithm for concept drift with shifting window. We name this concept drift model Weighted Incremental–Decremental SVM (WIDSVM). WIDSVM combines two of the above concept drift approaches: *(a)* instance selection (shifting window) and *(b)* weight-based. The shifting window uses weights assigned to its instances. In the current version, the window has a fixed size. The weights control how, as time passes, old information changes its relevance.

A classifier that considers the entire data set but also focuses on the current region of interest is not trivial. WIDSVM uses the following intuitive principle: it strengthens the margin classification constraint for some of the vectors, while relaxing this constraint for others. This approach imposes specific constraints for the most recently processed input vectors, while relaxing the constraints for the previously learned vectors. Past information becomes less and less relevant, in a manner dictated by the considered concept drift profile. The extreme case would be when the constraints are changed abruptly, by just removing the oldest vector from the shifting window, and adding the latest vector with full importance.

The technical contributions of this paper are summarized as follows:

- We show that the soft margin SVM problem with constraints imposed on the shifting window is reduced to an incremental–decremental SVM problem;
- We determine the exact symmetric migration conditions for incremental learning and decremental unlearning, that is, we determine which data sample will migrate next;
- We eliminate data inconsistencies that would lead to matrix singularities and infinite coefficient updates;
- We compute the initial SVM parameters starting from scratch, using two samples of opposite classes. Then, we compute the hyperplane location at mid-point distance. New data samples are added by incremental learning, starting from this two-point solution;
- We prove that WIDSVM has the SW Property;
- We apply the WIDSVM algorithm on several artificial and real-world benchmarks that have concept drift, and show that it can be trained faster than starting from scratch after each window shift.

The rest of the paper is structured as follows. Section 2 summarizes the CP incremental–decremental SVM algorithm. Section 3 introduces the WIDSVM algorithm, a generalization of the CP algorithm able to handle concept drift with shifting window. Section 4 presents and discusses experiments on artificial and real-world benchmarks with concept drift. Section 5 contains final remarks and open problems.

2. Background: Incremental–decremental SVM

Since the base of the WIDSVM training method is the CP algorithm, this section reviews the notations and theoretical framework of the CP algorithm.

2.1. SVM and the dual problem

The SVM computes the separating hyperplane w as a discriminator function $g(x)$:

$$g(x) = w^T x + b \qquad \text{where} \tag{1}$$

$$sign\{g(x)\} = \begin{cases} +1, & w^T x + b \geq 0 \\ -1, & w^T x + b < 0 \end{cases} \tag{2}$$

For a set of data samples x_i with labels $y_i \in \{-1, +1\}$, $i = 1, \ldots, N$, the separation hyperplane can be found by solving the primal optimization problem:

$$\min_w \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i \tag{3}$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0,\ i = 1, \ldots, N \tag{4}$$

Further, we define the penalty function $h(x_i)$ for data samples x_i as:

$$h(x_i) = y_i g(x_i) - 1 = y_i(w^T x_i + b) - 1 \tag{5}$$

For a correctly classified data sample, $g(x_i)$ has the same sign as its class label y_i and $h(x_i) > 0$ if the margin to the hyperplane is at least $\frac{1}{\|w\|}$. In this case, the constraint is not active, thus $\xi_i = 0$.

If the sample is not classified correctly within a sufficient margin distance, $h(x_i) < 0$ and the $\xi_i > 0$ penalty variable will compensate for this. Eq. (3) imposes the smallest ξ_i penalty. The parameter C controls the trade-off between increasing the margin $\frac{1}{\|w\|}$ and ensuring that most samples are correctly classified within this margin.

The associated Lagrangian for the primal problem is:

$$L(w, b, \lambda, \mu) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \mu_i \xi_i - \sum_{i=1}^{N} \lambda_i \left[y_i(w^T x_i + b) - 1 + \xi_i\right], \tag{6}$$

$$\text{where} \qquad \xi_i, \lambda_i, \mu_i \geq 0 \tag{7}$$

where λ_i and μ_i are the non-negative Lagrange multipliers.

For a feasible solution (w^*, b^*), the constraints are satisfied, thus the minimum is bounded below by

$$\frac{1}{2}\|w^*\|^2 + C\sum_{i=1}^{N} \xi_i \geq L(w^*, b^*, \lambda, \mu)$$

The expression $L(w^*, b^*, \lambda, \mu)$ is upper bounded and the solution can be found by solving:

$$\min_{w,b} \max_{\lambda,\mu} L(w, b, \lambda, \mu)$$

The problem is usually reformulated using the primal and dual problem notation:

$$\theta_P(w, b) = \max_{\lambda,\mu} L(w, b, \lambda, \mu) \quad \text{[the primal problem]} \tag{8}$$

$$\theta_D(\lambda, \mu) = \min_{w,b} L(w, b, \lambda, \mu) \quad \text{[the dual problem]} \tag{9}$$

$$\begin{aligned} \theta_D(\lambda^*, \mu^*) &= \max_{\lambda,\mu} \theta_D(\lambda, \mu) = \max_{\lambda,\mu} \min_{w,b} L(w, b, \lambda, \mu) \\ &\leq \min_{w,b} \max_{\lambda,\mu} L(w, b, \lambda, \mu) \\ &= \min_{w,b} \theta_P(w, b) = \theta_P(w^*, b^*) \end{aligned} \tag{10}$$

Assuming that the constraint functions used in the Lagrangian are affine and convex, the primal and dual problems have the same solution: $\theta_D(\lambda^*, \mu^*) = \theta_P(w^*, b^*)$. Therefore, solving the primal problem is equivalent to solving the dual problem, where the dual problem transforms to:

$$\max_{\lambda} \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j \tag{11}$$

$$\text{subject to} \quad \begin{matrix} \sum_{i=1}^{N} \lambda_i y_i = 0 \\ 0 \leq \lambda_i \leq C \end{matrix} \qquad i = 1, \ldots, N \tag{12}$$

2.2. The role of Kuhn Tucker (KT) conditions in incremental learning

For a feasible solution, the dual problem must verify the Kuhn–Tucker (KT) conditions:

$$\frac{\partial L}{\partial w_i} L(w, b, \lambda, \mu) = 0 \qquad i = 1, \ldots, N \tag{13}$$

$$\frac{\partial L}{\partial b} L(w, b, \lambda, \mu) = 0 \tag{14}$$

$$\lambda_i[h(x_i) + \xi_i] = 0 \qquad i = 1, \ldots, N \tag{15}$$

$$h(x_i) + \xi_i \geq 0 \qquad i = 1, \ldots, N \tag{16}$$

These conditions, and especially (12), impose stronger conditions on λ_i, so that (11) does not actually allow much flexibility. Thus, the dual solution is directly determined by the constraint of Eq. (13), which gives the optimal solution:

$$w^* = \sum_{i=1}^{N} \lambda_i y_i x_i \tag{17}$$

With further substitution in Eq. (1), the expression of the separating hyperplane becomes:

$$g(x_i) = \sum_{j=1}^{N} \lambda_j y_j x_j^T x_i + b \tag{18}$$

The hyperplane only depends on the scalar products of the input vectors x_i.

To increase the dimensionality of the initial input space where the linear separation is performed, SVMs usually employ the kernel trick. We use the kernel functions expressed as $K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$, where $\Phi(\cdot)$ is the mapping in a space where we can express the dot-product using the kernel property.

For the definition of the incremental–decremental SVM, we use the shortcut notation:

$$Q_{ij} = y_i y_j K(x_i, x_j) \tag{19}$$

The expression of the separation hyperplane defined in Eq. (18) becomes:

$$g(x_i) = \sum_{j} \lambda_j y_i Q_{ij} + b, \tag{20}$$

where we considered that y_i is limited to $\{-1, +1\}$ only. The expression of the penalty function defined by Eq. (5) becomes:

$$h(x_i) = \sum_{j} \lambda_j Q_{ij} + b y_i - 1 \tag{21}$$

There is a KT condition that allows for direct classification of the vectors used in the incremental–decremental SVM algorithm: the complementary slackness condition described by Eq. (15). In

this context, the penalty $h(x_i)$ takes the following values:

$$h(x_i) \text{ is } \begin{cases} > 0, & \lambda_i = 0 \quad \text{is met and } \xi_i = 0 \\ = 0, & 0 < \lambda_i < C, \\ & h(x_i) + \xi_i = 0 \quad \text{is met and } \xi_i = 0 \\ < 0, & \lambda_i = C, \\ & h(x_i) + \xi_i = 0 \quad \text{is met and } \xi_i > 0 \\ & \qquad \text{to compensate for} \\ & \qquad \text{the negative } h(x_i) \end{cases} \tag{22}$$

A vector x_i could belong to one of the following sets:

- **support vectors**, where $h(x_i) = 0$ and $0 < \lambda_i < C$; these define the separation hyperplane;
- **error vectors**, where $h(x_i) < 0$ and $\lambda_i = C$; these are situated on the wrong side of the separation hyperplane or in the separation region;
- **rest vectors**, where $h(x_i) > 0$ and $\lambda_i = 0$, which are the vectors situated on the correct side of the separation hyperplane, outside of the separation region.

2.3. Migration relations used in incremental learning

The context of the incremental–decremental SVM is completely determined by Eqs. (12), (21) and (22). The idea of the incremental SVM is to add to a stable state of vectors, described by these equations, a new vector x_c. It starts with a characteristic value $\lambda_c = 0$ and it increments this value, ensuring that Eqs. (12) and (15) defined by the KT conditions are fulfilled again. The value of λ_c is increased at the expense of the support vectors' λ_s values. In the process, vectors may migrate among the support, rest and error sets.

Before any vector migrates, λ_c is increased from zero at the expense of the support vectors' λ_s values and the free coefficient b in relations (12) and (21):

$$\Delta h_i = \Delta h(x_i) = \sum_s Q_{is}\Delta\lambda_s + y_i\Delta b + Q_{ic}\Delta\lambda_c \tag{23}$$

$$0 = \sum_s y_s\Delta\lambda_s + y_c\Delta\lambda_c \tag{24}$$

This way, the KT conditions are always satisfied. Hence, the λ coefficients in Eq. (21) remain either C or 0, and the sum of class-weighted coefficients of the support vectors remains zero (Eq. (24)).

To determine the migration conditions among sets, we must monitor the penalty cost variation for all vectors. We describe these variations for each set of vectors (the support vectors (s), the rest and error vectors (r), and the vector (c) to be learned) in matrix form:

$$\begin{bmatrix} \Delta h_s \\ \Delta h_r \\ \Delta h_c \\ 0 \end{bmatrix} = \begin{bmatrix} y_s & Q_{ss} \\ y_r & Q_{rs} \\ y_c & Q_{cs} \\ 0 & y_s \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta\lambda_s \end{bmatrix} + \Delta\lambda_c \begin{bmatrix} Q_{sc} \\ Q_{rc} \\ Q_{cc} \\ y_c \end{bmatrix} \tag{25}$$

During the first training phase, the support vectors do not migrate. Thus, their penalty h_s remains zero, meaning that also $\Delta h_s = 0$. Considering this, grouping the first and last relations in Eq. (25) yields:

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & y_s \\ y_s & Q_{ss} \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta\lambda_s \end{bmatrix} + \Delta\lambda_c \begin{bmatrix} y_c \\ Q_{sc} \end{bmatrix} \tag{26}$$

The variations of λ_s and b when λ_c changes can be expressed as:

$$\begin{bmatrix} \Delta b \\ \Delta\lambda_s \end{bmatrix} = \underbrace{-\begin{bmatrix} 0 & y_s \\ y_s & Q_{ss} \end{bmatrix}^{-1} \begin{bmatrix} y_c \\ Q_{sc} \end{bmatrix}}_{\beta_s} \Delta\lambda_c \tag{27}$$

or simply:

$$\begin{bmatrix} \Delta b \\ \Delta\lambda_s \end{bmatrix} = \beta_s\Delta\lambda_c \tag{28}$$

Substituting this solution into lines 2 and 3 of Eq. (25), we obtain the variation of penalties for the rest, error and newly added x_c vectors, respectively:

$$\begin{bmatrix} \Delta h_r \\ \Delta h_c \end{bmatrix} = \underbrace{\left\{ \begin{bmatrix} y_r & Q_{rs} \\ y_c & Q_{cs} \end{bmatrix} \beta_s + \begin{bmatrix} Q_{rc} \\ Q_{cc} \end{bmatrix} \right\}}_{\gamma_s} \Delta\lambda_c \tag{29}$$

or, in brief:

$$\begin{bmatrix} \Delta h_r \\ \Delta h_c \end{bmatrix} = \gamma_s\Delta\lambda_c \tag{30}$$

These are the penalty changes that we monitor during training.

3. Weighted incremental–decremental SVM

This section introduces the WIDSVM algorithm (see Algorithm 1), a generalization of the CP algorithm for concept drift with shifting window. The challenge is to constrain the WIDSVM to exactly fulfill the SW condition introduced in Section 1.

The concept drift problem can be reformulated as an incremental–decremental SVM. The window of input samples that slides throughout the dataset can be regarded as only shifting the focus to those samples that we want to learn. The rest of the samples are disregarded.

Referring to Eq. (3), we formulate the optimization problem for the entire dataset. We assign a weight, $\rho_i \in [0, 1]$, to the ξ_i constraint, responsible for the classification error. The constraint will only be active for the samples within the shifting window. The separation hyperplane solution, described by Eq. (20), is expressed as a linear combination of the trained vectors; with this in mind, we constrain the classifier to use only certain vectors, up to specific weights ($\rho_i \leq 1$), and disregard all of the others ($\rho_i = 0$). The ρ_i parameter indicates how relevant a sample x_i is in the current context. This way, we can gradually learn and unlearn samples.

The reformulated optimization problem aims to classify the data samples inside the window, with the smallest possible error. The classification error for the previous data samples is no longer relevant and will be ignored. Considering the weight (relevance) ρ_i, the problem rewrites as:

$$\min_w \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \rho_i\xi_i \tag{31}$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0,\ i = 1, \ldots, N \tag{32}$$

$$\text{where} \begin{cases} 0 < \rho_i \leq 1, & \text{if } x_i \text{ is inside the shifting window} \\ \rho_i = 0, & \text{otherwise} \end{cases} \tag{33}$$

If ξ_i is unconstrained ($\rho_i = 0$), condition (32) can be always fulfilled by taking a sufficiently large ξ_i. The constraint is removed in Eq. (31) for the samples outside the shifting window, allowing these ξ_i to vary freely. This SVM optimizes ξ_i only for the samples within the shifting window.

The Lagrangian in Eq. (6) changes to:

$$L(w, b, \lambda, \mu) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \rho_i\xi_i - \sum_{i=1}^{N} \mu_i\xi_i - \sum_{i=1}^{N} \lambda_i\left[y_i(w^T x_i + b) - 1 + \xi_i\right] \tag{34}$$

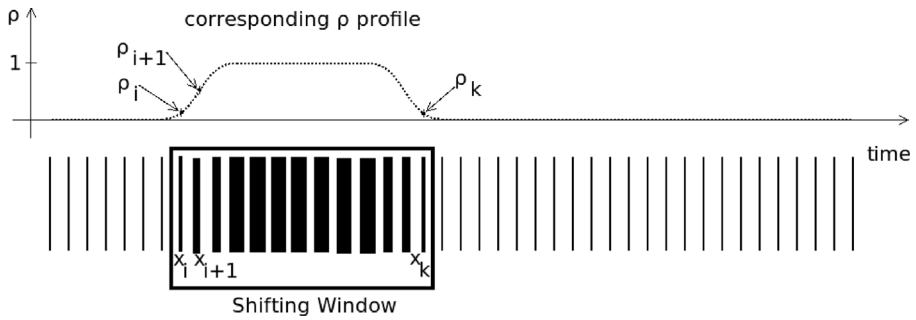

Fig. 1. An example of a weighted window shifting along the stream of samples. The sample thickness is proportional to the weight value ρ. The window size of this profile is $k - i + 1$.

where $\xi_i, \lambda_i, \mu_i \geq 0$,

$$\rho_i > 0 \text{ if } x_i \text{ is within the shifting window,} \quad \rho_i = 0 \text{ otherwise.} \tag{35}$$

When computing the first derivatives with respect to w_i, b, λ_i and ξ_i, the KT conditions do not change much, except for Eq. (12). We compute the derivative of the Lagrangian and make it zero:

$$\frac{\partial L(w, b, \lambda, \mu)}{\partial \xi_i} = C\rho_i - \mu_i - \lambda_i = 0 \tag{36}$$

Since $\mu_i \geq 0$, this leads to a slightly changed form of Eq. (12):

$$0 \leq \lambda_i \leq \rho_i C \tag{37}$$

This relation imposes that the data samples outside the window should have their associated $\lambda_i = 0$ (since $\rho_i = 0$).

Algorithm 1 is defined for an arbitrary training profile. An example of a training profile is depicted in Fig. 1. The window contains the samples $x_i, \ldots, x_k$. The weight (or importance) of sample $x_j, j = 1, \ldots, i-1$ is $\rho_j = 0$. Sample x_i, has ρ_i close to zero, whereas the sample in the middle of the window has $\rho = 1$. One may consider any other profile (logarithmic, exponential, linear, etc.), customized for a specific problem.

3.1. Incremental learning

The idea behind incremental learning (and conversely decremental unlearning) is to keep the KT conditions fulfilled at all times, as described by Eqs. (13)–(16). We start from a configuration for which KT conditions are fulfilled. For a new sample x_c and a new characteristic value which starts with $\lambda_c = 0$, Eq. (21) will measure the penalty $h(x_c)$. If there is no penalty ($h(x_c) > 0$), it means that the KT conditions are fulfilled for the newly added x_c. Otherwise, Eqs. (28) and (30) would give the needed increment $\Delta\lambda_c$ for decreasing the penalty $h(x_c)$, without affecting the already fulfilled KT conditions of the other samples. In the following, we discuss the necessary steps for reaching this goal.

Starting from Eqs. (28) and (30), we compute the exact quantities that determine the vectors to migrate between sets. Support vectors can migrate to the rest set (if their associated λ_s decreases to zero) or to the error set (if λ_s increases up to ρC). Likewise, the rest vectors can migrate to the support set when their positive penalty cost, h_r, decreases to zero. The error vectors can migrate to the support set when their penalty cost (which is negative) increases to zero.

A vector x_c that is newly added to a set may fulfill the KT conditions in Eq. (22) without requiring any processing. Otherwise, it becomes subject of the learning process and its λ_c, initially zero, is increased. We have to determine precisely how much λ_c should be increased before some vector is forced to change sets.

Algorithm 1 Concept drift WIDSVM learning and unlearning

procedure SHIFTINGWINDOW(data_stream)
  ▷ the data stream is considered continuous
  choose C parameter and $window\_size$
  set initial solutions using (x_1, y_1) and (x_2, y_2)
  ▷ window initialized with empty list
  $window \leftarrow \varnothing$
  while incoming data samples exist **do**
    $(x_k, y_k) \leftarrow$ next incoming sample
    extend kernel with (x_k, y_k)
    for samples x_j on descending profile
      ▷ x_k is the last sample of this profile
      $\rho_j \leftarrow \rho_j + \rho_{\text{step}}$
      LEARN(x_j, $\rho_j C$)
    if size($window$) > $window\_size$ **then**
      for samples x_i on ascending profile
        $\rho_i \leftarrow \rho_i - \rho_{\text{step}}$
        UNLEARN(x_i, $\rho_i C$)
      ▷ remove first sample in the window
      if λ_{i_1} reached 0 **then**
        remove sample x_{i_1} from window
    collect statistics for next unseen vectors

procedure LEARN(x_a, C_a)
  while x_a not yet learned **do**
    $Q \leftarrow$ compute_Q(kernel, y)
    $\beta_s \leftarrow$ compute_beta(Q, y)
    $\gamma_s \leftarrow$ compute_gamma(Q, y, β_s)
    $\Delta l_s \leftarrow$ compute_limits_for_support_vectors(x_a, C_a)
    $\Delta l_r \leftarrow$ compute_limits_for_rest_vectors(x_a, C_a)
    $\Delta\lambda_a \leftarrow \min\{\Delta l_s, \Delta l_r, C_a - \lambda_a\}$
    $\lambda_{j \neq a} \leftarrow \lambda_{j \neq a} + \beta_s \Delta l_a$
    $\lambda_a \leftarrow \lambda_a + \Delta\lambda_a$
    rearrange_vectors_in_sets()

procedure UNLEARN(x_a, C_a)
  while x_a not yet unlearned **do**
    if x_a removal leaves its class unrepresented **then**
      return
    $Q \leftarrow$ compute_Q(kernel, y)
    $\beta_s \leftarrow$ compute_beta(Q, y)
    $\gamma_s \leftarrow$ compute_gamma(Q, y, β_s)
    $\Delta l_s \leftarrow$ compute_limits_for_support_vectors(x_a, C_a)
    $\Delta l_r \leftarrow$ compute_limits_for_rest_vectors(x_a, C_a)
    $\Delta\lambda_a \leftarrow \max\{\Delta l_s, \Delta l_r, C_a - \lambda_a\}$
    $\lambda_{j \neq a} \leftarrow \lambda_{j \neq a} + \beta_s \Delta l_a$
    $\lambda_a \leftarrow \lambda_a + \Delta\lambda_a$
    rearrange_vectors_in_sets()

We consider the incremental case where $\Delta\lambda_c > 0$ (λ_c grows from zero).

Support vectors migrate if we take $\Delta\lambda_s = \beta_s \cdot \Delta\lambda_c$:

- For $\beta_s > 0$, $\Delta\lambda_s$ may grow up to ρC, and the first vector that migrates is given by $\Delta\lambda_c^{\text{support}_1} = \min_s \left\{ \frac{\rho C - \lambda_s}{\beta_s} \right\}$; the support vector migrates to the error set;
- For $\beta_s < 0$, $\Delta\lambda_s$ may decrease to 0, and the first support vector that migrates is given by $\Delta\lambda_c^{\text{support}_2} = \min_s \left\{ -\frac{\lambda_s}{\beta_s} \right\}$; the support vector migrates to the rest set.

Error and rest vectors migrate if we take $\Delta h_r = \gamma_r \cdot \Delta\lambda_c$:

- This scenario happens either for a rest vector ($h_r > 0$), when the corresponding $\gamma_r < 0$ and h_r decreases; or for an error vector ($h_r < 0$), when $\gamma_r > 0$ and thus h_r increases. In both cases, the first rest/error vector to migrate is the vector given by the same relation, $\Delta\lambda_c^{\text{rest}} = \min_r \left\{ -\frac{h_r}{\gamma_r} \right\}$. The rest/error vector migrates to the support set.
- We note that if a rest vector ($h_r > 0$) has its associated $\gamma_r > 0$, having a positive $\Delta\lambda_c$ only increases h_r further; this vector does not migrate. The same happens for an error vector with $\gamma_r < 0$ - its h_r only decreases further. In this case no further checking is necessary.

We should not increment λ_c past ρC (Eq. (37)). An additional constraint is that $\Delta\lambda_c^{\text{margin}} \leq \rho C - \lambda_c$.

The maximum allowed update is computed as:

$$\Delta\lambda_c = \min \left\{ \Delta\lambda_c^{\text{support}_1}, \Delta\lambda_c^{\text{support}_2}, \Delta\lambda_c^{\text{rest}}, \Delta\lambda_c^{\text{margin}} \right\}$$

3.2. Decremental unlearning

In this case, we want to remove a vector x_c that is part of the solution. If its corresponding $\lambda_c = 0$, the removal is straightforward. However, if $\lambda_c \neq 0$, we first must decrease λ_c to zero before removing the vector. Therefore, $\Delta\lambda_c < 0$.

Support vectors migrate if we take $\Delta\lambda_s = \beta_s \cdot \Delta\lambda_c$:

- For $\beta_s > 0$, $\Delta\lambda_s$ may decrease to 0, and the first vector to migrate is given by $\Delta\lambda_c^{\text{support}_1} = \max_s \left\{ -\frac{\lambda_s}{\beta_s} \right\}$; the support vector migrates to the rest set.
- For $\beta_s < 0$, $\Delta\lambda_s$ may grow up to ρC, and the first support vector to migrate is given by $\Delta\lambda_c^{\text{support}_2} = \max_s \left\{ \frac{\rho C - \lambda_s}{\beta_s} \right\}$; the support vector migrates to the error set.

Error and rest vectors migrate if we take $\Delta h_r = \gamma_r \cdot \Delta\lambda_c$:

- This scenario happens either for a rest vector ($h_r > 0$), when corresponding $\gamma_r > 0$ and h_r decreases; or for an error vector ($h_r < 0$), when $\gamma_r < 0$ and thus h_r increases. In both cases, the first rest or error vector to migrate is the vector given by the relation $\Delta\lambda_c^{\text{rest}} = \max_r \left\{ -\frac{h_r}{\gamma_r} \right\}$. The rest/error vector migrates to the support set.
- Identically, we note that if a rest vector ($h_r > 0$) has $\gamma_r < 0$, having a negative $\Delta\lambda_c$ only increases h_r further; this vector does not migrate. The same happens for an error vector with $\gamma_r > 0$ - its h_r only decreases further. In this scenario, no further checking is necessary.

The target is to decrease λ_c to the new ρC value (which can be even 0). The additional condition is that $\Delta\lambda_c^{\text{margin}} \geq \rho C - \lambda_c$.

Thus, the minimum allowed update before any migration takes place (all these values being negative) is

$$\Delta\lambda_c = \max \left\{ \Delta\lambda_c^{\text{support}_1}, \Delta\lambda_c^{\text{support}_2}, \Delta\lambda_c^{\text{rest}}, \Delta\lambda_c^{\text{margin}} \right\}$$

We change the λ_c by migrating the vectors among sets, step by step, and thus we keep all KT conditions satisfied (except for the sample x_c in question).

3.3. Detecting training data inconsistencies

During training, it may happen that certain data features cause undetermined conditions and computation cannot be performed. One such issue may occur when computing the matrix inverse in Eq. (27). We have to skip learning samples that are linear combinations of already trained samples within the shifting window, otherwise the matrix becomes singular.

Another situation that impedes training is when $\gamma_c = 0$ in Eq. (30). We have to avoid the γ_c value of the newly introduced vector becoming zero (or very close to zero), because learning is not possible in this case. For every new vector x_c, following Eqs. (27) and (29), we compute γ_c and test it against zero:

$$\gamma_c = -\begin{bmatrix} y_c & Q_{sc}^T \end{bmatrix} \begin{bmatrix} 0 & y_s \\ y_s^T & Q_{ss} \end{bmatrix}^{-1} \begin{bmatrix} y_c \\ Q_{sc} \end{bmatrix} + Q_{cc} \neq 0 \tag{38}$$

If $\gamma_c = 0$ or very close to zero, the vector will be discarded.

3.4. Initial solution

The incremental learning process is initialized with two initial vectors x_1 and x_2 that belong to different classes, such that besides $y_{1,2} \in \{-1, +1\}$ we have $y_1 \cdot y_2 = -1$. These two vectors are considered to be the first support vectors. Hence, their penalty cost is zero:

$$0 = h(x_i) = \sum_{j=1}^{2} \lambda_j Q_{ij} + y_i b - 1 \qquad i = 1, 2 \tag{39}$$

By solving this system of equations, the expressions for λ_i and b are computed as:

$$\lambda_1 = \lambda_2 = \frac{2}{Q_{11} + 2Q_{12} + Q_{22}} \tag{40}$$

and

$$b = \frac{Q_{11} - Q_{22}}{Q_{11} + 2Q_{12} + Q_{22}} \cdot y_2 \tag{41}$$

The initial value for the regularization constant C is chosen such that $\lambda_{1,2} \leq C$.

3.5. Computational complexity

The incremental–decremental SVM solution is computed only from the samples inside the current shifting window. Let N be the window size. The ShiftingWindow procedure, as described in Algorithm 1, calls the Learn/Unlearn procedures. Both procedures follow these steps:

1. a test to establish whether the associated λ_a is within the allowed limits, $0 \leq \lambda_a \leq \rho C$, while testing whether the penalty h_a has either reached zero (due to x_a migrating to support set) or a positive or negative value (due to migration to the rest/error sets);
2. computation of Q, as described by Eq. (19), which is in $O(N^3)$;
3. computation of β_s, given by Eq. (27), is based on matrix inversion, so it is in $O({n_{SV}}^3)$, where $n_{SV} \leq N$ is the number of support vectors;
4. the computation of γ_s is in $O({n_{SV}}^2)$ as given by Eq. (29);
5. procedure **compute_limits_for_support_vectors()** is in $O(n_{SV})$, as it computes the maximum/minimum for $\Delta\lambda$ values associated to support vectors;

6. procedure **compute_limits_for_rest_vectors()** is in $O(N^2)$, since it involves computing the penalties h for all vectors;
7. procedure **rearrange_vectors_in_sets()** has linear time.

Overall, the internal loop of the learn/unlearn procedures is in $O(N^3)$. At each iteration of the inner loop, the value of λ_a approaches C_a for learning, or zero for unlearning. The number of loops depends on the dataset. The algorithm will eventually converge, since with each iteration we are getting closer to satisfying the Kuhn–Tucker condition for x_a, as described in Cristianini and Shawe-Taylor (2000). One can observe that, for a large absolute negative value of the penalty h_a, the KT condition is not fulfilled. However, in case of initial h_a being positive, the inner loop is not executed, since Eq. (15) is verified from the start ($\lambda_a = 0$).

4. Experiments

We describe a series of experiments performed on artificial and real-world datasets which exhibit concept drift. First, we compare the performance (classification accuracy) of WIDSVM with a standard SVM (without shifting window). Our baseline is the C-Support Vector Classification SVM implementation from the Scikit-Learn library.[1] We refer to it as the Classic SVM (C-SVM). Next, we compare the training times of the WIDSVM and the C-SVM retrained from scratch. Finally, we compare the performance of WIDSVM with other concept drift models available in the literature.

4.1. Datasets with concept drift

The most popular synthetic and real-world benchmark dataset for testing concept drift handling models are (Iwashita & Papa, 2019): KDDCUP'99, STAGGER, Electricity, Hyperplane, SEA, Gauss, and Forest Covertype. An important problem with most of the real-world benchmark datasets is that there is little concept drift in them, or concept drift is introduced artificially (Gama et al., 2014; Tsymbal, 2004). In addition, some of these datasets are corrupted.

KDDCUP'99 is the most widely used dataset for the evaluation of concept drift systems (Iwashita & Papa, 2019). It contains computer network intrusion data, classifying legitimate and illegitimate connections. There are 23 class labels; we chose to test the "smurf" and "normal" classes, which are the most represented and account for about 80% of the dataset. We trained the classic SVM on the first 9994 samples (2.6%), where we found 7787 normal and 2207 smurf instances and tested on other 368,074 (97.4%). We obtained a 99.99% classification accuracy (only 48 misclassified patterns). This nearly 100% accuracy is strong evidence that the concept drift in the stream, if present, is irrelevant. The analysis performed by Tavallaee, Bagheri, Lu, and Ghorbani (2009) leads to a similar conclusion: they employed 21 learned machines (7 learners, each trained 3 times with different train sets) and labeled the records of the entire KDD train and test sets. They showed that about 98% of the records in the train set and 86% of the test set were correctly classified with all the 21 learners. For these reasons, we skip KDDCUP'99 in our experiments.

The Electricity dataset is a popular benchmark for testing concept drift models (Harries, 1999). The dataset covers a period of two years of recordings, instances recorded every half hour, for 7 instance variables, one being a categorical variable (day of the week). The dataset has 45,312 instances. The task is to predict an increase (UP) or a decrease (DOWN) for the price of electricity in New South Wales, Australia. The prior probability for price increase is 58%. The price is subject to change due to consumption habits, seasonality or unexpected events. Six of the variables are normalized (Centre for Open Software Innovation, 2019) and for the seventh (the day) we used one-hot encoding, to express the day-of-week seasonality.

The Forest Covertype dataset (Blackard & Dean, 2000) describes the forest coverage type for 30 × 30 m cells, provided by the US Forest Service (USFS). It consists of 581,012 instances and 54 attributes, not counting the class type. Out of these, only 10 are continuous, and the rest of them (such as wilderness area and soil type) are one-hot encoded. The set defines seven classes; we only used the two most represented classes, with a total of 495,141 data samples. The classes are equally balanced: 211,840 in class 1 vs. 283,301 in class 2. The dataset was already normalized (Centre for Open Software Innovation, 2019). For the SVM to work properly, we detected the sets of temporally consecutive data samples belonging to the same class. We observed that, apart from a set of 5692 consecutive elements of the same class, which was skipped, all other such sequences had less than 300 elements. For those long sequences, we switched the middle element with the most recent element of a different class, in order to ensure that none are longer than 150 samples from the same class. This is similar with SVM keeping its latest sample of the opposite class, in its definition of the hyperplane.

The Circles dataset (Pesaranghader, Viktor, & Paquet, 2018) is a synthetic dataset with gradual induced concept drift. It has two attributes, distributed on the [0, 1] interval. The classification function is a circle of predefined radius and center. Points that belong to the interior of the circle are classified as belonging to the first class, whereas the exterior points belong to the second class. The drift happens when the definition of the circle function changes. This happens every 25,000 samples.

The Sine1 dataset (Pesaranghader et al., 2018) is another synthetic dataset, with abrupt concept drift. There are two continuous attributes uniformly distributed on the [0, 1] interval. The classification function is $y = sin(x)$: instances are classified as positive if they are under the curve; otherwise they are classified as negative. Concept drift occurs by reversing the class labels, happening at every 20,000 samples.

We studied the STAGGER dataset but we did not use it in our experiments because of its nature. There are only three categorical attributes for a sample, making the transitions rather abrupt as the concept shifts too rapidly. This is out of our focus.

4.2. Implementation details and experimental setup

Experiments were run on a Dell Tower platform with Intel(R) Xeon(R) W-2145 CPU @ 3.70 GHz. Our WIDSVM code is implemented in PyTorch and available on CodeOcean.[2]

In our implementation, the matrix operations run in a tensor-optimized form. To speed up the execution, we use caching to pre-calculate the kernel values, so the update of the kernel values in Eq. (19) is performed in linear time, avoiding $O(N^2)$ kernel recalculation for each new sample.

At the core of the algorithm lies the matrix inversion in Eq. (27), which must be reconstructed every time a vector migrates in/out of the support set. The support set size directly determines the dimension of the inverted matrix. By increasing the number of samples within the shifting window, the number of support vectors increases, though this is data dependent.

During training, we detect the training vectors that lead to rank reduction of matrix Q in Eq. (19) by computing the penalty introduced by a new vector $h(x_c)$ and testing for equality with the previous terms, computed in Eq. (21).

[1] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[2] https://doi.org/10.24433/CO.2865125.v1.

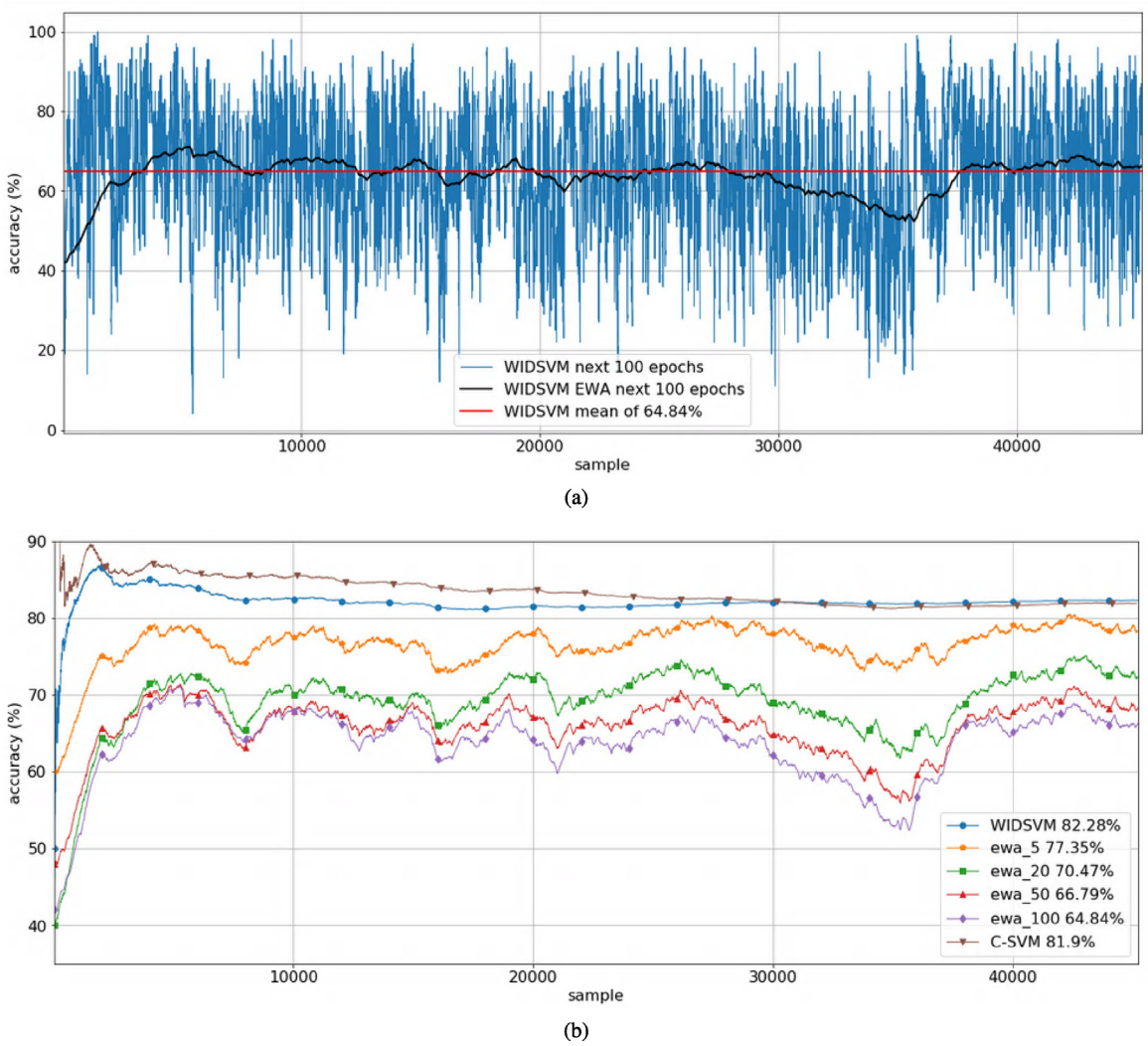

Fig. 2. Experiments on the Electricity dataset. (a) Accuracy of WIDSVM tested on the next 100 samples after the shifting window is shown together with the exponentially-weighted average (EWA) and the global mean. (b) The mean accuracy assessed on the next sample after the shifting window (WIDSVM, shown in blue) reaches 82.28% on average. The EWA accuracy is assessed on the 5, 20, 50 and 100 next samples. It is visible that performance degrades in time. Classic SVM (C-SVM) accuracy is close. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The terms λ change during training, within the interval described by Eq. (37). Because they are determined by Eqs. (28) and (30), even if by these computations they are supposed to be zero or ρC, they do not reach these exact values. In practice, we relax the condition from Eq. (37) by testing that each λ_i is within limits:

$$\lambda_i = \epsilon \quad \text{or} \quad \lambda_i = \rho C - \epsilon \tag{42}$$

We also consider the penalty of support vectors to be very close to zero, so condition (22) is written as:

$$h(x_{\text{support}}) = \epsilon \tag{43}$$

We found $\epsilon = 10^{-10}$ to be an acceptable tolerance value to prevent numerical stability problems. During training, it is possible that, for a small window, all samples are from the same class. To prevent this, we ensure that at least one sample of each class is represented in the window.

We use the exponential kernel function in Eq. (19), $K(x_i, x_j) = e^{-s\|x_i - x_j\|^2}$, and the optimal s is $s_{optimal} = 1/(N \cdot Var[X])$. For the Electricity dataset, we determined the optimal value for s using the first 6750 samples (approx. 15% of the whole set). Similarly, for Forest Covertype, we determined this value using the first 4000 samples (less than 1%). For Circles we considered the first 10 000 samples. The initial incremental SVM solution for WIDSVM determines the appropriate values for λ_1 and λ_2, which dictate the lower bound of the regularization constant C. During learning, the WIDSVM classifier is trained on a fixed size window of samples. We determined the window size by trial and error, trying to optimize the classifier accuracy by training on a window of samples and testing on the next samples following the window. The window size influences the time complexity, since it determines the maximum size of the inverse in Eq. (27), and also the size of the h penalty that is repeatedly computed in Eq. (30) until the solution fulfills the KT conditions. The hyper-parameters are presented in Table 1.

Table 1
Hyper-parameters used for each of the datasets.

Dataset	Window size	C	s
Electricity	155	100	0.8364
Forest Covertype	1000	100	0.2415
Circles	1000	1	7.7977
Sine1	100	10	5.9913

We used three types of weight profiles for the shifting window: trapezoidal, exponential and abrupt step. These profiles

were chosen for no particular reason, just as an illustration of the concept. We wanted to prove that WIDSVM is able to learn samples with different associated weights. We used the following trapezoidal shape: $\rho_1 = 0.5$, $\rho_2 = \rho_3 \cdots = \rho_{N-1} = 1$, $\rho_N = 0.5$, for a window size of N samples. This particular profile is not optimized for a specific dataset or problem; however, in practice we found that for the chosen datasets, the results on this profile are practically similar with those using the step profile, due to the large size of the shifting window.

4.3. Results

We set a window size of 155 samples for the Electricity dataset and, as the window advances, we classify the next 100 instances in the stream after the shifting window, using the model trained on the current window. We show the next 100 instances classification result after learning each data sample in the set. Fig. 2(a) shows, in blue line, the accuracy of the model when tested on the next 100 samples, as the window shifts. We count how many of these next 100 samples are correctly classified. We represent the exponentially weighted moving average (EWA) of this quantity and the mean accuracy computed for all data samples (64.84%). In Fig. 2(b), we compute in the same manner the accuracy on a set of next 5, 20, 50 and 100 samples. We represent the exponentially weighted moving average (EWA) accuracy. The EWA accuracy V_t on sample t is computed as $V_t = \beta V_{t-1} + (1-\beta)A_t$, where A_t is the instant accuracy on sample t. In our experiments, we take $\beta = 0.9995$, equivalent to a weighted average for about the last 2000 samples. For testing on the next sample only, we use a different technique. Because representing the accuracy in this case would only generate two possible values (0% and 100%), we compute the ratio of correctly classified samples so far. As a result, the "WIDSVM" line in Fig. 2(b) would converge to its mean value as the number of samples increases. The mean accuracy of this profile (its last value) reaches 82.28%. To emphasize the trends of the graphs, we added solid markers every 2000 samples. The performance of the classic SVM (C-SVM), trained using the same shifting window, is shown for comparison. Here, its performance of 81.9% is not far from the performance of WIDSVM. However, the C-SVM is retrained from scratch every time the window shifts by one sample. There are some known issues with the Electricity dataset (Žliobaite, 2013). It has long sequences of samples belonging to the same class. For example, if we construct a classifier that always predicts the class of the next sample to always be the class of the current sample (the moving average of one), then for this particular dataset the obtained accuracy is 85.3%. If the data were independently distributed, the moving average of one predictor would give an accuracy of 51%. Also, the seasonality of the data is very strong, the autocorrelation function peaking every 24 h. In our experiments, we omit the day of year feature, since this is steadily increasing.

Fig. 3 shows the performance of the WIDSVM on the Forest Covertype dataset. The exponential weighted average of the accuracy indicates a better performance for the first half, and a poorer performance on the second half, followed by a quick recovery in the end. The performance of WIDSVM tested only on the next sample is on average 92.28%. The performance on the next 5, 20, 50 and 100 samples following the shift window is also presented. As expected, due to the concept drift, the accuracy is higher when testing on nearby samples, rather than on the farther ones.

We calculated the accuracies for different training profiles on the Forest Covertype dataset in Fig. 4. We used the abrupt step, trapezoidal and exponential profiles (Fig. 4(a)). In Fig. 4(b) we show both the performance of WIDSVM when tested only on the next sample, as well as when tested on the next 20 samples in the stream. The window size is 2000 samples, whereas the profile slopes are defined by only the first and last 10 samples in the window size. The results show that the best performing profile is the step profile, indicating, as expected, that for concept drift, the most relevant samples for classification seem to be the most recent ones. The difference in accuracy between the step profile and the exponential profile (which puts less emphasis on newly trained samples) is on average 1.3%.

Table 2
Accuracy (in percents) for training C-SVM and WIDSVM on different window sizes, given the datasets.

Electricity	Window size							
	100	125	155	175	200	300	500	1000
C-SVM	81.86	81.83	81.84	81.98	81.94	81.57	81.74	80.31
WIDSVM	81.77	81.98	82.24	82.36	82.54	81.43	81.50	79.88

Circles	Window size					
	50	100	200	300	500	1000
C-SVM	83.87	85.55	86.44	86.81	87.06	87.15
WIDSVM	83.88	85.51	86.43	86.15	87.04	87.06

Sine1	Window size					
	50	100	200	300	500	1000
C-SVM	83.14	85.55	86.88	87.24	87.45	87.10
WIDSVM	83.06	85.57	86.88	87.25	87.30	86.44

Covertype	Window size					
	150	200	300	500	1000	1500
C-SVM	91.13	90.81	92.25	93.85	94.38	93.05
WIDSVM	90.86	89.27	91.54	94.17	94.14	92.21

Fig. 5 shows the performance of the WIDSVM on the synthetic Circles dataset. Here we used a window size of 100 samples. The performance is around 87.14%, independent of the window size, up to a size of 50 samples inspected in advance. It is close to the 85.56% performance of the C-SVM trained on a shifting window. The difference is explained by the incremental–decremental process, which discards some of vectors early due to detection of singularity in inversion matrix of Eq. (27) or condition (38). We found out that the number of vectors composing the solution is slightly different in the two implementations, supporting our hypothesis.

The same behavior of sudden accuracy drop was found on the Sine1 dataset. While the shifting window is able to cope with slow drift to a certain degree, it fails to adapt to fast concept drift. We tackled this problem in a different work (Gâlmeanu & Andonie, 2021).

4.4. Comparison with classic SVM

In Table 2 we compare the performance of WIDSVM against the classic SVM. Tests were made on different datasets, for different window sizes, computing the accuracy on the sample following the shifting window. For the Electricity dataset, the best window size is 175 for C-SVM, whereas for WIDSVM it is 200, which is close. For Circles dataset, the optimum window size seems to be 1000 for both methods. The same goes for Sine1 (with window size of 500), whereas for the Covertype dataset, the optimum window size is 1000 for C-SVM, and 500 for WIDSVM. These experiments suggest that the behavior of the two classifiers is similar, which is expected; both construct the optimal separation hyperplane. The WIDSVM however, by design, removes some of the samples with contribution too similar with the existing ones, avoiding γ close to zero in Eq. (38). This explains the reason why we do not observe identical results, but most similar ones.

To see how accuracy varies for C-SVM and WIDSVM for test sizes, we performed another set of experiments (see Table 3). The results were computed by averaging performances for every

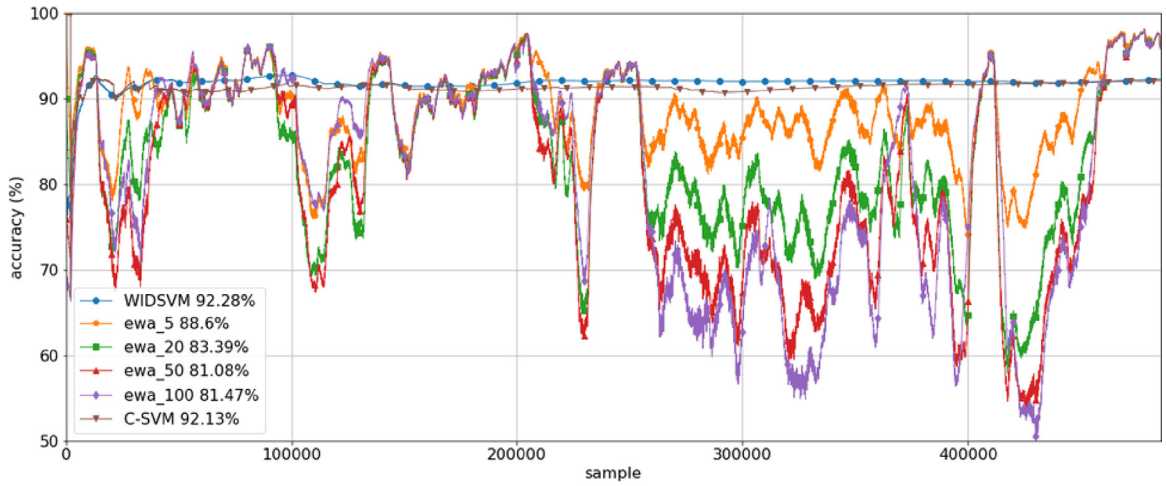

Fig. 3. Experiments on the Forest Covertype dataset. The accuracy assessed on the next sample after the shifting window (WIDSVM) reaches 92.28% on average. The EWA accuracy is assessed on the 5, 20, 50 and 100 next samples. The C-SVM accuracy is close to WIDSVM accuracy. This is explained by the fact that WIDSVM may be forced to discard samples that trigger condition number issues for matrix Q in Eq. (27), as observed experimentally.

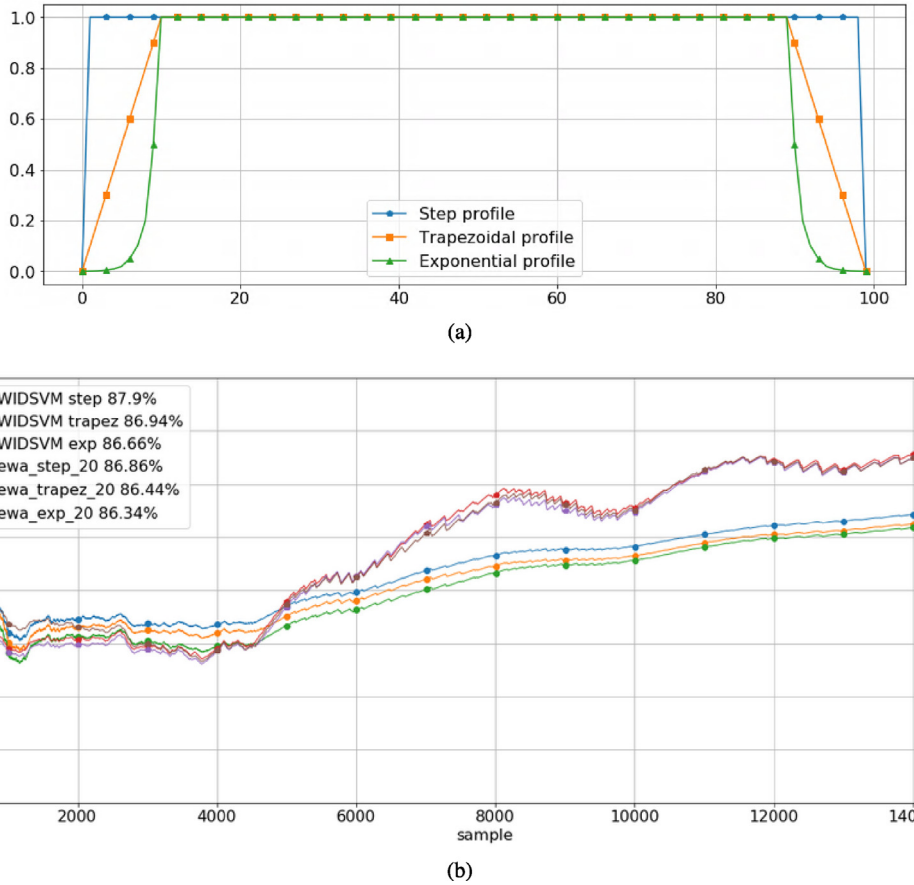

Fig. 4. Experiments with learning profiles. (a) Weight profiles used for WIDSVM, for a window size of 100. (b) Accuracies obtained for different weight profiles on the Forest Covertype dataset, with a fixed window size of 2000 samples. We present the accuracy for testing on the next sample as well as on the next 20 samples.

shift position of the sliding window. We observe that the accuracy decreases as the test set becomes farther extended in the future. This behavior is natural since, as we test farther in the future, the underlying distribution of the data changes due to drift, and farther samples become harder to predict. Also, although on small set sizes the C-SVM and WIDSVM results are comparable, as set size increases, WIDSVM adapts better to drift, due to its learning profile properties.

4.5. Training time

Comparing directly the execution time of WIDSVM and C-SVM is not very precise, since WIDSVM has a custom implementation

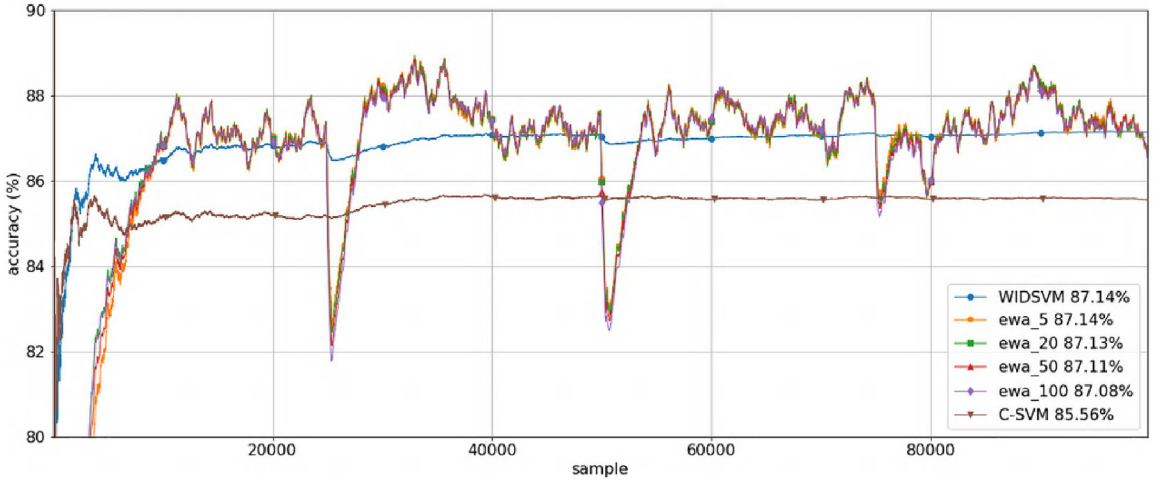

Fig. 5. Experiments with the Circles dataset. The concept drifts appears clearly at samples 25,000, 50,000 and 75,0000.

Table 3
Accuracy (in percents) for testing C-SVM and WIDSVM on different set sizes following the shifting window.

Electricity	Window size				
	1	5	20	50	100
C-SVM	81.90	73.44	68.93	64.05	62.64
WIDSVM	82.28	77.35	70.47	66.79	64.84
Covertype	Window size				
	1	5	20	50	100
C-SVM	92.13	90.81	81.29	76.81	74.22
WIDSVM	92.28	88.60	83.39	81.08	81.47

Table 4
Average training time per sample, in milliseconds, for C-SVM vs. WIDSVM, on different window sizes.

Method	Window size							
	100	200	500	1000	1200	1500	1700	2000
C-SVM	0.7	1.7	8.6	37.3	52.2	75.8	98.5	136.6
WIDSVM	6.7	10.3	12.9	15.5	17.0	24.4	27.3	32.9

and C-SVM is based on scikit-learn. For informative purposes, we determined the training times for the WIDSVM and C-SVM algorithms on the Forest Covertype dataset. The purpose was to depict the improvements when increasing the window size. We used the same training samples for both classifiers. We started from the beginning and trained up to the point when the shifting window "fills up". For C-SVM, we retrained from scratch after each window shift. In contrast, WIDSVM uses the incremental–decremental approach. For each shift step, we measured the training time. We performed a number of steps equal to the window size; thus, the window shifts until the last window no longer contains samples from the first window. The average time was computed considering *window_size* steps. The results are presented in Table 4, in ms.

We observe that for smaller window sizes, the training time is slightly greater than the one achieved by retraining from scratch. The advantage becomes significant as the size of the sliding window grows. We also observe that the matrix inversion in Eq. (27) accounts for about 10% of training time. Further improvements are possible, by employing the Sherman–Morrison–Woodbury formula (Laskov et al., 2006).

By performing the fitting of a regression line on the log–log plot of the training time versus the training size, we find an empirical scaling for both the C-SVM as well as for the WIDSVM.

Table 5
Performance comparison of different concept drift models.

Dataset	Model			
	Naive Bayes	Very Fast Decision Tree	C-SVM	WIDSVM
Electricity	73.64%	77.93%	81.90%	**82.28**%
Forest Covertype	64.63%	86.90%	92.13%	**92.21**%
Circles	72.06%	82.09%	85.55%	**87.14**%
Sine1	57.21%	58.33%	**85.59**%	85.28%

C-SVM training time is found in the order of $O(N^{1.9})$, as expected; it is the same as the one described in the original SMO paper (Platt, 1998). According to our experiments, WIDSVM training time is in $O(N^{1.7})$.

4.6. Performance comparison with other concept drift models

Finally, Table 5 compares the WIDSVM accuracy results with the performances of other concept drift models, on the selected datasets.

We compared our model with Naive Bayes (NB) and Very Fast Decision Tree (VFDT, or Hoeffding Tree Classifier) classifiers implemented in Scikit-Multiflow framework (Montiel, Read, Bifet, & Abdessalem, 2018). Both classifiers are designed to support streaming data, performing a so-called prequential (Montiel et al., 2018) training: they replace the batch training mode, and use every sample to test the performance before considering it as training data. NB classifier provided poor performance; however VFDT is better adapted to concept drift; it grows an alternate subtree whenever the old one becomes out-of-date (Hulten, Spencer, & Domingos, 2001).

NB and VFDT do not use shifting window. For a fair comparison, we also trained them on a shifting window, and tested them using the next sample after the shifting window. However, the results observed were much worse than the previous prequential training mode — the best average performance is around 60%, considering all four datasets.

Overall, WIDSVM performed better than NB and VFDT in all cases. Its behavior is comparable to the C-SVM trained on the same shifting window, having the advantage that it does not have to be retrained from scratch.

5. Conclusions and open problems

We introduced the WIDSVM algorithm, a generalization of the incremental–decremental SVM for concept drift with shifting window. WIDSVM enables the incremental SVM to learn

the newly arrived data samples while it forgets the older ones, both operations being controlled by the weights inside a shifting window applied to the data input stream. Both learning and forgetting can operate in a gradual manner.

The main difficulty was to constrain the WIDSVM to fulfill the SW Property. Thus, when adding/removing samples, for the new sample we increase the allowed characteristic λ value, whereas we remove the obsolete sample by decreasing its allowed characteristic value. This is accomplished by the incremental and decremental procedures that construct the new solution as a difference from the previous one. Evolving from the CP method that gives the general bookkeeping conditions, WIDSVM defines the maximum increment (or decrement) that determines the next vector migration while keeping the Kuhn–Tucker conditions satisfied for the existing samples and reducing the penalty cost for the new sample. During this process, vectors may migrate among sets, so we computed the exact conditions for the migrations. The system changes with each vector migration, until it reaches the KT equilibrium again. Using weighted constraints for the vectors inside the shifting window, the KT conditions are fulfilled by repeatedly applying the CP algorithm. This is much faster than the traditional SVM since we do not relearn the already learned samples.

While the SW property is intuitively best characterized by a rectangular profile, it can also be extended to more general weight profiles. For instance, by shifting the window and associated instance weights, there is no need to retrain from scratch on barely modified new input samples. In this case, for the new context, it is more convenient to update the solution until the Kuhn–Tucker conditions are met.

We used a fixed size window, determined a priori. This can work well if information on the time-scale of change is available. However, this is rarely the case (Bifet & Gavaldà, 2007) and it is difficult to determine a priori an optimal window size. Using an adaptive windowing is very appealing and was investigated by several authors. For instance, we could automatically grow the window when no change is apparent, and shrink it when data changes, the strategy used in Bifet and Gavaldà (2007). We recently introduced such an incremental–decremental SVM model, capable of dynamically detecting the concept drift and adjusting the size of the shifting window (Gâlmeanu & Andonie, 2021).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. In *SDM'07: Vol. 7, Proceedings of the seventh SIAM international conference on data mining, Minneapolis, USA* (p. 6). http://dx.doi.org/10.1137/1.9781611972771.42.

Blackard, J., & Dean, D. (2000). Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, *24*(3), 131–151.

Burges, C., & Chrisp, D. (1999). Uniqueness of the SVM solution. *NIPS*, *99*.

Carpenter, G. A., & Grossberg, S. (1988). The ART of adaptive pattern recognition by a self-organizing neural network. *Computer*, *21*(3), 77–88.

Cauwenberghs, G., & Poggio, T. (2000). Incremental and decremental support vector machine learning. In *NIPS'00, Proceedings of the 13th international conference on neural information processing systems* (pp. 388–394). Cambridge, MA, USA: MIT Press.

Centre for Open Software Innovation, T. U. o. W. (2019). Datasets - MOA. https://moa.cms.waikato.ac.nz/datasets/ (accessed May 1, 2020).

Chang, X., Yu, Y., Yang, Y., & Xing, E. (2017). Semantic pooling for complex event analysis in untrimmed videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(8), 1617–1632.

Chen, Y., Xiong, J., Xu, W., & Zuo, J. (2019). A novel online incremental and decremental learning algorithm based on variable support vector machine. *Cluster Computing*, *22*, 11. http://dx.doi.org/10.1007/s10586-018-1772-4.

Chitrakar, R., & Huang, C. (2014). Selection of candidate support vectors in incremental SVM for network intrusion detection. *Computers and Security*, *45*, 231–241. http://dx.doi.org/10.1016/j.cose.2014.06.006.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods* (1st ed.). Cambridge University Press.

Diehl, C. P., & Cauwenberghs, G. (2003). SVM incremental learning, adaptation and optimization. In *IJCNN'03: Vol. 4, Proceedings of the international joint conference on neural networks, 2003* (pp. 2685–2690).

Elwell, R., & Polikar, R. (2009). Incremental learning in nonstationary environments with controlled forgetting. In *2009 international joint conference on neural networks* (pp. 771–778).

Elwell, R., & Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, *22*(10), 1517–1531.

Farid, D. M., Zhang, L., Hossain, A., Rahman, C. M., Strachan, R., Sexton, G., et al. (2013). An adaptive ensemble classifier for mining concept drifting data streams. *Expert Systems with Applications*, *40*(15), 5895–5906.

Gâlmeanu, H., & Andonie, R. (2008). Implementation issues of an incremental and decremental SVM. In *ICANN '08, Proceedings of the 18th international conference on artificial neural networks, part I* (pp. 325–335). Berlin, Heidelberg: Springer-Verlag, http://dx.doi.org/10.1007/978-3-540-87536-9.34.

Gâlmeanu, H., & Andonie, R. (2009). A multi-class incremental and decremental SVM approach using adaptive directed acyclic graphs. In *2009 international conference on adaptive and intelligent systems* (pp. 114–119).

Gâlmeanu, H., & Andonie, R. (2021). Concept drift adaptation with incremental–decremental SVM. *Applied Sciences*, *20*, 97–106. http://dx.doi.org/10.3390/app11209644.

Gâlmeanu, H., Sasu, L. M., & Andonie, R. (2016). Incremental and decremental SVM for regression. *International Journal of Computers Communications & Control*, *11*(6), 755–775. http://dx.doi.org/10.15837/ijccc.2016.6.2744, URL http://univagora.ro/jour/index.php/ijccc/article/view/2744.

Gama, J. (2010). *Knowledge discovery from data streams* (1st ed.). Chapman & Hall/CRC.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, *46*(4), 1–37.

Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., et al. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning*, *106*(9–10), 1469–1495. http://dx.doi.org/10.1007/s10994-017-5642-8.

Harries, M. (1999). *Splice-2 comparative evaluation: Electricity pricing: Tech. rep.*, University of New South Wales.

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. In *KDD'01: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 97–106). http://dx.doi.org/10.1145/502512.502529.

Iwashita, A. S., & Papa, J. P. (2019). An overview on concept drift learning. *IEEE Access*, *7*, 1532–1547.

Klinkenberg, R. (2004). Learning drifting concepts: Example selection vs. Example weighting. *Intelligent Data Analysis*, *8*(3), 281–300.

Klinkenberg, R., & Joachims, T. (2000). Detecting concept drift with support vector machines. In *ICML, ICML '00: Proceedings of the seventeenth international conference on machine learning* (pp. 487–494).

Laskov, P., Gehl, C., Krüger, S., & Müller, K.-R. (2006). Incremental support vector learning: Analysis, implementation and applications. *Journal of Machine Learning Research*, *7*, 1909–1936.

Lazarescu, M. M., Venkatesh, S., & Bui, H. H. (2004). Using multiple windows to track concept drift. *Intelligent Data Analysis*, *8*(1), 29–59.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2019). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, *31*(12), 2346–2363.

Ma, J., Theiler, J., & Perkins, S. (2003). Accurate on-line support vector regression. *Neural Computation*, *15*(11), 2683–2703. http://dx.doi.org/10.1162/089976603322385117.

Ma, Y., Zhao, K., Wang, Q., & Tian, Y. (2020). Incremental cost-sensitive support vector machine with linear-exponential loss. *IEEE Access*, *8*, 149899–149914. http://dx.doi.org/10.1109/ACCESS.2020.3015954.

Martin, M. (2002). On-line support vector machine regression. In T. Elomaa, H. Mannila, & H. Toivonen (Eds.), *Machine learning: ECML 2002* (pp. 282–294). Berlin, Heidelberg: Springer Berlin Heidelberg.

Montiel, J., Read, J., Bifet, A., & Abdessalem, T. (2018). Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, *19*, 1–5, https://github.com/scikit-multiflow/scikit-multiflow.

Pesaranghader, A., Viktor, H., & Paquet, E. (2018). Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams. *Machine Learning*, *107*, 1711–1743.

Platt, J. (1998). *Sequential minimal optimization: A fast algorithm for training support vector machines*: *Tech. Rep. MSR-TR-98-14*, Microsoft, https://www.microsoft.com/en-us/research/publication/sequential-minimal-optimization-a-fast-algorithm-for-training-support-vector-machines/.

Rüping, S. (2001). Incremental learning with support vector machines. In *Proceedings 2001 IEEE international conference on data mining* (pp. 641–642).

Shen, Y., Zhu, Y., Du, J., & Chen, Y. (2018). A Fast Learn++.NSE classification algorithm based on weighted moving average. *Filomat*, *32*, 1737–1745. http://dx.doi.org/10.2298/FIL1805737S.

Syed, N. A., Liu, H., & Sung, K. K. (1999). Handling concept drifts in incremental learning with support vector machines. In *KDD '99*, *Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 317–321). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/312129.312267.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications* (pp. 1–6).

Tsymbal, A. (2004). *The problem of concept drift: definitions and related work*: *Tech. rep.*, Dublin: Department of Computer Science, Trinity College.

Voosen, P. (2019). New climate models predict a warming surge. *Science*, *16*.

Wang, X., & Xing, Y. (2019). An online support vector machine for the open-ended environment. *Expert Systems with Applications*, *120*, 72–86. http://dx.doi.org/10.1016/j.eswa.2018.10.027.

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*(1), 69–101. http://dx.doi.org/10.1023/A:1018046501280.

Yalcin, A., Erdem, Z., & Gurgen, F. (2007). Ensemble based incremental SVM classifiers for changing environments. In *2007 22nd international symposium on computer and information sciences* (pp. 1–5).

Yang, X., Song, Q., & Cao, A. (2007). Weighted support vector machine for data classification. In *Proceedings of the IEEE international joint conference on neural networks, Vol. 21* (pp. 859–864). http://dx.doi.org/10.1109/IJCNN.2005.1555965, vol. 2,

ZareMoodi, P., Siahroudi, S. K., & Beigy, H. (2016). A support vector based approach for classification beyond the learned label space in data streams. In *SAC '16*, *Proceedings of the 31st annual ACM symposium on applied computing* (pp. 910–915). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/2851613.2851652.

Žliobaite, I. (2013). How good is the electricity benchmark for evaluating concept drift adaptation. arXiv:13013524.