Tackling Heterogeneous Concept Drift with the Self Adjusting Memory (SAM)

Viktor Losing, Barbara Hammer, Heiko Wersing

2018

Preprint:

This is an accepted article published in Knowledge and Information Systems.
The final authenticated version is available online at:
https://doi.org/10.1007/s10115-017-1137-y

Tackling Heterogeneous Concept Drift with the Self Adjusting Memory (SAM)

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

***Abstract*—Data mining in non-stationary data streams is particularly relevant in the context of Internet of Things and Big Data. Its challenges arise from fundamentally different drift types violating assumptions of data independence or stationarity. Available methods often struggle with certain forms of drift or require unavailable a priori task knowledge. We propose the Self-Adjusting Memory (SAM) model for the k-nearest-neighbor (kNN) algorithm. SAM-kNN can deal with heterogeneous concept drift, i.e., different drift types and rates, using biologically inspired memory models and their coordination. Its basic idea is to have dedicated models for current and former concepts used according to the demands of the given situation. It can be easily applied in practice without meta parameter optimization. We conduct an extensive evaluation on various benchmarks, consisting of artificial streams with known drift characteristics and real-world datasets. We explicitly add new benchmarks enabling a precise performance analysis on multiple types of drift. Highly competitive results throughout all experiments underline the robustness of SAM-kNN as well as its capability to handle heterogeneous concept drift. Knowledge about drift characteristics in streaming data is not only crucial for a precise algorithm evaluation, but it also facilitates the choice of an appropriate algorithm on real-world applications. Therefore, we additionally propose two tests, able to determine the type and strength of drift. We extract the drift characteristics of all utilized datasets and use it for our analysis of the SAM in relation to other methods.**[1]

I. Introduction

In the classical batch setting of data mining / machine learning, it is assumed that the complete task specific data are always available and can be accessed simultaneously without any restriction regarding the processing time. Furthermore, the i.i.d. property and stationary environments are presumed. State of the art machine learning methods are able to obtain very accurate results within this framework. However, an ever growing field of real-world applications generates data in streaming fashion at increasing rate, requiring large-scale and real-time processing. Streaming data are prevalent in domains such as health monitoring, traffic management, financial transactions and social networks [1] and are the foundation of the Internet of Things [2] technology. Supplementing streaming data with non-stationary environments leads to one of the recent key areas in data mining research: learning in streams under concept drift. Here, algorithms are challenged by a variety of possible forms of drift under strict limitations in terms of memory consumption and processing time.

In recent years, a few algorithms have been published able to handle specific types of drift such as abrupt [3], incremental [4] as well as reoccurring [5] drift. Some methods can be used for several types of drift when their metaparameters are set appropriately. However, this requires explicit prior knowledge about the task at hand. Furthermore, it is still an open question how to identify the type of drift in a given real-world data stream. In real-world applications, changes often occur in multiple and even concurrent forms at various rates. One example is the field of personalized assistance, in which individual user behavior is taken into account to provide appropriate assistance in various situations [6]. However, individual behavior in particular can change in arbitrary ways. Systems anticipating only certain forms of drift will perform suboptimal at best, or fail completely at worst, when unexpected forms of change occur.

Our Self-Adjusting Memory (SAM) in combination with the k-nearest-neighbor (kNN) classifier [7] is able to cope with heterogeneous concept drift and can be easily applied in practice without any parametrization. It exhibits several analogies to the structure of the human memory as we explicitly partition knowledge between short- and long-term memory. We regard the intuitive combination of the those memories as well as the explicitly preserved consistency as the main novelty of this contribution.

To enable a proper analysis of algorithms in terms of suitability for certain forms of drift, we do not only contribute new artificial benchmarks with ground truth drift type but also vision-based real-world datasets recorded outdoors: an environment where concept drift is naturally present.

Furthermore, we developed two tests able to determine the type of drift and its degree in a dataset, which, to the best of our knowledge, has not been done so far. This is particularly relevant for real-world datasets, where the drift characteristics are usually unknown. Information about the drift properties not only enables a more detailed evaluation of specific algorithms, but is also crucial for the choice of a suitable algorithm in the presence of limited resources.

Our extensive evaluation on artificial and real-world benchmarks demonstrates the gain of SAM-kNN in comparison with state-of-the-art approaches. As the only method, it achieves highly competitive results throughout all experiments, demonstrating its robustness and the capability of handling heterogeneous concept drift.

II. Framework

Our focus is data stream classification under supervised learning for incremental/online algorithms. The aim in supervised classification is to predict a target variable $y \in \{1, \ldots, c\}$ given a set of features $\mathbf{x} \in \mathbb{R}^n$.

In the classical batch setting, an algorithm generates a model

[1]The final publication is available at Springer via https://doi.org/10.1007/s10115-017-1137-y

h based on a training set $D_{\text{train}} = \{(\mathbf{x}_i, y_i) \,|\, i \in \{1, \ldots, j\}\}$. For convenience, we alternatively use the notation $y(\mathbf{x}_i) = y_i$. In the subsequent test phase, the model is applied on another set $D_{\text{test}} = \{(\mathbf{x}_i, y_i) \,|\, i \in \{1, \ldots, k\}\}$, whose labels are kept hidden. The model predicts a label $\hat{y}_i = h(\mathbf{x}_i)$ for every point $x_i \in D_{\text{test}}$ and the 0-1 loss

$$\mathcal{L}(\hat{y}_i, y_i) = \mathbb{1}(\hat{y}_i \neq y_i)$$

is calculated.

A. Streaming data

Data stream classification is usually evaluated in the online learning setting. A potentially infinite sequence $S = (s_1, s_2, \ldots, s_t \ldots)$ of tuples $s_i = (\mathbf{x}_i, y_i)$ arrives one after another. As t represents the current time stamp, the learning objective is to predict the corresponding label y_t for a given input x_t, which is supposed to be unknown. The prediction $\hat{y}_t = h_{t-1}(\mathbf{x}_t)$ is done according to the previously learned model h_{t-1}. Afterward, the true label is revealed and the loss $\mathcal{L}(\hat{y}_t, y_t)$ determined. Before proceeding with the next example, the applied learning algorithm generates a new model h_t on the basis of the current tuple s_t and the previous model h_{t-1}:

$$h_t = \text{train}(h_{t-1}, s_t).$$

The Interleaved Test-Train error for a sequence up to the current time t is given by:

$$E(S) = \frac{1}{t} \sum_{i=1}^{t} \mathbb{1}(h_{i-1}(x_i) \neq y_i). \tag{1}$$

Algorithms applied for streaming data face the challenge of anytime model adaptation; furthermore, the examples have to be processed in incoming order, often violating the assumption of sample independence.
Methods handling concept drift are, in addition, coping with non-stationary environments, characterized by a change of the underlying data generation process.

B. Concept drift

Concept drift [8] occurs when the joint distribution $P_t(\mathbf{x}, y)$ changes for at least two time steps t_0 and t_1:

$$\exists\, \mathbf{x} : P_{t_0}(\mathbf{x}, y) \neq P_{t_1}(\mathbf{x}, y),$$

The joint distribution can also be written as:

$$P_t(\mathbf{x}, y) = P_t(\mathbf{x}) P_t(y|\mathbf{x}),$$

where $P_t(\mathbf{x})$ is the distribution of the features and $P_t(y|\mathbf{x})$ the posterior probability of the classes. The term *real drift* is used to specify that the relation between observation and labels $P_t(y|\mathbf{x})$ varies over time. *Virtual drift*, also known as covariance shift, is present when the feature distribution $P_t(\mathbf{x})$ changes without affecting the posterior of the classes $P_t(y|\mathbf{x})$. Furthermore, the rate at which drift is taking place can be classified either as *abrupt*, resulting in a severe shift within the distribution, e.g., caused by a malfunctioning sensor, or *incremental*, an evolving change over time, e.g., evoked by a slowly degrading sensor. In the context of seasonal effects, drift is often characterized as *reoccurring* to describe that previous concepts are repeatedly emerging.

III. Related work

Algorithms dealing with concept drift can be roughly divided in active and passive approaches [9].
Active approaches explicitly detect the time of change t and usually discard the accumulated knowledge up to this point. Often, statistics extracted from data windows, covering different time periods, are analyzed for significant deviations. The most common statistic is the classification error [3], [10], but measures as the Kullback Leibler divergence, evaluating the distance between explicitly modeled probability distributions [11], are used as well. However, approaches detecting changes on the distributions are only applicable for low-dimensional data because the run-time grows exponentially with the number of dimensions and / or their performance drops significantly [11], [12].
ADaptive sliding WINdowing (ADWIN) [10] efficiently monitors the binary error history (it could be any i.i.d. value between 0 and 1) in a window containing all values since the last detected change. The window is repeatedly partitioned into two sub-windows of various size. Whenever the difference of their average error exceeds a threshold, depending on the size of the sub windows and a confidence parameter, a change is detected and the older window is dropped.
Active approaches are often combined with a sliding window containing the most recent examples, as these are assumed to be the most valuable for current predictions. Hereby, the size is a trade-off between fast adaptation (small window) and good generalization in stable phases without drift (large window). The size is adjusted dynamically to achieve both properties at the same time .
In Probabilistic Adaptive Windowing (PAW) [13] examples from the window are removed randomly leading to a mix of recent and older instances. The window size is not strictly bounded and varies around a target size. ADWIN is used as change detector and the window is cleared accordingly. This approach is coupled with the kNN classifier and achieves a low error rate in the experiments.
Active methods are able to detect abrupt drift quickly. However, they struggle with incremental change, which may not be significant enough and remains undetected.
The size of the sliding window can also be adapted by using a heuristic [14], or by minimizing the amount of errors in the past. Klinkenberg et al. present a theoretically well founded approach for Support Vector Machines (SVM) in [15]. They adapt the size such that an estimation of the leave-one-out error is minimized without any parametrization. This method is able to deal with abrupt and incremental change since no significant difference is required to adapt the size. We reuse ideas of this approach in our method but distinctly extend it as it is discussed in section 2.
A general weakness of active methods is that knowledge either slowly fades out or is discarded in case of detected drift. Although the most recent examples are usually the most valuable for current predictions, there are also cases in which older data carries crucial information, e.g. reoccurring drift. Here, the methods have to relearn former concepts and produce more mistakes than necessary. Our approach explicitly preserves information of former concepts and increases the conservation time span by repeated compression, allowing the access to former knowledge when necessary.
Passive approaches continuously adapt their model without

explicit awareness of occurring drift. On the one hand, this prevents pitfalls such as missed or false detected drifts, but on the other hand the adaptation speed is more or less constant leading to costly delays in the case of abrupt drift. Passive algorithms are dominated by ensembles, consisting usually of tree based models such as The Very Fast Decision Tree (VFDT) [16] [1]: A very fast anytime decision tree algorithm with asymptotic guarantees for its output, which incrementally adds splits based on the Hoeffding bound.
Ensemble members are continuously updated with incoming examples using techniques such as Bagging. Furthermore, new knowledge can be incorporated by the integration of new members, whereas irrelevant information is discarded by the deletion of corresponding old classifier. However, one major drawback is the comparably high computational effort.
Jaber et al. presented Dynamic Adaption to Concept Changes (DACC) in [17], an algorithm inspired by the Dynamic Weighted Majority [4] method. A classifier of the worst half of the pool is removed randomly after a predefined number of examples and replaced by a new one. Newly generated classifiers are excluded from this elimination process for a predefined time. Incoming examples are exclusively classified by the classifier with the highest accuracy in the past. This intuitive approach performed well within incremental and abrupt drift scenarios.
In [18] Bifet et al. propose to increase the randomization of Online Bagging [19] and thereby the diversity of the ensemble. This is done by a rather high λ value for the Poisson distribution and the usage of output detection codes. Additionally, ADWIN is used as change detector for every ensemble member. In case of a detected change, the worst classifier is replaced by a new one. The resulting active-passive method Leveraging Bagging (LVGB) achieves accurate results on streaming data containing drift.
Learn++.NSE [5] processes incoming examples in chunks with a predefined size. A base classifier is trained for each chunk and added to the ensemble. The weight of each member is determined by averaging the loss on recent chunks is with a sigmoid function. Similar to AdaBoost [20], instances are weighted such that misclassified inputs have a higher impact on the calculated loss. In contrast to other methods, members are not trained further, but preserve their initial state. In case of reoccurring drift, they can be revived to enhance the adaptation.
Due to its inherently generative nature, prototype-based models have been used in a variety of scenarios where (mostly virtual) drift is present, displaying excellent abilities to manage the stability-plasticity dilemma which occurs in this context. As before, chunking provides excellent behavior as well as considerable computational speed up [21], [22]. A method which combines concept drift adaptation with rejections was recently proposed in [23].
Passive approaches can deal with incremental drift, but their inherent adaptation speed has to be adjusted to the task specific drift speed. In case of abrupt drift, the adaptation delay is usually more pronounced than in active methods. Even though ensembles allow more flexibility in form of addition and deletion of new members, the fundamental issues remain, albeit somewhat weakened. Our approach is also an ensemble, but it distinguishes itself by the explicit generation of dedicated classifiers for current and past knowledge. Members are neither added nor removed, instead the ensemble is used to switch the knowledge basis for current predictions. This allows flexible adaptation to real and virtual drift at various rates without any parametrization.
We couple the proposed architecture with the kNN classifier, as it is one of the most popular nonparametric models. kNN has already been applied in the streaming scenario, mainly with the goal to provide an efficient search [24] or a compressed representation [25]. It was also considered for drifting data streams [13], [26]. However, one of its key advantages, the simple editing of the data, has not been exploited as extensively as in our method, enabling the preservation of more relevant data for future prediction.
To provide a more detailed comparison of our method, we tackle the common problem of unknown drift characteristics in real-world datasets. To the best of our knowledge, this has not been done so far in the literature. Nonetheless, active drift approaches such as ADWIN clearly could be used to distinguish between drift and no drift datasets, depending on the amount as well as the magnitude of detected changes. However, we propose two tests not only able to detect drift, but also its type and degree. Hence, significantly more information about the drift characteristics is provided by our approach.

IV. Architecture of the Self Adjusting Memory(SAM)

In the research field of human memory the dual-store model [27], consisting of the Short-Term and Long-Term memory (STM & LTM), is largely accepted. Sensory information arrives at the STM and is joined by context relevant knowledge from the LTM. Information getting enough attention by processes such as active rehearsal is transferred into the LTM in form of Synaptic Consolidation [28]. The capacity of the STM is quite limited and information is kept up to one minute, whereas the LTM is able to preserve it for years [29]. Immediate processing e.g. remembering the beginning of a read sentence, largely uses the STM. Whereas knowledge recalled from the past either explicitly requires the LTM e.g. consciously remembering events in life, or in an implicit way e.g. how to ride a bike.
The SAM architecture is partly inspired by this model and exhibits the following analogies:

- Explicit separation of current and past knowledge, stored in dedicated memories.
- Different conservation spans among the memories.
- Transfer of filtered knowledge from the STM to the LTM.
- Situation dependent usage.

The basic idea is to combine dedicated models for the current concept $P_t(\mathbf{x}, y)$ and all former ones $P_{t-1}(\mathbf{x}, y), \ldots, P_1(\mathbf{x}, y)$ in such a way that the prediction accuracy is maximized. This is a very general concept, which could be coupled with different type of models, requiring a different realization. In case of dedicated parametric models, for example, a proper management of the distribution parameters would be necessary. However, in this paper we implement our approach with the

[1]The VFDT is often called Hoeffding Tree (HT)

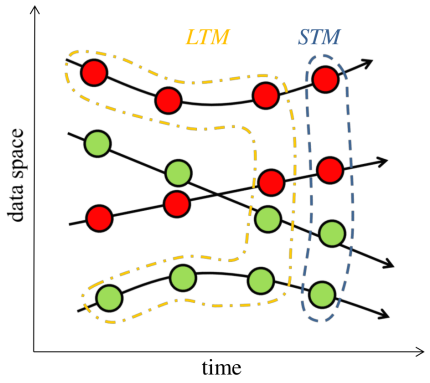

Fig. 1. Illustration of the general approach. Each column represents one concept, different colors encode different classes. The STM contains only the current concept, while the LTM preserves only knowledge which is consistent in regard to the STM.

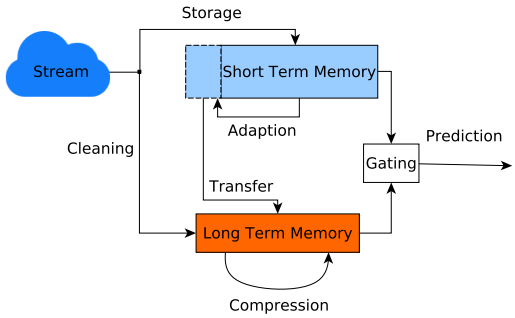

Fig. 2. SAM architecture: Incoming examples are stored within the STM. The cleaning process keeps the LTM all-time consistent with the STM. Whenever, the STM is reduced in size, its discarded knowledge is transfered into the LTM. Accumulated knowledge is compressed each time the available space is exhausted. Both models are considered during prediction, depending on their past performances.

non-parametric method kNN and therefore construct two different memories: The Short-Term Memory (STM), containing data of the current concept, and the Long-Term Memory (LTM), maintaining knowledge of past concepts. Figure 1 illustrates this approach. We share the general assumption of new data being more relevant for current predictions. Hence, we remove those information from former concepts which is in conflict with the current one, but we explicitly preserve the rest in compressed fashion. We avoid any parametrization, by exploiting the minimization of the error on recent data at various steps. Our architecture is depicted in Figure 2 and described below in detail.

A. Model definition

Memories are represented by sets M_{ST}, M_{LT}, M_{C}. Each memory is a subset in $\mathbb{R}^n \times \{1,\dots,c\}$ of varying length, adjusted during the adaptation process. The STM represents the current concept and is a dynamic sliding window containing the most recent m examples of the data stream:

$$M_{\text{ST}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{1,\dots,c\} \,|\, i = t-m+1,\dots,t\}. \quad (2)$$

The LTM preserves all former information which is not contradicting those of the STM in a compressed way. In contrast to the STM, the LTM is neither a continuous subpart of the data stream nor given by exemplars of it, but instead a set of p points:

$$M_{\text{LT}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{1,\dots,c\} \,|\, i = 1,\dots,p\}.$$

The combined memory (CM) is the union of both memories with size $m + p$:

$$M_{\text{C}} = M_{\text{ST}} \cup M_{\text{LT}}.$$

Every set induces a classifier, in our case a distance weighted kNN : $\mathbb{R}^n \mapsto \{1,\dots,c\}$, $\text{kNN}_{M_{\text{ST}}}$, $\text{kNN}_{M_{\text{LT}}}$, $\text{kNN}_{M_{\text{C}}}$. The kNN function assigns a label for a given point $\mathbf{x}$ based on a set $Z = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{1,...,c\} \,|\, i = 1,\dots,n\}$:

$$\text{kNN}_Z(\mathbf{x}) = \arg\max_{\hat{c}} \{ \sum_{x_i \in N_k(\mathbf{x},Z) | y_i = \hat{c}} \frac{1}{d(\mathbf{x}_i, \mathbf{x})} \quad |\hat{c} = 1,\dots,c\},$$

where $d(\mathbf{x}_1, \mathbf{x}_2)$ is the Euclidean distance between two points $\mathbf{x}_1$, $\mathbf{x}_2$ and $N_k(\mathbf{x}, Z)$ returns the set of k nearest neighbors of $\mathbf{x}$ in Z. Weights w_{ST}, w_{LT}, w_C are representing the accuracy of the corresponding model on the current concept and are determined as described in section IV-B4.
The prediction of our complete model relies on the sub-model with the highest weight[1] and is defined for a given point $\mathbf{x}$ as:

$$\mathbf{x} \mapsto \begin{cases} \text{kNN}_{M_{\text{ST}}}(\mathbf{x}) & \text{if } w_{\text{ST}} \geq \max(w_{\text{LT}}, w_{\text{C}}) \\ \text{kNN}_{M_{\text{LT}}}(\mathbf{x}) & \text{if } w_{\text{LT}} \geq \max(w_{\text{ST}}, w_{\text{C}}) \\ \text{kNN}_{M_{\text{C}}}(\mathbf{x}) & \text{if } w_{\text{C}} \geq \max(w_{\text{ST}}, w_{\text{LT}}). \end{cases}$$

This model is adapted incrementally for every time t as described in section IV-B.

Model parameter: During the adaptation phase we adjust the following parameters:

- The size m of the STM.
- The data points in the LTM.
- The weights w_{ST}, w_{LT}, w_{C}.

The model has the subsequent hyperparameters, which can be robustly chosen and do not require a task specific setting:

- The number of neighbors k.
- The minimum length L_{min} of the STM.
- The maximum number of stored examples L_{max} (STM and LTM combined).

We used for all experiments the same values which underlines the robustness of our approach. The number of neighbors was set to $k = 5$, a common default value in various implementations [30]. The minimum length was set to $L_{\text{min}} = 50$ which is usually large enough to robustly estimate the current error on the one hand and still allows a quick reaction to drift on the other. We set $L_{\text{max}} = 5000$ as a reasonable trade-off between a fast run-time / low memory consumption and a high degree of freedom for the approach to prove its qualities. Please note that a too restrictive size makes a precise evaluation of the system's adaptation capabilities impossible.

[1] In case of ties, we prioritize the models in the following order: $\text{kNN}_{M_{\text{ST}}}$, $\text{kNN}_{M_{\text{LT}}}$, $\text{kNN}_{M_{\text{C}}}$

B. Model adaptation

The adaptation comprises every memory as well as the corresponding weights. We denote a data point at time t as $(\mathbf{x}_t, y_t)$ and the corresponding memories M_{ST_t}, M_{LT_t}, M_{C_t}.

1) Adaptation of the Short Term Memory: The STM is a dynamic sliding window containing the most recent examples. Every incoming example of the stream gets inserted such that the STM grows continuously. Its role is to exclusively contain data of the current concept. Therefore, its size has to be reduced whenever the concept changes such that examples of the former concept are dropped. However, we do not explicitly detect a concept change. Instead, we adjust the size such that the Interleaved Test-Train error of the remaining STM is minimized. This approach relies on the fact that a model trained on internally consistent data yields fewer errors. We assume that the remaining instances represent the current concept or are sufficiently "close" to it.
Formally, we evaluate differently sized STMs and adopt the one with minimum Interleaved Test-Train error E (see Equation 1). We use bisection to compare only a logarithmic number of windows. Tested windows are:

$$M_l = \{(\mathbf{x}_{t-l+1}, y_{t-l+1}), \ldots, (\mathbf{x}_t, y_t)\},$$

where $l \in \{m, m/2, m/4 \ldots\}$ and $l \geq L_{\min}$.

$$M_{\mathrm{ST}_{t+1}} = \underset{S \in \{M_m, M_{m/2}, \ldots\}}{\arg\min} E(S).$$

Whenever the STM is shrunk, the set of discarded examples O_t is defined as

$$O_t = M_{\mathrm{ST}_t} \setminus M_{\mathrm{ST}_{t+1}}. \tag{3}$$

Instead of the commonly used cross validation error, we rely on the Interleaved Test-Train error because it has various advantages in the streaming setting. The former is applied on random splits of the data and requires multiple repetitions to deliver a stable estimation of the error, which significantly increases the computational cost. Whereas the latter uses every example for test and training in the original order and therefore is the natural choice in the incremental learning setting.
Our way to adapt the size of the STM has similarities with [15]. However, the approach is based on SVM specific estimates of the leave-one-out error and the authors do not indicate how to choose evaluated window sizes. In contrast, we propose to directly determine the Interleaved Test-Train error on recursively bisected windows, which is applicable for arbitrary models.

2) Cleaning and Transfer: The LTM contains all data of former concepts that is consistent with the STM. This requires a cleaning of the LTM according to every seen example. Furthermore, whenever the STM is reduced in size, we do not simply discard the sorted out data, since it still may contain valuable information for future prediction. In case of reoccurring drift, methods preserving past knowledge do not have to relearn former concepts and therefore produce fewer errors.
Instead, we transfer as much knowledge as possible into the LTM. Before doing so, we delete examples from the separated set O_t (see Equation 3) which are contradicting those in $M_{ST_{t+1}}$.
This adaptation is formalized by two operations.

1) We *clean* a set A by another set B regarding an example $(\mathbf{x}_i, y_i) \in B$

$$\text{clean} : (A, B, (\mathbf{x}_i, y_i)) \mapsto \hat{A}$$

where $A, B, \hat{A} \subset \mathbb{R}^n \times \{1, \ldots, c\}$ and $(\mathbf{x}_i, y_i) \in B$. $\hat{A}$ is defined in two steps.
(1) We determine the k nearest neighbors of $\mathbf{x}_i$ in $B \setminus (\mathbf{x}_i, y_i)$ and select the ones with label y_i. These define the threshold

$$\theta = \max\{d(\mathbf{x}_i, \mathbf{x}) | \mathbf{x} \in N_k(\mathbf{x}_i, B \setminus (\mathbf{x}_i, y_i)), y(\mathbf{x}) = y_i\}.$$

(2) The k nearest neighbors of $\mathbf{x}_i \in A$ which are inconsistent to B are cleaned based on the threshold, yielding the result of this operation:

$$\hat{A} = A \setminus \{(\mathbf{x}_j, y(\mathbf{x}_j)) | \mathbf{x}_j \in N_k(\mathbf{x}_i, A), d(\mathbf{x}_j, \mathbf{x}_i) \leq \theta, y(\mathbf{x}_j) \neq y_i\}.$$

2) Furthermore, we require a *cleaning operation* for the full set B

$$\text{clean} : (A, B) \mapsto \hat{A}_{|B|}$$

where $A, B, \hat{A}_{|B|} \subset \mathbb{R}^n \times \{1, \ldots, c\}$. This is defined iteratively by applying the former cleaning for all $(\mathbf{x}_i, y_i) \in B = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{|B|}, y_{|B|})\}$ as

$$\begin{aligned} \hat{A}_0 &= A \\ \hat{A}_{t+1} &= \text{clean}(\hat{A}_t, B, (\mathbf{x}_{t+1}, y_{t+1})). \end{aligned}$$

The adaptation of the LTM takes place at two different steps. To ensure a consistent model at any time, cleaning takes place according to every incoming sample $(\mathbf{x}_t, y_t)$

$$\tilde{M}_{\mathrm{LT}_t} = \text{clean}(M_{\mathrm{LT}_t}, M_{\mathrm{ST}_t}, (\mathbf{x}_t, y_t)).$$

Whenever the STM is shrunk, the discarded set O_t is transferred into the LTM after cleaning, i.e. the LTM becomes

$$M_{\mathrm{LT}_{t+1}} = \begin{cases} \tilde{M}_{\mathrm{LT}_t} \cup \text{clean}(O_t, M_{\mathrm{ST}_{t+1}}) & \text{if STM is shrunk} \\ \tilde{M}_{\mathrm{LT}_t} & \text{otherwise} \end{cases}$$

3) Compression of the LTM: In contrast to the FIFO principle of the STM, instances are not fading out as soon as the size limit of the LTM is reached. Instead, we condense the available information to a sparse knowledge representation via clustering. This enables a far longer conservation than possible with simple out fading. Formally, for every class label $\hat{c}$ we group the corresponding data points in the LTM

$$M_{\mathrm{LT}_{\hat{c}}} = \{\mathbf{x}_i | (\mathbf{x}_i, \hat{c}) \in M_{\mathrm{LT}}\}.$$

We use the clustering algorithm kMeans++[1] with $|M_{LT_{\hat{c}}}|/2$ clusters. The resulting prototypes $\hat{M}_{LT_{\hat{c}}}$ represent the compressed original data. The LTM is given by the union of all prototypes

$$M_{\mathrm{LT}} = \bigcup_{\hat{c}} \{(\mathbf{x}_i, \hat{c}) | \mathbf{x}_i \in \hat{M}_{LT_{\hat{c}}}\}$$

This process is repeated each time the size limit is reached leading to a self-adapting level of compression.

[1] We used kMeans++ [31] because of its efficiency and scalability to larger datasets.

4) Model weight adaptation: The weight of a memory is its accuracy averaged over the last m_t samples, where $m_t = |M_{\text{ST}_t}|$ is the size of the current STM. Hence, the weight of the LTM at time stamp t equals

$$w_{\text{LT}}^t = \frac{|\{i \in \{t-m_t+1,\ldots,t\} \,|\, \text{kNN}_{M_{\text{LT}_i}}(\mathbf{x}_i) = y_i\}|}{m_t}$$

and analogous for STM and CM.

C. Time Complexity

The adaptation of the STM is by far the most time consuming process of SAM and determines the upper bound of the complexity. Therefore, we exclusively give a complexity analysis of this part and neglect the others.
Given a query point, the determination of its k nearest neighbors within a pool of m examples is linear $\mathcal{O}(m)$. Each time, we are evaluating up to $\log \frac{L_{\text{max}}}{L_{\text{min}}}$ differently sized STMs, where L_{min} and L_{max} are the minimum and maximum lengths of the STM. The complete calculation of the error for one STM is upper bounded by $\mathcal{O}(L_{\text{max}}^2)$. This results in the overall worst case complexity of $\mathcal{O}(nL_{\text{max}}^2 \log \frac{L_{\text{max}}}{L_{\text{min}}})$ for n examples. There is a lot of room to reduce this complexity, for instance, the calculation of the error can be done incrementally whenever the following condition is met:
Given an exemplary STM S_{t-1} (defined as in equation 2). If its successor S_t is simply an extension of S_{t-1}, such that $S_t = [S_{t-1}, (\mathbf{x}_t, y_t)]$, the corresponding error is given by

$$E(S_t) = \frac{(t-1)E(S_{t-1}) + \mathbb{1}(h_{t-1}(\mathbf{x}_t) \neq y_t)}{t}.$$

This is the case whenever an STM simply is growing by the current example, which happens frequently in practice and clearly dwindles the time complexity of the method.
We are currently experimenting with approximations of the error to reduce the complexity further.

V. Datasets

We used well known artificial as well as real-world datasets for the experiments. Links to all datasets and algorithms, including our own, are available at GitHub[1]. In the following we describe the data more detailed.

A. Artificial Data

Artificial datasets have the advantage that any desired drift behavior can be explicitly simulated. They are often 2-dimensional to enable a straightforward visualization. We considered published benchmarks or used generators from Massive Online Analysis (MOA) [32] with common parametrization. We also added four new datasets allowing an extensive evaluation on specific drift types, including virtual drift, which is often ignored in the community. Table I shows their main characteristics.

TABLE I. Evaluated artificial datasets. Section VI describes how we obtained the drift degree.

				Drift properties		
Dataset	#Samples	#Feat.	#Class	Degree	Rate	Type
SEA Concepts	50K	3	2	low	abrupt	real
Rotating Hyperplane	200K	10	2	low	incremental	real
Moving RBF	200K	10	5	high	incremental	real
Interchanging RBF	200K	2	15	high	abrupt	real
Moving Squares	200K	2	4	high	incremental	real
Transient Chessboard	200K	2	8	high	abr. reoccuring	virtual
Mixed Drift	600K	2	15	high	abr./incr./reoc.	real/virtual

SEA Concepts This dataset was proposed in [33] and consists of 50000 instances with three attributes of which only two are relevant. The two class decision boundary is given by $f_1 + f_2 = \theta$, where f_1, f_2 are the two relevant features and θ a predefined threshold. Abrupt drift is simulated with four different concepts, by changing the value of θ every 12500 samples. Also included are 10% of noise.

Rotating Hyperplane A hyperplane in d-dimensional space is defined by the set of points x that satisfy $\sum_{i=1}^{d} w_i x_i = w_0$. The position and orientation of the hyperplane are changed by continuous addition of a term δ to the weights $w_i = w_i + \delta$. We used the Random Hyperplane generator in MOA with the same parametrization as in [13] (10 dimensions, 2 classes, δ=0.001).

Moving RBF Gaussian distributions with random initial positions, weights and standard deviations are moved with constant speed v in d-dimensional space. The weight controls the partitioning of the examples among the Gaussians.
We used the Random RBF generator in MOA with the same parametrization as in [13] (10 dimensions, 50 Gaussians, 5 classes, v=0.001).

Interchanging RBF Fifteen Gaussians with random covariance matrices are replacing each other every 3000 samples. Thereby, the number of Gaussians switching their position increases each time by one until all are simultaneously changing their location. This allows to evaluate an algorithm in the context of abrupt drift with increasing strength. Altogether 67 abrupt drifts are occurring within this dataset.

Moving Squares Four equidistantly separated, squared uniform distributions are moving in horizontal direction with constant speed. The direction is inverted whenever the leading square reaches a predefined boundary. Each square represents a different class. The added value of this dataset is the predefined time horizon of 120 examples before old instances may start to overlap current ones. This is especially useful for dynamic sliding window approaches, allowing to test whether the size is adjusted accordingly.

Transient Chessboard Virtual drift is generated by revealing successively parts of a chessboard. This is done square by square randomly chosen from the whole chessboard such that each square represents an own concept. Every time after four fields have been revealed, samples covering the whole chessboard are presented. This reoccurring alternation penalizes algorithms tending to discard

[1] https://github.com/vlosing

TABLE II. CONSIDERED REAL-WORLD DATASETS. THE DRIFT CHARACTERISTICS ARE UNKNOWN FOR REAL-WORLD DATASETS. THEREFORE, WE USED TESTS AS DESCRIBED IN SECTION VI TO OBTAIN THEM.

Dataset	#Samples	#Feat.	#Class	Drift properties Degree	Type
Weather	18159	8	2	low	virtual
Electricity	45312	5	2	high	real
Cover Type	581012	54	7	low	real
Poker Hand	829200	10	10	low	virtual
Outdoor	4000	21	40	high	virtual
Rialto	82250	27	10	high	virtual

former concepts. To reduce the impact of classification by chance we used eight classes instead of two.

Mixed Drift The datasets Interchanging RBF, Moving Squares and Transient Chessboard are arranged next to each other and samples of these are alternately introduced. Therefore, incremental, abrupt and virtual drift are occurring at the same time, requiring a local adaptation to different drift types.

B. Real World Data

A few real-world drift benchmarks are available of which we considered the largest ones. We contribute two new challenging datasets obtained from visual data. The characteristics are given in Table II.

Weather Elwell et al. introduced this dataset in [5]. In the period of 1949-1999 eight different features such as temperature, pressure, wind speed etc. were measured at the Offutt Air Force Base in Bellevue, Nebraska. The target is to predict whether it is going to rain on a certain day or not. The dataset contains 18159 instances with an imbalance towards no rain (69%).

Electricity market dataset This problem is often used as a benchmark for concept drift classification. Initially described in [34], it was used thereafter for several performance comparisons [35], [36], [13], [3]. A critical note to its suitability as a benchmark can be found in [37]. The dataset holds information of the Australian New South Wales Electricity Market, whose prices are affected by supply and demand. Each sample, characterized by attributes such as day of week, time stamp, market demand etc., refers to a period of 30 minutes and the class label identifies the relative change (higher or lower) compared to the last 24 hours.

Forest Cover Type Assigns cartographic variables such as elevation, slope, soil type, ... of 30×30 meter cells to different forest cover types. Only forests with minimal human-caused disturbances were used, so that resulting forest cover types are more a result of ecological processes. It is often used as a benchmark for drift algorithms [13], [38], [39].

Poker Hand One million randomly drawn poker hands are represented by five cards each encoded with its suit and rank. The class is the resulting poker hand itself such as one pair, full house and so forth. This dataset has in its original form no drift, since the poker hand definitions do not change and the instances are randomly generated. However, we used the version presented in [13], in which virtual drift is introduced via sorting the instances by rank and suit. Duplicate hands were also removed.

Fig. 3. Objects, positioned in a garden, are approached by a mobile robot under different lighting conditions. Each row shows the first, fifth and tenth image of one approach.

Outdoor Objects We obtained this dataset from images recorded by a mobile in a garden environment [40]. The task is to classify 40 different objects, each approached ten times. under varying lighting conditions affecting the color based representation (see Figure 3). Each approach consists of 10 images and is represented in temporal order within the dataset. The objects are encoded in a normalized 21-dimensional RG-Chromaticity histogram.

Rialto Bridge Timelapse Ten of the colorful buildings next to the famous Rialto bridge in Venice are encoded in a normalized 27-dimensional RGB histogram. We obtained the images from time-lapse videos captured by a webcam with fixed position. The recordings cover 20 consecutive days during may-june 2016. Continuously changing weather and lighting conditions affect the representation as can be seen in Figure 4. We generated the labels by manually masking the corresponding buildings and excluded overnight recordings since they were too dark for being useful.

VI. DETERMINATION OF THE DRIFT CHARACTERISTICS IN REAL WORLD DATA SETS

Before investigating the performance of the SAM architecture for various data sets, we analyze the necessity of such a technology for realistic settings. More precisely, we investigate whether drift is present in popular real life benchmarks for drift detection and, if so, which types of it. While drift is explicitly generated in artificial datasets, it is rather difficult to identify in real-world data. The common approach is simply to reason about whether one expects drift being present due to domain knowledge. However, any knowledge about the drift characteristics is crucial. From the scientific point of view, it enables a more detailed analysis of drift algorithms. Furthermore, it is not even clear whether commonly used real-world benchmarks contain any drift at all and are, therefore, rightly used for evaluation. In terms of practical use, information about the drift

Fig. 4. Images of the Rialto bridge recorded at different times of day. The color based representation of the buildings changes significantly during each of the 20 recorded days. Source of the images: https://www.skylinewebcams.com.

characteristics clearly facilitates the choice of an appropriate algorithm for a given task. For instance, whenever no real drift is present at all, classical incremental / online algorithms such as Naive Bayes or incremental decision trees may be the better choice in terms of performance and / or efficiency.

We propose a method to determine the drift type, real or virtual, and its degree in a given dataset. To the best of our knowledge, this has not been done so far in literature. The proposed drift tests process data in online fashion, however, multiple passes are required. In case of very large or potentially infinite streams, we assume a sufficiently large subpart to be available, containing similar drift characteristics as the whole stream.

Our approach consists of two consecutive tests. The first detects real drift. If that is not the case, we apply the second one to test for virtual drift. No drift is present whenever both tests are negative. We use sliding windows of various sizes applied on bootstrap samples [41] of the dataset and analyze the resulting classification errors for significant deviations. The degree of drift is both times inferred from the magnitude of the error differences. Even though the type of classifier coupled with the sliding windows is exchangeable, we utilize kNN once again.

A. Prerequisites for the Drift Tests

Assume a representative set of streaming samples is given $S = (s_1, s_2, \ldots, s_n)$ with n tuples $s_i = (\mathbf{x}_i, y_i)$, whereby n is chosen as large as possible, respecting computational restrictions.[1] The tests heavily rely on bootstrap samples from the given data. In the following, we will use two types of bootstrap samples: Samples $B_1, \ldots, B_m$ with $|B_i| = |S|$ are randomly sampled from S with replacement with *random ordering* of the samples. In contrast, $\tilde{B}_1, \ldots, \tilde{B}_m$ with $|\tilde{B}_i| = |S|$ are randomly sampled from S with replacement whereby the *ordering of the samples as present in S is preserved*.

For every such data stream B_i and $\tilde{B}_i$, we evaluate the behavior of classifiers (as already mentioned, we use kNN again) induced by sliding windows of different size. We consider a small number of different representative sliding window sizes $W = \{w_1, \ldots, w_b\}$, where b is usually small for computational efficiency and to prevent false positives of the tests[2]. Given a window size w and data stream $(s_1, s_2, \ldots, s_n)$, a sliding window induces kNN models $h_t = \text{kNN}_{\{s_{t-w+1}, \ldots, s_t\}}$ and a corresponding Interleaved test-train error. We refer to the errors which are induced by models of window size w and the data stream B_i as e_i^w, and for data stream $\tilde{B}_i$ as $\tilde{e}_i^w$, respectively.

B. Test for Real Drift

This test is inspired by the observation that whenever real drift is present, an algorithm using a sliding window approach tends to make fewer errors with smaller windows than with very large ones. This contradicts the classical assumption for i.i.d. datasets as stated in the PAC model [42]: The error rate of a consistent learning algorithm decreases with increasing number of examples towards the Bayes error. However, streaming data containing real drift is not i.i.d. More mistakes are done when outdated instances are in conflict with examples of the current concept. In such a case, small windows contain less outdated instances and deliver better results in comparison to large ones.

Within the test, the behavior of large windows is evaluated by a reference model $\widehat{\text{kNN}}_t := \text{kNN}_{\{s_{t-\hat{w}+1}, \ldots, s_t\}}$. This model is coupled with a window size $\hat{w}$, chosen as large as possible regarding computational restrictions[3]. We compare its classification error to the behavior of models with smaller window size w, as evaluated by $\tilde{e}_i^w$. More precisely, we test whether any test model yields a significantly lower error rate than the reference model. We refer to the error of the reference model $\widehat{\text{kNN}}$ on the bootstrap sample $\tilde{B}_i$ as $\tilde{r}_i$. We use a one sided hypothesis test with an α-value of 1%. For a given window size w, the null hypothesis is that the error of the reference model $\tilde{r}$ is smaller than or equal to the error $\tilde{e}^w$ of the sliding window kNN with size w. We determine the first percentile β_w of the error differences $\{\tilde{r}_i - \tilde{e}_i^w | i \in \{1, \ldots, m\}\}$. The null hypothesis is rejected as soon as $\beta_w > 0$. If this is the case for any window size w, we infer that real drift is contained within the dataset.

C. Test for Virtual Drift

In case of no present real drift, we proceed with the test for virtual drift. Here, we test whether the error rate of any test model is lower for the ordered bootstrap samples in comparison to the unordered ones. Intuitively, randomizing the temporal order of virtual drift datasets leads to a higher error rate because of the following reasons: If $P(\mathbf{x})$ is changing, one concept is processed after another, making the problem less challenging than learning the whole distribution at once. Furthermore, the representation of each concept is richer, since the space of the sliding window is largely used for one concept at a time instead of being partitioned among all of them.

For a given model with a sliding window size w, the null hypothesis is the error e^w being smaller or equal to $\tilde{e}^w$. Analogous to the previous test, we determine the first percentile β_w of the error differences $\{e_i^w - \tilde{e}_i^w | i \in (1, \ldots, m)\}$. We infer the presence of virtual drift if for any model $\beta_w > 0$.

[1] We used the complete benchmarks in our experiments but for real-world tasks this is often unrealistic.

[2] We used three test models with $W = \{500, 1000, 5000\}$ in all tests.

[3] We consistently used $\hat{w} = 20000$.

D. Drift Degree

Whenever one of the tests is positive, we estimate the corresponding drift degree. The degree simply correlates with the magnitude of the relative error differences. An error fraction ψ is used to distinguish between low and high degrees of drift. ψ is defined differently for both tests:

1) Real Drift: Regarding the real drift test, ψ is the minimum error fraction of all test models in comparison to the reference model:

$$\psi = \arg\min_{w \in W} \frac{\sum_{i=1}^{m} \tilde{e}_i^w}{\sum_{i=1}^{m} \tilde{r}_i}.$$

2) Virtual Drift: In case of the virtual drift test, ψ is defined as the minimum fraction between the error achieved on the ordered and unordered bootstrap samples.

$$\psi = \arg\min_{w \in W} \frac{\sum_{i=1}^{m} \tilde{e}_i^w}{\sum_{i=1}^{m} e_i^w}.$$

To estimate the degree, we simply check whether ψ is smaller than γ:

$$\psi \mapsto \begin{cases} \text{high} & \text{if } \psi < \gamma \\ \text{low} & \text{otherwise.} \end{cases}$$

E. Results of the Drift Tests

Apart from real-world datasets, the tests were also applied on artificial benchmarks as a proof of concept. Furthermore, we also perform tests on data without any drift at all. One of them is *Chessboard i.i.d.*, which simply is a random permutation of the *Transient Chessboard* dataset. The other one is the popular *MNIST* [43] dataset.
We generated 500 bootstrap samples per dataset, but the results are often unambiguous such that a considerably smaller amount would be sufficient for robust inference. The reference model $\widehat{\text{kNN}}$ was coupled with a sliding window $\hat{w} = 20000$ samples, whereas the test models had windows of 500, 1000 and 5000 samples. The threshold γ was set to 0.85 for both tests resulting in a *high* drift degree as soon as the relative mean error difference is larger than 15%.

1) Real Drift Test: All three test models were tested with an α-value of 1%. Under the union bound this results in a total α-value of 3% for the complete real drift test. Table III shows the results. In all artificial datasets real drift is correctly detected. Also its absence in the datasets *Transient Chessboard*, *Chessboard i.i.d.* and *MNIST* is accurately inferred. The artificial real drift datasets *SEA Concepts* and *Rotating Hyperplane* are classified to have a low drift degree. This is reasonable, since within *SEA Concepts* there are only three minor abrupt changes, having a rather small impact on the accuracy. Regarding *Rotating Hyperplane*, the incremental change is very slow and slightly affects the performance. All remaining artificial datasets have a high drift degree since they show substantial error rate deviations. Figure 5 exemplary depicts some histograms of error differences.
Our results confirm the common assumption of real-world tasks having only little amounts of real concept drift. Instead, most of them contain virtual drift. Only the *Electricity* and *CoverType* tasks incorporate real drift but in a lower degree than the artificial benchmarks.

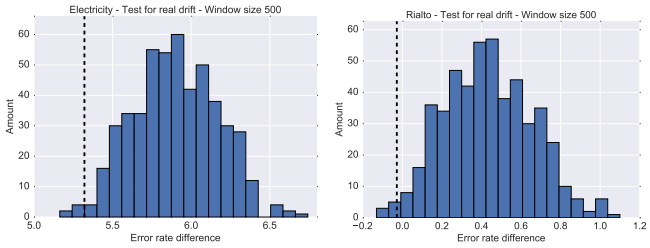

Fig. 5. Exemplary histograms of error rate differences achieved on 500 ordered bootstrap samples. The differences shown here were obtained between the reference model using a sliding window of 20000 samples and a test model with 500 samples. The test is positive whenever the first percentile of the distribution, illustrated by the dashed line, is larger than 0. This is true for the *Electricity* dataset (left) but not for *Rialto*. The first percentile of *Rialto* is just barely below 0, indicating a small but insignificant amount of real drift.

Figure 6 illustrates the error rate course of differently sized sliding windows. In the case of real drift (*Interchanging RBF*), the error rate of larger windows is distinctly higher. It is the other way around when no real drift is contained (*Transient Chessboard*).

2) Virtual Drift Test: The overall α-value for the virtual drift test was also 3%. Table IV shows the results of the virtual drift detection. The test correctly identifies virtual drift within the *Transient Chessboard* as well as its non-existence in *Chessboard i.i.d.* and *MNIST*. *Weather* contains a very low degree of virtual drift, whereas all others incorporate a high one.
For all datasets with a priori known drift type the tests reliably detect real and virtual drift as well as their absence. However, the tests were not able to infer that the *Mixed Drift* dataset contains next to the detected real drift also virtual one. This issue is extensively discussed in section VI-F. Figure 7 shows exemplary distributions of the error differences.

F. Limitations of the Drift Tests

One disadvantage of the method is its heavy dependence on properly chosen sliding window sizes. The measured error differences crucially depend on the drift speed as well as the complexity of the task and are only properly reflected in adequate window sizes. In case of the real drift test, too large windows may not lead to significant error differences in comparison to the reference model, even though the tested dataset has a high degree of drift. Since the drift degree is estimated according to the magnitude of error differences, the choice of the window sizes naturally affects this result, too. This requires prior knowledge, which is usually not available. In practice, however, very slow changes are rather irrelevant because they have a small impact in a system which learns continuously from the stream. Therefore, small sizes should be preferably tested to correctly detect fast changes. Furthermore, the number of tested sizes has to be strictly limited to minimize the probability of false positives. One way to test more window-sizes without increasing the risk of false positives is to accordingly adapt the α-value see, e.g. Bonferroni correction [44]. Nonetheless, we showed in our experiments that the size of evaluated windows can be robustly chosen across multiple datasets as long as they cover a reasonable range.
The significance requirement naturally prevents the method to

TABLE III. MEAN ERROR RATES AND CORRESPONDING STANDARD DEVIATIONS ACHIEVED BY KNN CLASSIFIER WITH DIFFERENTLY SIZED SLIDING WINDOWS ON ORDERED BOOTSTRAP SAMPLES. A DATASET IS TESTED POSITIVE FOR REAL DRIFT AS SOON AS THE FIRST PERCENTILE OF THE DISTRIBUTION OF ERROR DIFFERENCES BETWEEN REFERENCE AND ANY TEST MODEL IS LARGER THEN 0. ENTRIES LEADING TO THE HIGHEST POSITIVE FIRST PERCENTILE β_w ARE MARKED BOLD. NATURALLY, THIS STRONGLY CORRELATES WITH THE HIGHEST DIFFERENCES BETWEEN THE MEAN VALUES. WE OMMITED THE FIRST PERCENTILE VALUES OF EACH MODEL FOR THE SAKE OF CLARITY.

	Dataset	kNN_{500}	kNN_{1000}	kNN_{5000}	$\widehat{kNN}_{20000}$	Real drift	Degree	ψ
Artificial	SEA Concepts	15.58(.21)	14.95(.21)	**14.70(.21)**	16.99(.20)	✓	low	.87
	Rotating Hyperplane	20.96(.11)	19.58(.11)	**17.52(.11)**	17.72(.12)	✓	low	.99
	Moving RBF	**13.35(.08)**	13.72(.07)	21.16(.09)	28.09(.10)	✓	high	.48
	Interchanging RBF	**5.39(.03)**	9.82(.05)	36.91(.12)	56.32(.12)	✓	high	.10
	Moving Squares	**47.50(.10)**	52.95(.10)	60.96(.11)	59.94(.13)	✓	high	.80
	Transient Chessboard	17.83(.07)	17.13(.07)	8.02(.06)	4.88(.05)	✗	-	-
	Mixed Drift	**13.45(.03)**	20.38(.05)	27.35(.05)	36.33(.07)	✓	high	.37
	Chessboard i.i.d.	25.48(.09)	17.72(.08)	8.05(.07)	4.33(.05)	✗	-	-
Real World	Weather	22.09(.39)	21.76(.39)	21.21(.35)	21.09(.43)	✗	-	-
	Electricity	**22.82(.20)**	23.67(.21)	26.47(.26)	28.72(.25)	✓	high	.79
	Cover Type	6.84(.03)	**4.38(.02)**	4.54(0.03)	4.71(0.02)	✓	low	.92
	Poker Hand	19.69(.04)	17.90(.04)	17.72(.05)	14.48(.05)	✗	-	-
	Outdoor	22.39(.38)	20.43(.43)	-	19.14(.47)	✗	-	-
	Rialto	25.24(.13)	25.61(.16)	26.56(.19)	25.79(.23)	✗	-	-
	MNIST	17.32(.11)	13.34(.17)	7.53(.10)	5.25(.09)	✗	-	-

TABLE IV. MEAN ERROR RATES AND CORRESPONDING STANDARD DEVIATIONS ACHIEVED BY KNN CLASSIFIER WITH DIFFERENTLY SIZED SLIDING WINDOWS ON UNORDERED BOOTSTRAP SAMPLES. A DATASET IS TESTED POSITIVE FOR VIRTUAL DRIFT AS SOON AS THE FIRST PERCENTILE OF THE DISTRIBUTION OF DIFFERENCES BETWEEN THE ERROR ACHIEVED ON ORDERED (SEE TABLE III) AND UNORDERED BOOTSTRAP SAMPLES WITH THE SAME WINDOW SIZE IS LARGER THEN 0. ENTRIES LEADING TO THE HIGHEST POSITIVE FIRST PERCENTILE β_w ARE MARKED BOLD. WE OMITTED THE FIRST PERCENTILE VALUES FOR THE SAKE OF CLARITY.

	Dataset	kNN_{500}	kNN_{1000}	kNN_{5000}	Virtual drift	Degree	ψ
Artificial	Transient Chessboard	**25.47(.11)**	17.72(.08)	7.98(.06)	✓	high	.70
	Chessboard i.i.d.	25.51(.10)	17.74(.10)	8.04(.07)	✗	-	-
Real World	Weather	**23.05(.38)**	22.37(.39)	21.52(.37)	✓	low	.95
	Poker Hand	36.56(.05)	**34.51(.05)**	29.77(.06)	✓	high	.52
	Outdoor	**33.94(.67)**	26.39(.69)	-	✓	high	.66
	Rialto	**55.83(.20)**	49.58(.21)	36.52(.21)	✓	high	.44
	MNIST	21.20(.50)	16.41(.33)	-	✗	-	-

TABLE V. THE COMPARED ALGORITHMS AND THE CHOSEN HYPERPARAMETER. ENSEMBLES CONSISTED OF 10 MEMBERS. WE USED A MAXIMUM WINDOW SIZES OF 5000 AND 1000 SAMPLES AND k WAS SET TO 5 FOR ALL KNN BASED METHODS.

Abbr.	Classifier	Parameter
L++.NSE	Learn++.NSE with CART	chunk-size = optimized
DACC	DACC with HT	$n = 10$
LVGB	LVGB with HT	$n = 10$
kNN_S	kNN with fixed size sliding window	$L_{\max} = 5000$ & 1000, $k = 5$
kNN_{W_A}	kNN with PAW+ADWIN	$L_{\max} = 5000$ & 1000, $k = 5$
SAM	Self adjusting memory-kNN	$L_{\max} = 5000$ & 1000, $k = 5$

detect very little amounts of drift. Another weak point is its inability to infer whether virtual and real drift are contained in the data. Due to the fact that the test for virtual drift is likely to be also positive for real drift datasets, we initially test for real drift and, only if this is excluded, we can use the other test to identify virtual drift. Hence, the approach only detects real drift in datasets containing both types of drift. However, real drift usually affects the performance more severely than virtual one and and therefore is more important to detect.

VII. SAM EXPERIMENTS

We compare SAM with well-known state of the art algorithms for handling drift in streaming data. Thereby, we also utilize the previously obtained drift characteristics for the analysis. Implementations of the other algorithms were either obtained from the original authors or were already available in MOA. Table V gives an overview of the algorithms as well as the chosen hyperparameter. Apart from the methods already discussed in III, we compare against a distance weighted kNN classifier with a sliding window of fixed size. Learn++.NSE is combined with Classification and Regression Trees (CART) [45] as it is done by its author.

Window based approaches were allowed to store 5000 samples (we also report results for a size of 1000 samples) but never more than 10% of the whole dataset.[1] This rather large amount gives the approaches a high degree of freedom and prevents the concealment of their qualities with a too restricted window. The chunk size parameter of L++.NSE has to be predefined, which plays a similar role as the size of sliding windows. To avoid any disadvantage we evaluated several sizes and report the best result. No further dataset specific hyperparameter were optimized, since we wanted to provide as little prior knowledge as possible.

A. Results

We evaluated the methods by measuring the Interleaved Test-Train error. The error rates of all experiments are shown in Table VI. The proposed method SAM-kNN outperforms the others quite significantly as it almost halves the average error rate of the second best method LVGB. Even more important is the fact that other methods struggle at one or another dataset but our approach delivers consistently robust results. All drift types are handled better or at least competitively. This is particularly clarified in the large gap achieved within the Mixed Drift dataset, which contains incremental, abrupt and virtual drift at the same time.

[1] Regarding our approach, the available space is shared between the STM and LTM.

TABLE VI. INTERLEAVED TEST-TRAIN ERROR RATES OF ALL EXPERIMENTS. WINDOW BASED APPROACHES ARE CONSIDERED WITH THE MAXIMUM SIZE OF 5000 EXAMPLES FOR THE RANKING. FOR EACH DATASET THE LOWEST VALUE IS MARKED IN BOLD.

	Drift properties			Window size 5000						Window size 1000		
Dataset	Degree	Rate	Type	L++.NSE	DACC	LVGB	kNN_S	kNN_{W_A}	SAM	kNN_S	kNN_{W_A}	SAM
SEA Concepts	low	abrupt	real	14.48	15.68	**11.69**	13.83	13.39	12.50	13.96	13.22	13.44
Rotating Hyperplane	low	incremental	real	15.58	18.20	**12.53**	16.00	16.16	13.31	18.44	18.02	14.53
Moving RBF	high	incremental	real	44.50	54.34	44.84	20.36	24.04	**15.30**	12.89	17.3	12.12
Interchanging RBF	high	abrupt	real	27.52	**1.40**	6.11	45.92	8.56	5.70	10.15	8.56	2.60
Moving Squares	high	incremental	real	65.90	**1.17**	12.17	68.87	61.01	2.30	59.24	56.52	2.30
Transient Chessb.	high	abr. reoccuring	virtual	**1.98**	43.21	17.95	7.36	14.44	6.25	13.72	16.34	11.27
Mixed Drift	high	abr./incr./reoc.	real/virtual	40.37	61.06	26.29	31.00	26.75	**13.33**	20.27	23.99	12.22
Artificial ∅				30.05	27.87	18.80	29.05	23.48	**9.81**	21.24	21.99	9.78
Artificial ∅ Rank				4.00	4.57	2.86	4.29	3.57	**1.71**	-	-	-
Weather	low	-	virtual	22.88	26.78	21.89	**21.53**	23.11	21.74	22.09	23.40	22.45
Electricity	high	-	real	27.24	16.87	**16.78**	28.61	26.13	17.52	24.74	23.30	17.46
Cover Type	low	-	real	15.00	10.05	9.07	**4.21**	6.76	4.80	3.96	6.51	5.85
Poker Hand	high	-	virtual	22.14	20.97	**13.65**	17.08	27.94	18.45	17.08	27.94	19.77
Outdoor	high	-	virtual	57.80	35.65	39.97	13.98	16.30	**11.25**	14.02	16.90	11.52
Rialto	high	-	virtual	40.36	28.93	39.64	22.74	24.96	**18.58**	20.72	24.61	18.21
Real world ∅				30.90	23.21	23.50	18.03	20.87	**15.40**	17.10	20.44	15.88
Real word ∅ Rank				5.33	4.17	3.17	2.33	4.00	**2.00**	-	-	-
Overall ∅				30.44	25.72	20.97	23.96	22.27	**12.39**	19.33	21.28	12.60
Overall ∅ Rank				4.62	4.38	3.00	3.38	3.77	**1.85**	-	-	-

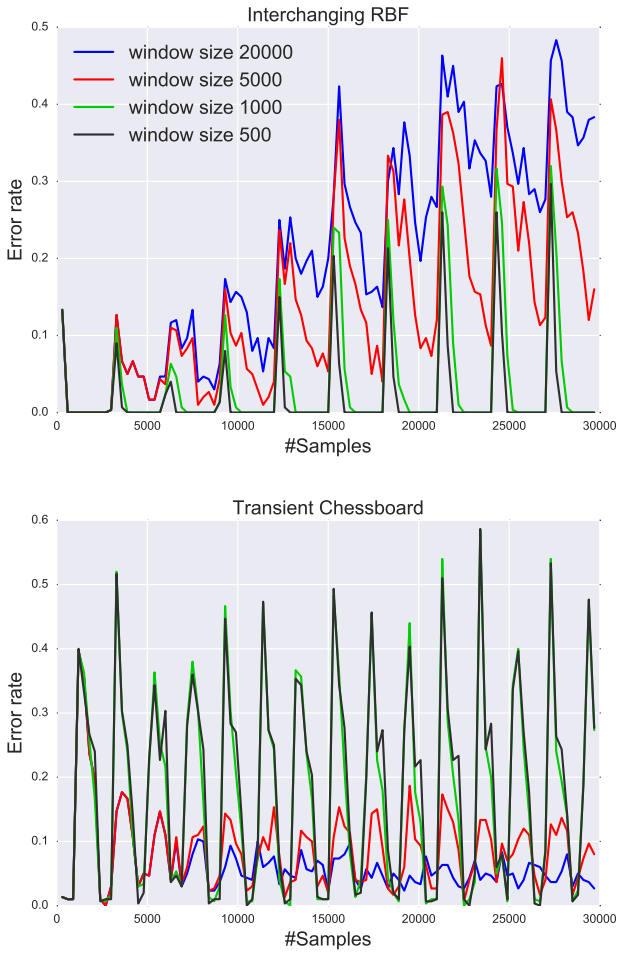

Fig. 6. Error rate course of kNN models with differently sized sliding windows. In the case of real drift (top), the error rate of larger windows recovers significantly slower after each abrupt drift. In contrast, larger windows achieve lower error rates when no real drift is present (bottom) because the additional data facilitates the problem.

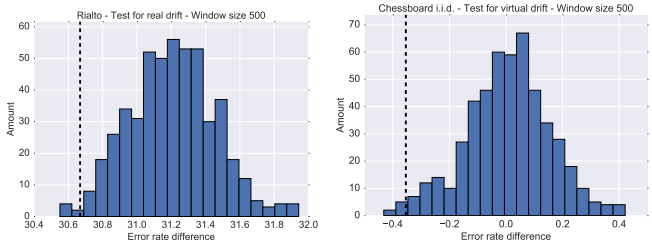

Fig. 7. Exemplary histograms of error rate differences achieved with the test model using a sliding window of 500 samples. The differences were obtained between the achieved error rate on unordered and ordered bootstrap samples. The test is positive whenever the first percentile of the distribution, illustrated by the dashed line, is larger than 0. This is true for the *Rialto* benchmark (left) but not for *Chessboard i.i.d.* (right).

Figure 8 depicts for various datasets the error rate courses of the respective three best methods. The robustness of SAM is emphasized once again, since it is within the three best methods in every experiment. It performs best in the *Rialto* dataset. The daily pattern is reflected in the up and down of the error rates. During midday the task is comparably easy because the colors are clearly pronounced, whereas the rather blurry conditions of the morning and evening are clearly more challenging. SAM performs second best in the Moving Squares dataset, slightly worse than the best method DACC.

Since kNN_{W_A} also uses kNN and actively manages its window, it is most closely related to SAM., However, it performs worse in all experiments. SAM uses a distance weighted kNN, whereas kNN_{W_A} relies on uniform weights (majority voting). We evaluated SAM also with majority voting and got an overall average error rate of 14.72%, emphasizing that its advantage is due to the memory architecture.

For the sake of completeness, we also report the error rates of all window based approaches with a window size of 1000 samples as done in [13]. Especially the results of kNN_S and kNN_{W_A} are significantly better with the smaller window. SAM also profits sometimes, e.g. for the *Moving RBF* dataset, albeit clearly weakened. The reason is that the smaller window

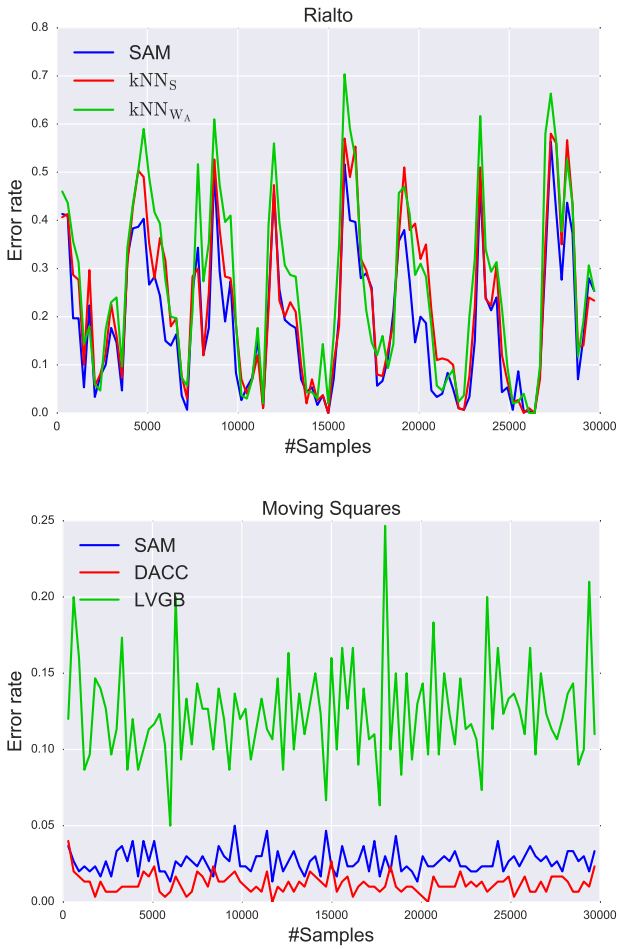

Fig. 8. Error rate courses of the three best methods in the *Rialto* and *Moving Squares* datasets. The daily pattern of the *Rialto* dataset is reflected in the up and down of the error rates. Whereas, the continuous incremental drift in the *Moving Squares* dataset results in a rather constant performance.

conceals the issue of the methods sometimes failing to shrink the window appropriately. Samples of former concepts are fading out faster of the smaller window and are, therefore, less often contradicting the current concept in the case of real drift. We chose the larger and more informative size of 5000 samples for the ranking.

Our results confirm the fact that kNN is in general a very competitive algorithm in the streaming setting. It is quite surprising that the fixed sliding window approach kNN_S performs comparably well or even better than more sophisticated methods such as DACC or L++.NSE. The fixed sliding window approach only struggles with high real drift tasks and achieves competitive results in all other settings. This is especially emphasized by the second best average result for real-world datasets, containing rather low degrees of real drift. Hence, this simple approach with a comparably low computation complexity may be a reasonable alternative to sophisticated drift algorithms in a variety of real-world tasks.

B. Run-time

We did not extensively compare the run-time of all methods because the results significantly depend on the efficiency of the specific implementation. However, we can show that the run-time for SAM is roughly comparable to kNN_WA. On an Intel i5-4440 SAM processed a dataset in $530.8\,\text{s}$ on average, whereas kNN_WA took $404.3\,\text{s}$. This might be surprising, since SAM incorporates more time-consuming processing steps such as the adaption of the STM and the cleaning of the LTM. However, the distance calculations of the STM are necessary anyways for the classification and can be reused in those steps, reducing the run-time significantly. In our implementation the distances are stored in a matrix and updated incrementally.

Apart from the suggestions in section IV-C, further improvements can be achieved by using parallelized Nearest-Neighbor implementations or efficient data structures such as Vantage Point Trees [46].

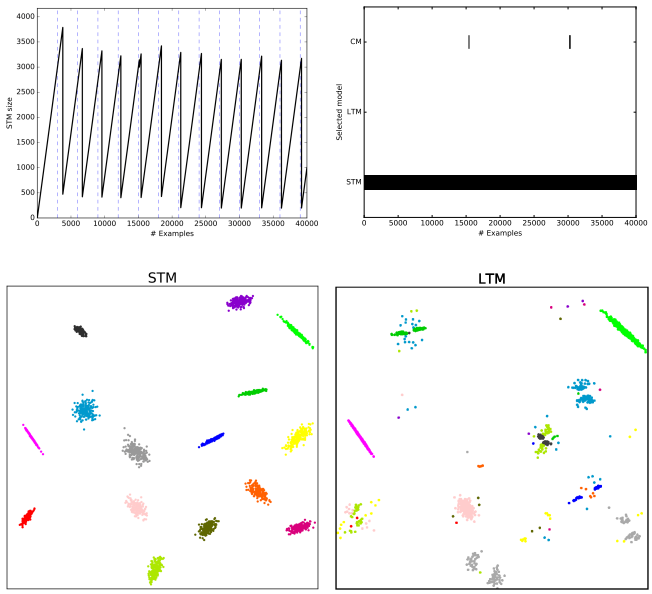

Fig. 9. Illustrations of the Interchanging RBF dataset. Only a part of the dataset is depicted for the sake of clarity. Top: The size adaptation of the STM is shown on the left. Dashed vertical lines mark the moments of abruptly changed concepts. The delay of the model shrinks with increasing strength of drift. Only the STM is used for prediction (right). Bottom: Snapshots of the STM and LTM. Points of the LTM that are in accordance with the STM are preserved. Different classes are represented by different colors.

C. Memory Behaviour

In this section we illustrate the adaptation of the memories as well as their task specific roles.

Figure 9 shows on the top left the size adjustment of the STM during the Interchanging RBF experiment. The algorithm reliably reduces the window size after each abrupt drift. However, we also observe a certain delay, during which wrong predictions are likely to occur. This delay is due to two reasons. Firstly, a certain amount of examples is required to reliably predict the new concept. Secondly, the more examples of the former concept are contained in the STM, the more stable is its representation and the more examples of the new concept are required to deteriorate the overall accuracy sufficiently. Hence, the delay illustrates a self-adjusting trade-off between adaptation speed to new concepts and the robustness against

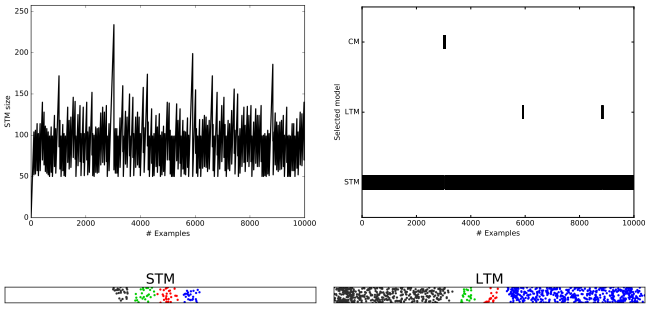

Fig. 10. Depictions of the Moving Squares benchmark. Top: The STM size is most of the time kept below 120 samples and avoids the overlap of current points by outdated ones. The STM is exclusively used for prediction (right). Bottom: Snapshots of the STM and LTM. Classes do not overlap within both memories.

noise, governed by the stability of both concepts. The adaptation delay decreases with increasing drift strength.

The selective cleaning of the LTM is also visible in Figure 9. The empty spots within the clusters are due to the cleaning of contradicting instances. The remaining samples were harmless and consequently are kept in memory.

As already mentioned, the Moving Squares dataset is designed such that the squares may overlap each other if more than 120 of the recent examples are kept in memory. Hence, the best strategy for this problem is to keep the window as small as possible and to use only the most recent examples for prediction.

Figure 10 shows how the size of the STM is mostly kept between 50 and 100 samples, allowing a nearly perfect prediction. This is also clarified by its snapshot, illustrating the absence of overlapping instances. The parameter controlling the minimum size of the STM prevents a size reduction below 50 examples. Otherwise, an even lower error rate result could be achieved.

In contrast to the previous two datasets, which basically do not require the LTM, we see in Figure 11 its importance during the prediction of the Transient Chessboard task. The STM is used alone whenever the data is presented square by square because it contains solely relevant information for the current square and produces less mistakes than in combination with the LTM. But during the periods in which examples are distributed over the whole board, the LTM is heavily used since it contains beneficial information from the past. Its snapshot reveals the compressed preservation of the whole chessboard.

The task dependent relevance of both memories is exemplified with real-world data in Fig.12: While the LTM is the preferred prediction model in the Weather dataset, it is the STM that classifies most of the instances of the *Rialto* task.

Even though our memory architecture performs very robustly, there are some cases leading to unnecessary deletions within the LTM. For instance, a large amount of spread noise erases information from the LTM because of the assumption of new data being the most valuable one. Contrary concepts temporarily interchanging each other cause also deletions.

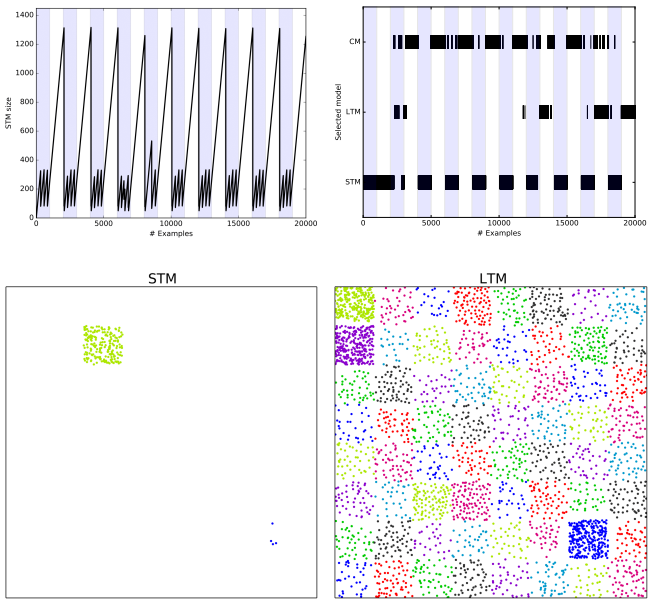

Fig. 11. Illustrations of the Transient Chessbord dataset. Top: Each 1000 examples the square by square revelation (blue background) is alternated by samples covering the whole chessboard(white background). The STM tracks the current concept: It shrinks after each revealed field in the first phase and grows during the second phase to contain the whole chessboard. Only a part of the dataset is depicted for the sake of clarity. Bottom: The LTM preserves the whole chessboard in compressed fashion, whereas only the current concept is contained in the STM.

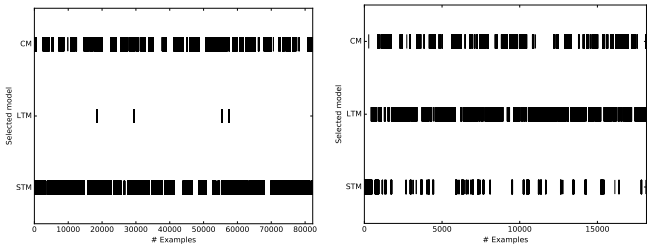

Fig. 12. Model selection for the datasets Rialto (left) and Weather (right).

D. Impact of Clustering on Generalization

The class-wise clustering distinctly restructures the LTM. We empirically analyzed its impact on the generalization error. Precisely, we measured the error rate of the LTM for the subsequent 1000 samples of the stream before and after each clustering. This has been done for all datasets with total memory sizes of 5000 and 1000 samples (for the STM and LTM combined). It turns out that the generalization error is only slightly affected. Precisely, the average error rate of the LTM before and afterwards the clustering is 33.62 % and 34.10 % for the memory size of 5000 samples. The size of 1000 samples results in an error rate of 33.63% and 35.14%.

VIII. Conclusion

In this paper, we presented the Self Adjusting Memory (SAM) architecture, designed to handle heterogeneous concept drift within streaming data. It explicitly separates the current concept from former ones and preserves both in dedicated

memories combining them according to the demands of the current situation. Thereby, it omits a common weakness of available methods that simply discard former knowledge and produce more mistakes in case of reoccurring drift. Our method is easy to use in practice since it requires neither meta-parametrization nor related prior knowledge about the task at hand.

To enrich the evaluation of our method we tackled the common problem of unknown drift characteristics in real-world datasets. Here, we applied two consecutive tests able to detect the drift type, virtual or real, and its degree. The tests correctly classified all artificial datasets and deliver a reasonable categorization of the real-world benchmarks. We were able to confirm the assumption that most real-world datasets contain only small amounts of real drift.

Equipped with this knowledge, we compared SAM in an extensive evaluation with state of the art methods. As the only algorithm, it demonstrated consistently accurate results for heterogeneous drift types: virtual versus real drift, abrupt versus incremental drift. We also showed that the different memory types fulfill different functions in the overall model. While the STM represents the actual concept, the LTM conserves established knowledge as long as it is consistent with the STM. This decomposition proved to be particularly effective to simultaneously deal with heterogeneous drift types in real-world streaming data.

Furthermore, the experiments showed that, apart from tasks with high real drift degree, the simple fixed sliding window approach is a reasonable alternative to more sophisticated drift algorithms. It has often significant advantages in terms of computational complexity.

We want to pursue our promising approach further and consider a spacial decomposition to enable a more fine-grained combination of the memory models. This allows a reaction to different concurring drift types for different locations in space. Furthermore, the generality of the proposed architecture permits a combination with alternative models such as memory and time efficient parametric probabilistic models or incremental/decremental SVMs. We already conducted some experiments with the Naive Bayes algorithm and the first results are very promising.

IX. Acknowledgement

Barbara Hammer acknowledges support by the Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277) at Bielefeld University, which is funded by the German Research Foundation (DFG).

References

[1] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Netw. and Appl.*, vol. 19, no. 2, 2014.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence–SBIA 2004*. Springer, 2004, pp. 286–295.

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

[5] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, Oct 2011.

[6] S. Schiaffino, P. Garcia, and A. Amandi, "eteacher: Providing personalized assistance to e-learning students," *Computers & Education*, vol. 51, no. 4, pp. 1744–1754, 2008.

[7] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *Systems, Man and Cybernetics, IEEE Transactions on*, no. 4, pp. 325–327, 1976.

[8] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 44, 2014.

[9] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *Computational Intelligence Magazine, IEEE*, vol. 10, no. 4, pp. 12–25, 2015.

[10] A. Bifet and R. Gavald, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM International Conference on Data Mining*, 2007, pp. 443–448.

[11] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi, "An information-theoretic approach to detecting changes in multi-dimensional data streams," in *In Proc. Symp. on the Interface of Statistics, Computing Science, and Applications*. Citeseer, 2006.

[12] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases-Volume 30*. VLDB Endowment, 2004, pp. 180–191.

[13] A. Bifet, B. Pfahringer, J. Read, and G. Holmes, "Efficient data stream classification via probabilistic adaptive windows," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. New York, NY, USA: ACM, 2013, pp. 801–806.

[14] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning*, vol. 23, no. 1, pp. 69–101, 1996.

[15] R. Klinkenberg and T. Joachims, "Detecting concept drift with support vector machines," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 487–494.

[16] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2000, pp. 71–80.

[17] G. Jaber, A. Cornuéjols, and P. Tarroux, "Online learning: Searching for the best forgetting strategy under concept drift," in *Neural Information Processing*. Springer, 2013, pp. 400–408.

[18] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, pp. 135–150.

[19] N. C. Oza, "Online bagging and boosting," in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, Oct 2005, pp. 2340–2345 Vol. 3.

[20] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[21] B. Hammer and A. Hasenfuss, "Topographic mapping of large dissimilarity data sets," *Neural Computation*, vol. 22, no. 9, pp. 2229–2284, 2010.

[22] N. Alex, A. Hasenfuss, and B. Hammer, "Patch clustering for massive data sets," *Neurocomputing*, vol. 72, no. 7-9, pp. 1455–1469, 2009.

[23] P. X. Loeffel, C. Marsala, and M. Detyniecki, "Classification with a reject option under concept drift: The droplets algorithm," in *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Oct 2015, pp. 1–9.

[24] P. Zhang, B. J. Gao, X. Zhu, and L. Guo, "Enabling fast lazy learning for data streams," in *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, ser. ICDM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 932–941.

[25] Y.-N. Law and C. Zaniolo, "An adaptive nearest neighbor classification algorithm for data streams," in *Knowledge Discovery in Databases: PKDD 2005*. Springer, 2005, pp. 108–120.

[26] E. S. Xioufis, M. Spiliopoulou, G. Tsoumakas, and I. Vlahavas, "Dealing with concept drift and class imbalance in multi-label stream classification," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, ser. IJCAI'11. AAAI Press, 2011, pp. 1583–1588.

[27] R. Atkinson and R. Shiffrin, "Human memory: A proposed system and its control processes," *Psychology of Learning and Motivation*, vol. 2, pp. 89 – 195, 1968.

[28] Y. Dudai, "The neurobiology of consolidations, or, how stable is the engram?" *Annual Review of Psychology*, vol. 55, pp. 51–86, 2004.

[29] G. A. Miller, "The magical number seven, plus or minus two: some limits on our capacity for processing information." *Psychological Review*, vol. 63, no. 2, p. 81, 1956.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, and M. Perrot, "Scikit-learn: Machine learning in Python," *JMLR*, 2011.

[31] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[32] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *The Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[33] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01. New York, NY, USA: ACM, 2001, pp. 377–382.

[34] M. Harries and N. S. Wales, "Splice-2 comparative evaluation: Electricity pricing," 1999.

[35] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *Fourth International Workshop on Knowledge Discovery from Data Streams*, vol. 6, 2006, pp. 77–86.

[36] L. I. Kuncheva and C. O. Plumpton, "Adaptive learning rate for online linear discriminant classifiers," in *Structural, Syntactic, and Statistical Pattern Recognition*. Springer, 2008, pp. 510–519.

[37] I. Zliobaite, "How good is the electricity benchmark for evaluating concept drift adaptation," *CoRR*, vol. abs/1301.3524, 2013.

[38] J. Gama, R. Rocha, and P. Medas, "Accurate decision trees for mining high-speed data streams," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2003, pp. 523–528.

[39] N. C. Oza and S. Russell, "Experimental comparisons of online and batch versions of bagging and boosting," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2001, pp. 359–364.

[40] V. Losing, B. Hammer, and H. Wersing, "Interactive online learning for obstacle classification on a mobile robot," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–8.

[41] R. R. Wilcox, *Introduction to robust estimation and hypothesis testing*. Academic Press, 2012.

[42] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[43] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[44] C. E. Bonferroni, *Teoria statistica delle classi e calcolo delle probabilita*. Libreria internazionale Seeber, 1936.

[45] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[46] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '93. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1993, pp. 311–321.