Online Metric Learning for an Adaptation to Confidence Drift

Lydia Fischer*†, Barbara Hammer* and Heiko Wersing†
*Bielefeld University, Universitätsstr. 25, 33615 Bielefeld, Germany
†HONDA Research Institute Europe, Carl-Legien-Str. 30, 63073 Offenbach am Main, Germany

***Abstract*—One of the main aims of lifelong learning architectures is to efficiently and reliably cope with the stability-plasticity dilemma. A viable solution of this dilemma combines a static offline classifier, which preserves ground knowledge that should be respected during training, with an incremental online learning of new or specific information encountered during use. A feasible realisation has been published lately based on intuitive distance-based classifiers using the concept of metric learning (*Fischer et al.: Combining offline and online classifiers for life-long learning (OOL), IJCNN'15*). One crucial aspect of such a system is how to combine the offline and online model. A generic approach, taken in OOL, uses a dynamic classifier selection strategy based on confidences of both classifiers. This can cause problems in the case of confidence drift, especially when the validity of the confidence estimation of the static offline classifier changes. This pitfall occurs in the context of metric learning whenever the metric tensor of the online system becomes orthogonal to the metric of the offline system, hence the respective internal data description mismatch. We propose an efficient metric learning strategy which allows an online adaptation of an invalid confidence estimation of the OOL system in case of confidence drift.**

I. Introduction

Lifelong learning has stirred quite some attention recently [1]: in many areas of every-day life, digital data are generated in a constant stream, stemming e. g. from distributed sensors in households, wearable technologies, intelligent smartphones, and smart systems in general. While classical machine learning enables us to equip data processing technologies with a suitable fundamental functionality, there is an increasing trend for the personalisation of electronic systems according to the specific needs of a user. Since these demands are not known priorly, this challenge requires lifelong model adaptation and continuous learning according to the observed data. Emerging areas which put quite some research effort into lifelong learning include autonomous robotics, autonomous driving, or assistive systems, for example [2]–[4].

While batch learning approaches still cover the majority of machine learning scenarios, humans always learn in an online setting based on data which are the result of their interaction with the environment. Thereby, humans can rely on fundamental concepts which they acquired during their lifetime; at the same time, they display an astonishing capability of rapidly integrating novel concepts and adapting to new situations. These demands correspond to two seemingly contradictory goals: guaranteeing the stability of ground concepts while enabling the flexibility to deal with a changing environment. These conflicting requirements are difficult to realise in technical systems: It is unclear how to address the so-called stability-plasticity dilemma, and, often, catastrophic forgetting of fundamental concepts can be encountered in technical realisations of adaptive models [5], [6]. Thus, basic functionality and safety-critical requirements are often hard-coded in such systems.

A number of promising approaches of how to deal with so-called concept drift has been published recently [7], [8]: concept drift refers to the fact that the probability distribution of the data is subject to change, hence a learned system becomes invalid and online adaptation is necessary. Concept drift can either affect the class conditional distribution $p(y|x)$ (real concept drift) or the input distribution $p(x)$ (virtual concept drift / covariate shift). Further, the drift can happen rapidly or gradually, requiring different strategies. Typically, rapid changes can better be taken into account by active strategies, while gradual changes are harder to detect and often accounted for by passive techniques [8]. Active methods aim for an identification of the time points where concept drift occurs, whereby they often rely on statistical tests of representative system parameters [9], [10]. Passive methods realise a continuous adaptation of the model at hand, e. g. based on online gradient methods [11]–[13]. Since such techniques face a large risk of catastrophic forgetting, popular models often combine more than one architecture, this way enabling a flexible control about which parts of the system are adapted based on the new data [14], [15]. Ensembles methods have proven as a particularly successful tool in this context, since they naturally represent a diversity of different concepts [9], [15], [16]. In this contribution, we will deal with gradual concept drift, i. e. we will focus on passive approaches and their suitable online adaptation.

We will tackle one specific setting which is of great relevance whenever safety-critical scenarios are treated or whenever there is a need to guarantee some basic functionality of the system at every time step: we assume, that a trained offline classifier is available which represents a basic model $p(y|x)$ for relevant regions of the space as mirrored by a probability distribution $p(x)$ of the offline training data. This model should remain stable while training. The goal of the system is to accompany this basic functionality with an adaptive model which can take into account new concepts, such as new classes y, or a refinement of the class conditional $p(y|x)$ for regions of the space which are not well covered in the offline model (i. e. $p(x)$ is small for the offline model but it becomes large for the online data). Note that, unlike concept drift as considered e. g. in [17] and unlike covariate shift as considered e. g. in

[18], we need to guarantee the functionality of the basic offline model in addition to the adaptive online model, independent of the shape of the observed drift.

Recently, an effective architecture has been proposed to deal with this challenge: A hybrid architecture dubbed OOL [19], [20] combines two complementary classifiers via a dynamic classifier selection scheme which is based on classifier confidences [21]–[24]. One of the classifiers is static and represents the basic functionality, while the other realises an adaptive model which can be adjusted to the online training data. Both classifiers are enhanced with a confidence estimation, based on which the decision is taken how to compute the overall output. This classifier selection strategy is accompanied by a scheme when and how to adapt the online model according to the given data. Since this second online classifier starts from scratch and learns incrementally during its whole lifetime, it allows to deal with concept-drift; in particular, it can react to new classes y and covariate shift; the latter requires a refinement of the offline model in specific regions of the data space. First promising results have been presented in [19], [20], based on intuitive prototype-based models.

This general architecture, though conceptually simple and efficient, bears a severe risk: while the offline classifier remains valid for regions of the data space which are covered by the offline training set, virtual concept drift, i. e. a change of the probability $p(x)$, can cause a wrong confidence estimation of the offline classifier for regions which become relevant while online learning. Provided training and test data were known in advance, this setting could be accounted for suitable data reweighing schemes, one of the most promising techniques to deal with covariate shift [18]. In our setting, this information is not available priorly, and there occurs the need to adapt the classifier confidence estimation according to the observed drift. We refer to this problem as *confidence drift*: while training, the estimation of the confidence of the offline or online classifier becomes invalid, such that their combination leads to a false result, albeit the single classifiers are still valid in their respective regions.

We focus on one popular type of classifiers where this problem is particularly pronounced: we consider intuitive distance-based classifiers which are enhanced by the powerful concept of metric learning [25], [26]. On the one hand, such models are represented in terms of typical prototypes, which enables a very natural interface to incremental learning with scalable memory resources. On the other hand, metric learning enhances these models to state-of-the-art classifiers which autonomously adapt the metric parameters, i. e. the internal representation of the given data according to their relevance for the classification task. While the latter aspect allows such models to efficiently deal with high dimensional or heterogeneous data sets where the standard Euclidean metric would not be suited, it adapts the data representation according to the given setting. Hence it fails whenever concept drift requires a different data representation, and it leads to a wrong confidence estimation in such cases. In this contribution, we propose a simple yet efficient approach how to deal with such confidence drift: we propose an online adaptation of the internal data representation by means of online metric learning for both, offline and online classifier, and a self-adjusted weighting scheme which autonomously adjusts the relevance of the respective data representation for confidence estimation.

This contribution is structured as follows: At first we briefly describe the considered scenario, and we recall the hybrid architecture which combines an online and offline model. We rely on a realisation of this architecture with learning vector quantisation (LVQ) schemes such as proposed in the approaches for batch scenarios [26]–[28] and its recent extension to incremental learning, online LVQ (ioLVQ) [29]. Thereafter we will explain an example of confidence drift, and we propose its remedy by means of suitable metric learning strategies for confidence estimation. Before the conclusions, we demonstrate the capability of the extended architecture on several data sets.

II. The OOL architecture [20]

We expect two training phases in our scenario. In the first phase (offline), we assume access to all offline training data for multiple passes trough the training set. In the second phase (online), we consider training data points which are available one after another, within a streaming setting. For both training phases, we assume labelled training data.

The basic idea of the OOL architecture (Fig. 1, [19], [20]) is the combination of a pre-trained offline model (first training phase) that preserves known knowledge of the desired task with an incremental online model adapting to special characteristics or new classes during the lifelong learning application in its (dynamic) environment (second training phase). Both classifiers also provide certainty levels, and classification of a given data point is done by the most reliable classifier with respect to their certainty values. Figure 1 shows the OOL scheme. Subsequently, we specify the role of its parts.

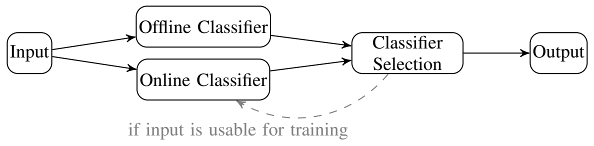


Fig. 1. OOL architecture combines an online and offline classifier [30].

Input: A data point $\mathbf{x} \in \mathbb{R}^n$ is the input of the system. It can contain a label y denoting a new or a known class. Both classifiers receive the input.

Offline Classifier: This module is a place holder for an offline classifier which provides a predicted class label y^{off} together with a certainty value $\Upsilon^{\text{off}}(\mathbf{x})$ for an input data point $\mathbf{x}$. During the whole application, this offline classifier preserves the gained knowledge, i. e. its output y^{off} is static. The output (y^{off}, $\Upsilon^{\text{off}}(\mathbf{x})$) is passed to the classifier selection.

Online Classifier: This module is a place holder for an online incremental classifier which provides a predicted class label y^{on} and a certainty value $\Upsilon^{\text{on}}(\mathbf{x})$ for an input data point $\mathbf{x}$. The online classifier is updated under certain conditions when receiving new training samples (see classifier selection). The output (y^{on}, $\Upsilon^{\text{on}}(\mathbf{x})$) is passed to the classifier selection.

Classifier Selection: The classifier selection module has two tasks: 1) choosing the online or the offline classifier based on their certainty values and 2) determining if the given input is a proper training example for the online classifier.

These two aspects work as follows: 1) The more reliable classifier determines the class label of the system

$$y^{\text{sys}} := \begin{cases} y^{\text{off}}, \Upsilon^{\text{off}}(\mathbf{x}) \geq \Upsilon^{\text{on}}(\mathbf{x}) \\ y^{\text{on}}, \text{else} \end{cases} .$$

2) The online classifier is trained as follows: Assume a labelled example is present $(\mathbf{x}, y)$. If the offline classifier is confident and classifies correctly, the online classifier is not updated. The online classifier is adapted, if at least one of the three following conditions is valid.

1. The online classifier is more reliable than the offline one
$$\Upsilon^{\text{on}}(\mathbf{x}) > \Upsilon^{\text{off}}(\mathbf{x}) \tag{1}$$
2. The offline classifier is more reliable but it provides a wrong class label.
$$\Upsilon^{\text{on}}(\mathbf{x}) < \Upsilon^{\text{off}}(\mathbf{x}) \wedge y^{\text{off}} \neq y \tag{2}$$
3. Both models are unreliable (here: $\Gamma = 0.6$).
$$\max\{\Upsilon^{\text{on}}(\mathbf{x}), \Upsilon^{\text{off}}(\mathbf{x})\} < \Gamma \tag{3}$$

Output: The output of the system, for a given input, is the selected class label y^{sys} with its certainty value.

A. OOL Using Learning Vector Quantisation

We use learning vector quantisation (LVQ) [31] for an implementation of the OOL architecture. LVQ schemes got popular lately in the context of big data and interpretable models due to its flexibility and intuitive classification scheme, see e. g. [32]–[41]. Assume v training data points $\mathbf{x}_i$ with class label y_i such that $(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{1, \ldots, C\}$. A LVQ classifier consists of a set of prototypes $W = \{\mathbf{w}_j \in \mathbb{R}^n\}_{j=1}^k$ equipped with class labels $c_j \in \{1, \ldots, C\}$. A given point $\mathbf{x}_i$ is classified according to the label of the closest prototype, the best matching unit (BMU), as measured in the squared Euclidean distance $\|\mathbf{x} - \mathbf{w}\|^2$ or variants thereof.

1) The Used LVQ Approaches: Given training data, the generalised LVQ (GLVQ) [27] is a learning scheme of LVQ that is based on a cost function. It performs a stochastic gradient descent on its cost function (4) with respect to the prototypes using a step size ε_w while performing several training epochs (here 100) through the data.

$$E = \sum_{i=1}^{v} \Phi(\underbrace{(d^+(\mathbf{x}_i) - d^-(\mathbf{x}_i))/(d^+(\mathbf{x}_i) + d^-(\mathbf{x}_i))}_{=:\mu(\mathbf{x}_i)}) \tag{4}$$

The function Φ is monotonic increasing, e. g. the logistic function. $d^{\pm}$ is the distance of the data point $\mathbf{x}_i$ to the closest prototype $\mathbf{w}^{\pm}$ of the correct/incorrect class. The cost function (4) strongly correlates to the classification error since a data point is classified correctly iff the nominator of the cost function is negative. Note that the relative similarity (RelSim)

$$\text{RelSim}(\mathbf{x}) := -\mu(\mathbf{x}) = (d^-(\mathbf{x}) - d^+(\mathbf{x}))/(d^-(\mathbf{x}) + d^+(\mathbf{x}))$$

relates to the summands of the GLVQ costs and can be interpreted as a certainty of the classification (for this reason multiplied by -1): The values $\text{RelSim}(\mathbf{x})$ range in the interval $(-1, 1)$ and values near 0 refer to high uncertainty, high values near 1 refer to high certainty, and negative ones refer to a wrong classification since $d^+ > d^-$. Lately, it has been investigated that the RelSim measure provides an efficient estimation of a confidence usable for rejection [42]. The RelSim shows a similar quality as an explicit probabilistic modelling but at lower computational costs. The calculation of the RelSim is based on the estimated class label with respect to the model for an unlabelled data point $\mathbf{x}$, i. e. d^+ is the distance of $\mathbf{x}$ to the BMU $\mathbf{w}_s$ (defines the label of $\mathbf{x}$); hence d^- is the distance of $\mathbf{x}$ to any prototype with a different class label than $\mathbf{w}_s$.

Recently the concept of metric learning gained much attention in distance-based classification which can be integrated in LVQ [25], [43], [44]. Especially, there exists a generalisation of the GLVQ towards a general quadratic form $(\mathbf{x}-\mathbf{w}_j)^T \Lambda (\mathbf{x}-\mathbf{w}_j)$ with positive semi-definite matrix Λ which is proposed under the acronym GMLVQ [26]. The updates of the metric regulates the learning rate ε_m. Formal learning theoretical guarantees have proved that GMLVQ networks constitute an efficient large margin scheme with excellent generalisation ability [26].

The ioLVQ [29] is applicable in scenarios where training data points arrive one after another, i. e. in incremental online scenarios. The model starts with no prototypes (no knowledge), i. e. $W = \emptyset$ and the formulas for prototype and metric updates are identical with the batch LVQ versions. The only difference is the dynamically changing set of prototypes due to data drift or new classes. Insertion and deletion of prototypes rely on the aim to decrease the costs (4):

1. New classes [45]: The first training data point $(\mathbf{x}, y)$ of a new class, i. e. $y \neq c_j, \forall j$, becomes a new prototype with label y. The class label is available since we consider a dialogue with the system, e. g. a user provides data points of a new class. Hence, we do not detect a new class from unsupervised data like e. g. [46].

2. Prototype insertion: Reducing errors lowers the costs (4). A set S with maximum storage capacity g_{max} stores errors during training [47], [48]. Once $|S| = g_{\text{max}}$, a prototype with label p is added for each class p for which S stores errors. The point $(\mathbf{x}_i, p)$ in S with lowest $\text{RelSim}(\mathbf{x}_i)$ identifies the prototype location. After insertion, S is cleared, i. e. $S := \emptyset$.

3. Prototype deletion: Since prototypes can become obsolete or can do more wrong than correct classifications, a strategy for prototype deletion is needed [49]. Each prototype $\mathbf{w}$ is accompanied by a parameter $\eta(\mathbf{w})$, initialised by zero, that integrates the certainty of this prototype. The update

$$\eta(\mathbf{w}_l) := \eta(\mathbf{w}_l) + \text{RelSim}(\mathbf{x})$$

efficiently computes $\eta(\mathbf{w}_l)$ for the BMU $\mathbf{w}_l$ provided the point $(\mathbf{x}, y)$ is encountered. The value $\eta(\mathbf{w}_l)$ is negative iff on average, the prototype did more wrong classifications than correct ones, weighted by their certainty. After every r_{num} training data points, prototypes with $\eta(\mathbf{w}) < 0$ are deleted.

Because of the promising results of the OOL architecture [20], we describe and rely on their realisation in the following.

2) The Precise Realisation: To describe the used scheme in detail, we have specify the used offline classifier, online classifier, and confidence estimation.

Offline Classifier: Here we choose a pre-trained GMLVQ[1] model, i. e. a set of trained prototypes $\mathbf{w}_j$ for the C known classes with a trained global metric using Λ. We use the RelSim as related certainty value for an input data point $\mathbf{x}$, i. e. $\Upsilon^{\text{off}}(\mathbf{x}) = \text{RelSim}^{\text{off}}(\mathbf{x})$.

Online Classifier: Here we choose the ioLVQ [29] based on a GMLVQ model. We use the RelSim as related certainty value for an input data point $\mathbf{x}$, i. e. $\Upsilon^{\text{off}}(\mathbf{x}) = \text{RelSim}^{\text{off}}(\mathbf{x})$.

Note that in general the metrics of the offline and online classifiers are different since they are trained on different data. This can cause problems in the case of confidence drift, in particular when the validity of the confidence estimation of the static offline classifier changes. This pitfall occurs in the context of metric learning whenever the metric tensor of the online system becomes orthogonal to the metric of the offline system, hence the respective internal data representations mismatch. In the following we give an example thereof and we develop a solution for it.

III. An Example of Confidence Drift

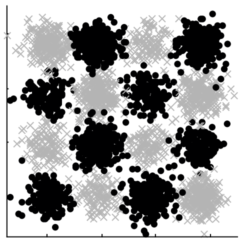
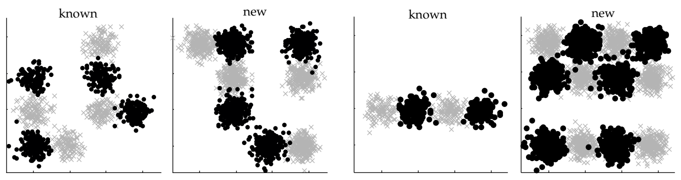


Fig. 2. The checkerboard data set (Colour encodes class)

Assume a binary 2D 4x4 checkerboard data set (Fig. 2). We consider two different data splits for the first and the second training phase of the architecture (Fig. 3). The offline classifier is trained on the known data only (first training phase) while the architecture during use encounters known and new data (second training phase). This leads to a desired data space partitioning with respect to the classifiers, as shown in Fig. 4.

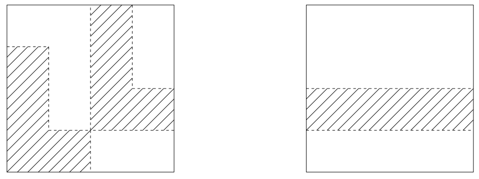
Analysing setting A, one recognises that all dimensions are necessary in order to classify the data and this holds for the known and the new data. Contrary in setting B the known data can be classified using only one dimension of the data while classification of the new data still has to use both dimensions. This difference matters for metric learning, in

[1]We use the LVQ toolbox at: http://matlabserver.cs.rug.nl/gmlvqweb/web/



(a) Setting A (b) Setting B

Fig. 3. The checkerboard data is used in three different settings: In setting A both dimensions of the data are necessary to classify the data in the known as well as in the new part of the data. In the known data of setting B, one dimension is enough for classification but in the new data both dimension are again needed. Setting C is setting B inverted.



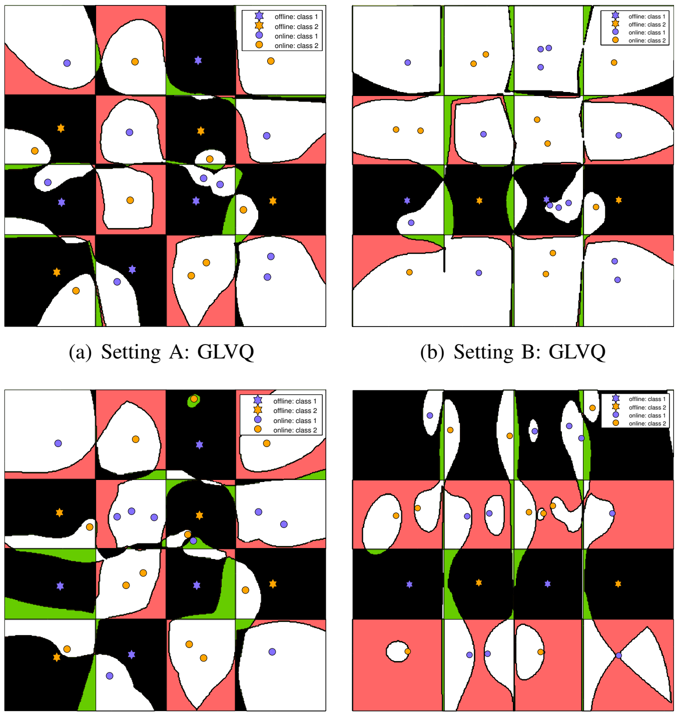
(a) Setting A (b) Setting B

Fig. 4. The desired partitioning of setting A and B. The hatched areas should be classified by the offline classifier while the white areas are related to the online classifier. The desired partitioning of setting C is (b), inverted.

particular for the metric of the offline classifier in setting B. In this case the matrix Λ will only focus on one dimension $\Lambda_{11}{\approx}1$ while all other elements are approximately zero. Figure 5 contains exemplary results of setting A and B for the described architecture with metric learning (GMLVQ) and a similar architecture without metric learning, simply using the Euclidean distance (GLVQ). The GLVQ architecture shows a similar data partitioning like the desired ones in Fig. 4. Hence for this models there is no problem with the classifier selection part. Analysing the GMLVQ architectures, one can see that setting A works fine too but setting B does not due to its trained offline metric. For setting B with the GMLVQ architecture a confidence drift happens which cannot be followed by the static internal metric which is suited for the known data but which is improper for the new data. The problem occurs due to the fact that the offline classifier deal with a too simplistic data representation: it disregards data dimensions which are irrelevant for the offline training data, but which become relevant for future data points; in consequence, confidence estimation based on this metric are wrong in the online scenario. Note that an abstraction from irrelevant regions is crucial for LVQ classifiers to provide good generalisations for high dimensional data sets. At the same time, it is priorly unclear which abstractions are too rigid for future data, hence online adaptation of this confidence measure becomes necessary. In the next section we develop a strategy which allows the offline classifier to adapt its certainty estimation in the case of confidence drift.

IV. Online Metric Learning for an Adaptation to Confidence Drift

The offline classifier is static and it consists of a set of prototypes W^{off} and a trained metric using Λ^{off}. Since we want to keep the gained knowledge of the offline classifier,



(a) Setting A: GLVQ (b) Setting B: GLVQ

(c) Setting A: GMLVQ (d) Setting B: GMLVQ

Fig. 5. The images show the results for the specific setting and a specific architecture. The black areas are classified by the offline classifier while the online classifier is responsible for the white areas (correctly classified). Red areas are misclassified by the offline classifier and the green areas are wrong classified by the online classifier. The prototypes of the classifiers are shown as stars and circles. Their colour indicates the class. GLVQ refers to the architecture using the standard Euclidean distance instead of metric learning.

we do not change the classifier itself. Instead, we introduce a metric Λ^{correct} which can be used to correct the wrong data representation for its confidence estimation, taking into account concept drift on the online training set. More specifically, RelSim, the confidence, is computed by the combination

$$\Lambda^{\text{conf}} = \alpha \cdot \Lambda^{\text{correct}} + (1-\alpha)\cdot\Lambda^{\text{off}}, \quad \alpha \in [0.1, 0.9] \tag{5}$$

where the scaling parameter α is initialised with 0.1 and the matrix Λ^{correct} is initialised with Λ^{off}, and both are adapted in online training, whenever the offline classifier is erroneous, i. e. (cp. (2)) holds (Fig. 6): Λ^{correct} is adapted using the GMLVQ update rule considering W^{off} together with Λ^{correct} as GMLVQ model (but no prototype update is done). In case of a point with new class label (unknown to the offline GMLVQ), $\mathbf{w}^+$ and d^+ are taken from the online classifier to update Λ^{correct}.

The choice of α is crucial, and a static prior choice is usually suboptimal. Therefore we use a simple Hebbian adaptation strategy for $\alpha \in [0.1, 0.9]$ as follows: A counter Γ (initialised with zero) counts the relevance of Λ^{correct} versus Λ^{off} on the training data. It is decreased by one when ever

$$\text{RelSim}^{\text{on}}(\mathbf{x}) < \text{RelSim}^{\text{off}}(\mathbf{x}) \wedge y^{\text{off}} = y \tag{6}$$

and it is increased by one whenever (2) holds. We enforce an upper and lower bound $a_1 \leq \Gamma \leq b_2$ on the counter to

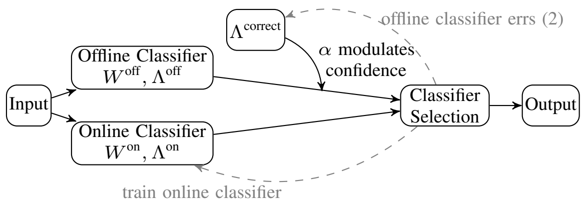

Λ^{correct}
offline classifier errs (2)
α modulates confidence
Offline Classifier $W^{\text{off}}, \Lambda^{\text{off}}$
Input
Classifier Selection
Output
Online Classifier $W^{\text{on}}, \Lambda^{\text{on}}$
train online classifier

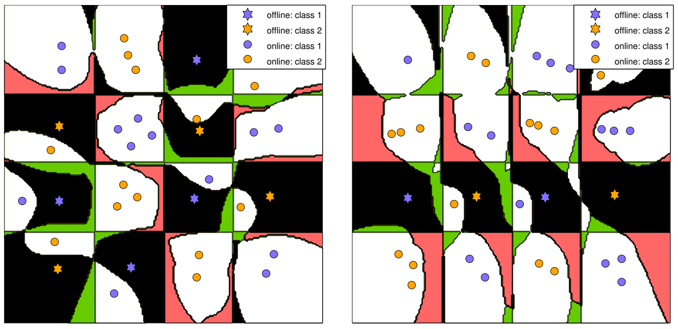
Fig. 6. Scheme of OOL extension for confidence drift adaptation. The offline classifier $(W,^{\text{off}} \Lambda^{\text{off}})$ still predicts the label y^{off} but in order to calculate $\text{RelSim}(\mathbf{x})$ the metric Λ^{conf} (5) is used which takes into account the quantities $\Lambda,^{\text{off}} \Lambda^{\text{correct}}$ and α.

prevent plateaus for the optimization, and we disregard counters approximately equal to zero $a_2 \leq \Gamma \leq b_1$ for an adaptation of α where $a_1{<}a_2{<}0{<}b_1{<}b_2$. Assume a small step size $\varepsilon_\alpha{>}0$, the update rule of the scaling parameter α is

$$\alpha = \alpha \begin{cases} -\varepsilon_\alpha, & a_1 < \Gamma < a_2 \wedge (2) \text{ holds} \\ +\varepsilon_\alpha, & b_1 < \Gamma < b_2 \wedge (2) \text{ holds} \end{cases}.$$

This corresponds to a Hebbian scheme since the relevance of Λ^{correct} is increased / decreased depending on its contribution to a correct classification.

Exemplary results for setting A and B of the architecture with confidence adaptation are visualised in Fig. 7. It can be seen that this change enables the hybrid architecture to learn correctly for both settings. In the next section we evaluate the proposed architecture on several data sets.



(a) Setting A (b) Setting B

Fig. 7. The images show the results for setting A and B for the architecture with confidence drift adaptation. The black areas are classified by the offline classifier while the online classifier is responsible for the white areas (correctly classified). Red areas are misclassified by the offline classifier and the green areas are wrong classified by the online classifier. The prototypes of the classifiers are shown as stars and circles. Their colour indicates the class.

V. Experiments

We consider experiments on four data sets: two 2D data sets (Blossom, Checkerboard) for visualisation and analysing the architecture, the U.S. Postal Service (USPS[2]) Handwritten Digits data, the Letter data set [52], and the Outdoor[3] data set as benchmarks. Each data set is divided into two partitions: known and new data. Known data are available during both

[2]Data is obtained from: http://www.cs.nyu.edu/~roweis/data.html

[3]Thanks to Viktor Losing for providing the data.

training phases while instances of new data are only accessible in the second phase.

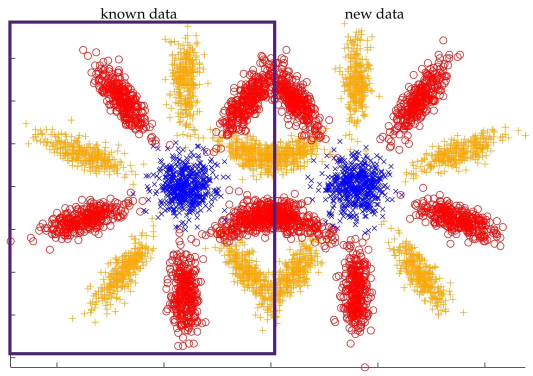


Fig. 8. The Blossom data: Each blossom forms a data partition. The colour encodes the class.

- Checkerboard: Artificially created two class data set (Fig. 2). Each *field* of the Checkerboard contains 1000 data points which sums up to 8000 points per class.
- Checkerboard noise: The Checkerboard data are enhanced by a noisy third dimension. The values of the third dimension are uniformly distributed in (0,1).
- Blossom: Artificial data set with three classes (Fig. 8). Each blossom forms one partition of the data.
- USPS: This data set contains 8-bit grayscale images of "0" to "9"; 1100 instances per class. The dimension of each instance is reduced to 30 with principal component analysis [50]. Half of the classes form the first respectively the second partition of the data.
- Letter: The letter data [52] contain 20,000 data points with 16 dimensions, related to 26 classes (capital letters A. . . Z). The first 14 letters form the first partition of the data and the other letters belong to the second partition.
- Outdoor: The outdoor obstacle data set [20], [51] consists of 40 objects. Each object has 100 representing instances consisting of 21 values of a 6 bin rg chromaticity diagram. Half of the classes form the first respectively the second partition of the data.

We use the same experimental setting as in [20]. Hence, we assume randomly ordered data points which are used only once, like in a streaming setting, using constant learning rates (except for the batch GMLVQ). The evaluation of the data sets is based on a 10-fold cross validation with ten repeats as follows: Each data set is divided into two parts: known and new data as stated before. Known data characterise data for offline training and online training. New data characterise data which are only available for online training. It contains new classes or data subject to drift of $p(x)$. For a 10-fold cross validation, each partition is divided into 10 folds (see Fig. 9). The offline training set train$^{\text{off}}$ consists of four folds of known data and the online training data train$^{\text{on}}$ (encountered during use) consists of four folds of known and of eight folds of new data. For a proper evaluation of the architecture, we consider three different test sets: test$^{\text{known}}$, test$^{\text{new}}$ and test$^{\text{all}}$. The test set

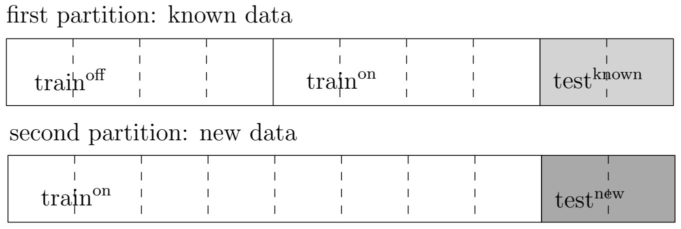


Fig. 9. Each data set is divided into two partitions: known and new data. Two folds of each set form a test set: test$^{\text{known}}$ and test$^{\text{new}}$ respectively.

test$^{\text{known}}$ are two folds of known data and its error will show the performance on these data which samples the static ground knowledge (i. e. the performance tests the stability of learning). The second test set is test$^{\text{new}}$ which contains two folds of new data and it will show the accuracy of newly gained knowledge in the online part of the system, i. e. it tests the flexibility the system. The overall performance is judged with the test set test$^{\text{all}} = \text{test}^{\text{known}} \cup \text{test}^{\text{new}}$, i. e. it tests the ability to deal with both challenges, stability and plasticity.

TABLE I
PARAMETERS OF THE OFFLINE AND THE ONLINE MODEL

	GMLVQ			ioLVQ			
	w/class	ε_w	ε_m	g_{max}	r_{num}	ε_w	ε_m
Checkerboard	4/2/6 (A/B/C)	0.3	0.07	20	320	0.1	0.01
Blossom	5 or 1	0.3	0.07	20	320	0.1	0.01
USPS	1	0.4	0.01	20	400	0.1	0.001
Outdoor	5	$4{\cdot}10^{-5}$	10^{-5}	5	900	$4{\cdot}10^{-5}$	10^{-5}
Letter[4]	1	0.01	0.001	20	400	0.01	0.001

Table I shows the parameters of the experiments and Tab. II and Tab. III show the results. They contain the accuracies of the three test sets and the number of prototypes, and (if available) the trained α value.

A. *Evaluation of the Checkerboard Data (Tab. II)*

Architecture without concept drift adaptation: Firstly, the results of Checkerboard and Checkerboard noise are similar which means that the integrated metric learning detects and neglects the noisy dimension. This indicates the merit of metric learning. Secondly, the number $|W|$ of the prototypes is in the same range for all tree setting indicating that the number of prototypes is kind of invariant to the used setting. Thirdly, the accuracy values of test$^{\text{known}}$ show a desired performance for all settings. The same holds for test$^{\text{new}}$ but for setting A only. In case of setting B or C the accuracy is low which means that a confidence drift happened and the plain architecture is unable to deal with it. This is also the reason for low test$^{\text{all}}$ accuracies (setting A/B).

Architecture with concept drift adaptation: This extended architecture provides similar results for both Checkerboard data sets. Hence noisy dimensions with no information are neglected, too. In comparison to the plain architecture, the number of used prototypes increases slightly. Since we know, that there is no

[4]We used 300 training epochs for the Letter data.

TABLE II
THE RESULTS OF THE CHECKERBOARD DATA

We report the average accuracy on the test sets, the related average prototype number $|W|$ of OOL, and the α values. Results with α=0 belong to the plain architecture without confidence drift adaptation.

Data	α	$\|W\|$	test$^{\text{known}}$	test$^{\text{new}}$	test$^{\text{all}}$
Checkerboard					
Setting A	0	28.14	96.45	90.64	93.60
Setting B	0	23.54	94.94	49.35	60.65
Setting C	0	27.42	95.58	43.71	56.81
Setting A	0.1127	36.58	93.91	93.78	93.79
Setting B	0.5217	36.84	92.06	90.28	90.91
Setting C	0.6336	45.90	92.39	93.50	93.12
Checkerboard noise					
Setting A	0	27.20	95.91	90.05	92.93
Setting B	0	22.25	94.82	49.08	60.54
Setting C	0	25.36	95.12	43.51	56.37
Setting A	0.1094	35.83	93.49	93.55	93.52
Setting B	0.5387	36.81	92.80	90.59	91.14
Setting C	0.6348	45.50	93.31	93.56	93.57

confidence drift in setting A, we expect the result to reflect this. The result does reflect it, since the value of α in both cases is near the lowest possible value and hence the architecture highlights that there is no need for a confidence adaptation. The other settings need a confidence adaptation which is nicely reflected by the α values which improve together with the trained λ^{correct} the performance especially on test$^{\text{new}}$ and test$^{\text{all}}$.

B. Evaluation of the Blossom and the Benchmarks (Tab. III)

TABLE III
THE RESULTS OF THE BLOSSOM AND THE BENCHMARK DATA

We report the average accuracy on the test sets, the related average prototype number $|W|$ of OOL, and the α values. Results with α=0 belong to the plain architecture without confidence drift adaptation.

Data	α	$\|W\|$	test$^{\text{known}}$	test$^{\text{new}}$	test$^{\text{all}}$
Blossom					
	0	27.33	96.49	97.13	96.80
	0.1	30.72	95.59	97.55	96.61
USPS					
	0	64.19	91.29	59.54	75.26
	0.6986	77.25	89.13	84.70	86.98
Letter					
	0	210.70	80.51	70.15	75.62
	0.8112	235.24	80.23	78.00	79.25
Outdoor					
	0	286.63	89.89	66.65	78.18
	0.4135	286.64	90.03	66.74	78.05

Since the known and the new data of the Blossom data set are similar except an offset in one coordinate, the trained metrics of the online and the offline classifier should be comparable and hence there is no confidence drift. This can also be seen on the already high accuracy values of the plain architecture. We expect that the extended architecture reflects this with a low α value and similar performances than the plain architecture. As can be seen in Tab III this expectation is met.

For the USPS, one can see that there might be a confidence drift since the performance on test$^{\text{new}}$ and test$^{\text{all}}$ of the plain architecture is bad. Comparing those results with them of the extended architecture, it turns out that the adaptation to confidence drift is used and that it helps to improve the performance on these two test sets. The same facts can be seen for the Letter data which emphasise the usefulness of confidence adaptation.

The Outdoor data are a difficult task as mentioned in [20]. The OOL architecture, however, manages to retain known knowledge (test$^{\text{known}}$) and to gain new information. Comparing the plain architecture with the extended one, it turns out that the results are similar. The reason therefore can be attributed to the fact that confidence drift is not present in this case.

C. Summing up the Results

The proposed extension increases the performance in cases with confidence drift (Checkerboard settings B/C, USPS, Letter) without lowering the performance in cases without confidence drift (Checkerboard settings A, Blossom, Outdoor). This behaviour is viable since confidence adaptation in the latter cases can be potential problematic.

VI. CONCLUSION

The OOL architecture constitutes an approach suitable for lifelong learning scenarios and for streaming data. Its basic components are a pre-trained, static offline classifier, an adaptive online classifier, and a dynamic classifier selection based on confidences of both classifiers. As pointed out in this article, the classifier selection strategy can cause problems in the case of confidence drift. This pitfall occurs in the context of metric learning whenever the internal data representation of the online and offline classifier mismatch. We have proposed an efficient extension to this architecture which allows an adaptation to confidence drift adaptation based on an online metric learning and weighting scheme. We analysed the OOL architecture based on the popular GMLVQ schemes. It turned out that the proposed modification is effective for a number of artificial and benchmark data: it enables an efficient scheme to avoid confidence drift for metric-based classifiers wherever it would be present with the original OOL scheme.

ACKNOWLEDGMENT

BH gratefully acknowledges funding by the CITEC center of excellence. LF acknowledges funding by the CoR-Lab Research Institute for Cognition and Robotics and acknowledges the financial support from Honda Research Institute Europe.

REFERENCES

[1] D. L. Silver, Q. Yang, and L. Li, “Lifelong Machine Learning Systems: Beyond Learning Algorithms,” in *AAAI Spring Symposium: Lifelong Machine Learning*, ser. AAAI Technical Report, vol. SS-13-05, 2013.
[2] S. Thrun and T. M. Mitchell, “Lifelong Robot Learning,” Robotics and Autonomous Systems, Tech. Rep., 1993.

[3] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, and K. Inoue, "Learning Revised Models for Planning in Adaptive Systems," in *Proc. of Int. Conf. on Software Engineering*. IEEE Press, 2013, pp. 63–71.

[4] D. Stavens, G. Hoffmann, and S. Thrun, "Online Speed Adaptation Using Supervised Learning for High-speed, Off-road Autonomous Driving," in *Proc. Int. Joint Conf. on Artif. Intell.*, 2007, pp. 2218–2224.

[5] Y. Jin and B. Sendhoff, "Alleviating Catastrophic Forgetting via Multi-Objective Learning," in *Proc. Int. Joint Conf. on Neural Netw., part of the IEEE World Congress on Comp. Intellig.* IEEE, 2006, pp. 3335–3342.

[6] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An Empirical Investigation of Catastrophic Forgeting in Gradient-Based Neural Networks," *ArXiv e-prints*, Dec. 2013.

[7] R. Polikar and C. Alippi, "Guest Editorial Learning in Nonstationary and Evolving Environments," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 9–11, 2014.

[8] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, p. 44, 2014.

[9] L. L. Minku and X. Yao, "DDD: A new ensemble approach for dealing with concept drift," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 4, pp. 619–633, 2012.

[10] C. Alippi, G. Boracchi, and M. Roveri, "Just-in-time classifiers for recurrent concepts," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 24, no. 4, pp. 620–634, 2013.

[11] H. He, S. Chen, K. Li, and X. Xu, "Incremental Learning From Stream Data," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 1901–1914, 2011.

[12] G. Cauwenberghs and T. Poggio, "Incremental and Decremental Support Vector Machine Learning," in *Advances in Neural Information Processing Systems. NIPS*, 2000, pp. 409–415.

[13] K. Crammer, A. Kulesza, and M. Dredze, "Adaptive Regularization of Weight Vectors," *Machine Learning*, vol. 91, no. 2, pp. 155–187, 2013.

[14] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn++: An Incremental Learning Algorithm for Supervised Neural Networks," *IEEE Trans. Sys., Man, and Cybern., C*, vol. 31, no. 4, pp. 497–508, 2001.

[15] M. J. Hosseini, Z. Ahmadi, and H. Beigy, "Using a classifier pool in accuracy based tracking of recurring concepts in data stream classification," *Evolving Systems*, vol. 4, no. 1, pp. 43–60, 2013.

[16] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *Journal of Machine Learning Research*, vol. 8, pp. 2755–2790, 2007.

[17] G. Ditzler and R. Polikar, "Incremental learning of concept drift from streaming imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 10, pp. 2283–2301, 2013.

[18] M. Sugiyama and M. Kawanabe, *Machine Learning in Non-Stationary Environments - Introduction to Covariate Shift Adaptation*, ser. Adaptive computation and machine learning. MIT Press, 2012.

[19] H. Wersing and J. F. Queißer, "System for controlling an automated device," Patent EP 2 690 582 A1, 2014.

[20] L. Fischer, B. Hammer, and H. Wersing, "Combining offline and online classifiers for life-long learning," in *Proc. International Joint Conference on Neural Networks, IJCNN*, 2015, pp. 2808–2815.

[21] B. V. Dasarathy and B. V. Sheela, "A Composite Classifier System Design: Concepts and Methodology," *Proceedings of IEEE*, vol. 67, no. 5, pp. 708–713, May 1979.

[22] M. Sabourin, A. Mitiche, D. S. Thomas, and G. Nagy, "Classifier combination for hand-printed digit recognition," in *2nd Int. Conf. Document Analysis and Recognition, ICDAR*. IEEE, 1993, pp. 163–166.

[23] K. S. Woods, W. P. Kegelmeyer, and K. W. Bowyer, "Combination of Multiple Classifiers Using Local Accuracy Estimates," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 4, pp. 405–410, 1997.

[24] A. S. Britto Jr., R. Sabourin, and L. E. S. de Oliveira, "Dynamic selection of classifiers - A comprehensive review," *Pattern Recognition*, vol. 47, no. 11, pp. 3665–3680, 2014.

[25] A. Bellet, A. Habrard, and M. Sebban, "A Survey on Metric Learning for Feature Vectors and Structured Data," *ArXiv e-prints*, Jun. 2013.

[26] P. Schneider, M. Biehl, and B. Hammer, "Adaptive Relevance Matrices in Learning Vector Quantization," *Neural Computation*, vol. 21, no. 12, pp. 3532–3561, 2009.

[27] A. Sato and K. Yamada, "Generalized Learning Vector Quantization," in *Adv. Neural Inf. Proces. Sys. NIPS*, vol. 7, 1995, pp. 423–429.

[28] S. Seo and K. Obermayer, "Soft Learning Vector Quantization." *Neural Computation*, vol. 15, no. 7, pp. 1589–1604, 2003.

[29] L. Fischer, B. Hammer, and H. Wersing, "Certainty-based prototype insertion/deletion for classification with metric adaptation," in *Proc. ESANN*, 2015, pp. 7–12.

[30] J. F. Queißer, "Using context for the combination of offline and online learning," Master Thesis, Bielefeld University, 2012. [Online]. Available: http://pub.uni-bielefeld.de/publication/2670657

[31] T. Kohonen, *Self-Organization and Associative Memory*. Springer Series in Information Sciences, Springer-Verlag, third edition, 1989.

[32] I. Giotis, K. Bunte, N. Petkov, and M. Biehl, "Adaptive Matrices and Filters for Color Texture Classification," *Journal of Mathematical Imaging and Vision*, vol. 47, pp. 79–92, 2013.

[33] M. Biehl, K. Bunte, and P. Schneider, "Analysis of Flow Cytometry Data by Matrix Relevance Learning Vector Quantization," *PLoS ONE*, vol. 8, no. 3, p. e59401, 2013.

[34] M. Biehl, P. Sadowski, E. B. G. Bhanot, A. Dayarian, P. Meyer, R. Norel, K. Rhrissorrakrai, M. D. Zeller, and S. Hormoz, "Inter-species prediction of protein phosphorylation in the sbv IMPROVER species translation challenge," *Bioinformatics*, 2014.

[35] J. G.-J. de Vries, S. C. Pauws, and M. Biehl, "Insightful stress detection from physiology modalities using learning vector quantization," *Neurocomputing*, vol. 151, Part 2, pp. 873 – 882, 2015.

[36] S. Kirstein, H. Wersing, and E. Körner, "A Biologically Motivated Visual Memory Architecture for Online Learning of Objects," *Neural Networks*, vol. 21, no. 1, pp. 65–77, 2008.

[37] S. Kirstein, H. Wersing, H.-M. Gross, and E. Körner, "A Life-Long Learning Vector Quantization Approach for Interactive Learning of Multiple Categories," *Neural Networks*, vol. 28, pp. 90–105, 2012.

[38] S. Kirstein, A. Denecke, S. Hasler, H. Wersing, H. Gross, and E. Körner, "A vision architecture for unconstrained and incremental learning of multiple categories," *Memetic Comp.*, vol. 1, no. 4, pp. 291–304, 2009.

[39] K. Bunte, P. Schneider, B. Hammer, F. Schleif, T. Villmann, and M. Biehl, "Limited Rank Matrix Learning, Discriminative Dimension Reduction and Visualization," *Neural Networks*, vol. 26, pp. 159–173, 2012.

[40] D. Nova and P. Estevez, "A review of learning vector quantization classifiers," *Neural Comp. and Appl.*, vol. 25, no. 3-4, pp. 511–524, 2014.

[41] X. Zhu, F. Schleif, and B. Hammer, "Adaptive conformal semi-supervised vector quantization for dissimilarity data," *Pattern Recognition Letters*, vol. 49, pp. 138–145, 2014.

[42] L. Fischer, B. Hammer, and H. Wersing, "Efficient rejection strategies for prototype-based classification," *Neurocomputing*, vol. 169, pp. 334 – 342, 2015.

[43] M. Biehl, B. Hammer, and T. Villmann, "Distance Measures for Prototype Based Classification," in *Brain-Inspired Computing - International Workshop, BrainComp*, 2013, pp. 100–116.

[44] A. Bellet and A. Habrard, "Robustness and Generalization for Metric Learning," *Neurocomputing*, vol. 151, no. 1, pp. 259–267, 2015.

[45] Y. Xu, F. Shen, and J. Zhao, "An incremental learning vector quantization algorithm for pattern classification," *Neural Computing and Applications*, vol. 21, no. 6, pp. 1205–1215, 2012.

[46] K. B. Dyer, R. Capo, and R. Polikar, "COMPOSE: A semisupervised learning framework for initially labeled nonstationary streaming data," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 25, no. 1, pp. 12–26, 2014.

[47] S. Kirstein, H. Wersing, and E. Körner, "Rapid Online Learning of Objects in a Biologically Motivated Recognition Architecture," in *Pattern Recognition Symposium DAGM*, 2005, pp. 301–308.

[48] T. C. Kietzmann, S. Lange, and M. Riedmiller, "Incremental GRLVQ: Learning Relevant Features for 3D Object Recognition," *Neurocomputing*, vol. 71, no. 13-15, pp. 2868–2879, 2008.

[49] M. Grbovic and S. Vucetic, "Learning Vector Quantization with Adaptive Prototype Addition and Removal," in *International Joint Conference on Neural Networks IJCNN*, 2009, pp. 994–1001.

[50] L. J. P. van der Maaten, "Matlab Toolbox for Dimensionality Reduction," 2013, http://homepage.tudelft.nl/19j49/Matlab_Toolbox_for_Dimensionality_Reduction.html.

[51] V. Losing, B. Hammer, and H. Wersing, "Interactive online learning for obstacle classification on a mobile robot," in *Proc. International Joint Conference on Neural Networks, IJCNN*, 2015, pp. 2310–2317.

[52] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml