How Well do Clustering Similarities-Based Concept Drift Detectors Identify Drift in case of Synthetic/Real-World Data?

Jindrich Pohl[1]

Supervisors: Jan Rellermeyer[1]**, Lorena Poenaru-Olaru**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 29, 2023

Name of the student: Jindrich Pohl, email: j.pohl@student.tudelft.nl
Final project course: CSE3000 Research Project
Thesis committee: Jan Rellermeyer, Lorena Poenaru-Olaru, Jesse Krijthe

Abstract

Concept drift is an unforeseeable change in the underlying data distribution of streaming data, and because of such a change, deployed classifiers over that data show a drop in accuracy. Concept drift detectors are algorithms capable of detecting such a drift, and unsupervised ones detect drift without needing the data's actual labels, which can be expensive to obtain. This work is concerned with the implementation and evaluation of two existing unsupervised concept drift detectors based on clustering, UCDD and MSSW, by evaluation on both synthetic and real-world data. Our biggest contribution is in making implementations publicly available. By evaluation, we also realise that UCDD detects drift earlier for simple numerical synthetic datasets, MSSW detects drift earlier for more complex synthetic datasets with categorical features, and none seems suitable for real-world datasets.

1 Introduction

Streaming data analysis is currently in high demand, and its rapid growth is expected in the upcoming years. The streaming data analytics market has been valued at $ 7 Billion in 2019, and is expected to grow to up to $ 50 Billion by 2027 [1]. However, the analysis of data streams is faced with multiple challenges stemming from the online learning environment. One such challenge is concept drift, which signifies a drastic, unpredictable change in streaming data over time, characterised by a change in the underlying distribution of said data (Note: in the rest of this paper, the terms "concept drift" and "drift" will be used interchangeably). Most streaming data analysis tasks can suffer from concept drift; illustrative examples include monitoring tasks [2], user modelling tasks [3], intrusion detection [4], or fraud detection [5].

A machine learning (ML) classifier - i.e. base classifier - is typically trained on stationary reference data that is assumed not to contain concept drift, and then released into production to process incoming streaming data, called testing data in our work. Once this incoming data starts exhibiting concept drift, the base classifier's accuracy drops significantly, precisely because such data is different from what the classifier has been trained on. This is why having mechanisms dealing with concept drift is of high importance.

One way of dealing with concept drift is detecting it with concept drift detectors (DDs). A DD is a monitoring algorithm for a deployed ML model in production that can detect drift, so that the ML model can be quickly adapted to new, shifted (drifted) data to avoid accuracy loss. Most publicly available DDs are supervised: they assume that true testing data labels are available shortly after prediction, and use the difference between predicted testing labels and true testing labels to test for drift. Nonetheless, in reality, obtaining true streaming data labels is expensive, so unsupervised DDs might be preferred in industry. Unsupervised DDs work by observing differences between reference and testing data itself, so they do not need access to testing data labels. However, in papers describing and evaluating unsupervised DDs, implementations are only rarely publicly available. Moreover, most of these DDs are evaluated only on synthetic datasets, which might not reflect the detectors' usefulness in the real world.

Our contributions are therefore the following. Firstly, we replicate implementations of two existing clustering similarities-based unsupervised drift detectors in Python 3.9.0 with NumPy[1] and Pandas[2] and make these implementations public[3], so that they can be used by researchers and practitioners. Most notably, we extend the work of Lorena et al. [6] - a comparative study of drift detectors, severely limited by the unavailability of unsupervised drift detectors. Secondly, by this replication, we validate results of two existing research papers. Finally, we evaluate the implemented detectors on real-world data, which has not been done before.

We aim to answer the following research question: How well do clustering similarities-based concept drift detectors identify drift in case of synthetic/real-world data? This question can be further divided into these sub-questions: What is the false-positive rate (FPR) and latency of the algorithms under different settings such as abrupt versus gradual drift, or the presence of categorical features in synthetic data? And how well, in terms of FPR and detection accuracy, do these algorithms detect drift in real-world datasets?

The paper is organised as follows. Section 2 presents the existing categorization of unsupervised drift detectors, and introduces UCDD and MSSW, the two implemented algorithms. Section 3 describes the methodology: datasets used, their preparation through division to reference data and testing batches, evaluation metrics such as latency and FPR, and implementation decisions such as libraries to use. In Section 4, the promising results on synthetic datasets and suboptimal results in real-world datasets can be found, followed by a discussion of why these results were obtained in Section 5. Section 6 touches upon ethical aspects such as this work's reproducibility, and conclusions summarize our contributions in Section 7.

2 Related work

Four different categories of unsupervised drift detectors were extracted from [7] and [8]. These are statistical-based drift detectors - drift detection based on statistical tests, margin density-based drift detectors - drift detection based on the density of points on a classification boundary, clustering similarities-based drift detectors - drift detection relying on clustering, and mixed drift detectors - multiple different strategies combined.

This paper is concerned with the category of clustering similarities-based drift detectors. These detectors work by using clustering to assess whether clustering characteristics in newly incoming data differ from clustering characteristics in old data. Moreover, label estimates can be useful to detect concept drift, e.g. to observe that samples of one class suddenly appear elsewhere in the feature space. With shifted

[1] https://numpy.org/

[2] https://pandas.pydata.org/

[3] https://github.com/Jindrich455/clustering-drift-detection

data, a base classifier would estimate labels in a biased way, while label estimates from unsupervised clustering are potentially of more use by being unbiased. Two existing algorithms were found, implemented and evaluated in this paper, and in the following paragraphs, each is introduced briefly. Note that a drift detector can signal that drift occured, but only if the drift really happened can we say that the detector detected drift.

The first algorithm, **UCDD** (Unsupervised Concept Drift Detection) [9], works as follows. Firstly, data from a fixed-size reference and testing window - a window signifies multiple time-consecutive streaming data samples - is merged. K-means is used on this dataset to obtain two clusters, called plus and minus, which should serve as data label estimates, assuming there are two classes in the data. Then, each cluster is split back to reference and testing data; this (ideally) results in four clusters in total. Subsequently, as illustrated by Figure 1, the reference cluster of the plus class is compared to reference and testing clusters of the minus class through the number of nearest neighbours in each. If there is a significant difference in these two numbers, assessed through a critical value of the CDF of the Beta distribution, then concept drift is signaled. The same happens for the minus class reference cluster and the two plus clusters.

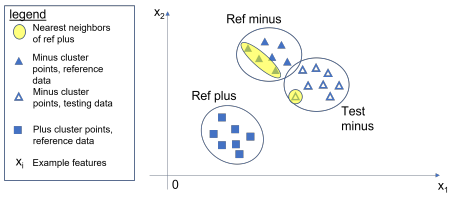

Figure 1: Comparison of nearest neighbours in UCDD in the plus reference cluster against reference and testing minus clusters.

The second algorithm will be referred to here as **MSSW** (Multi-Scale Slide Windows) [10], and works thus. First, all reference data is clustered through k-means with weighted Euclidean distance. Then, a lower and upper bound of acceptable total average distances to centroids is calculated from this reference data. Each testing batch (i.e. fixed-size window) is subsequently clustered according to the obtained reference data centroids, and when the total average distance to centroids in the testing batch exceeds the defined bounds, drift is signaled.

3 Methodology

In this section, the entire experimental setup is outlined. The data setup is described in 3.1, followed by the description of datasets in 3.2. Evaluation metrics used to assess the algorithm's performance are introduced in 3.3. Once the evaluation setup is clear, the necessary implementation adaptations done to the respective algorithms are explained in 3.4.

3.1 Data setup and dataset types

In this work, the setup in Figure 2 was used for the algorithm evaluation. Each dataset used in the evaluation was first split to reference and testing data. Testing and possibly also reference data was then split to equal-sized batches, and some testing batches, shown in gray, were known or defined to contain concept drift. The drift detectors implemented in this paper accept reference data and then individual testing batches as input, and should be able to tell which testing batches contain drift. Two types of datasets were used for the evaluation - synthetic and real-world, and the next paragraphs explain the data setup for each.

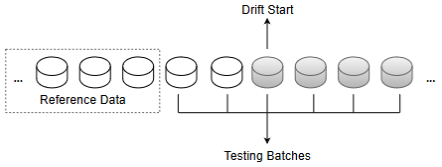

Figure 2: High-level data setup for evaluation. Data is split to reference and testing and then divided to batches. Grey batches contain drifting data. Note: the number of batches differs per dataset. Adapted from Lorena et al. [6].

Synthetic datasets were generated artificially, so drift was chosen to start at one testing batch. All remaining testing batches also exhibit concept drift, like in Figure 2. Figure 3 shows the two types of concept drift: abrupt drift means a sudden complete change of concept, while gradual drift has a width, and means a progressive introduction of a new concept. Drift start location, type, and width were chosen in advance, so we knew exactly where a drift detector should signal drift, and interesting results could be extracted about different drift types and widths.

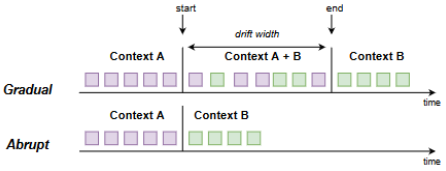

Figure 3: Concept drift can be abrupt or gradual. The gradual drift's width is the number of samples from the start of the new concept until the new concept is fully established. Adapted from Lorena et al. [6].

The detection performance of a drift detector on real-world data is expected to reflect the usefulness of the said detector on real-world problems much better than the detection performance on synthetic data. Nevertheless, in real-world datasets, the exact concept drift locations were unknown, and therefore had to be defined. In this case, testing batches were defined as drifting on a case-by-case basis, so we could no longer work with concept drift "starting" at some testing batch.

3.2 Description of datasets

All datasets used in this paper have binary classification labels. Our work is therefore mostly useful for binary streaming data classification. The exact synthetic and real-world datasets used are described in the following paragraphs.

Synthetic datasets used in this paper were the same as those employed by Lorena et al. [6]. All have 100 000 total samples, the first 30 000 samples were taken as reference data, the remaining 70 000 as testing, and experiments were done with batches of 10 000 samples. Drift always starts at the third testing batch. *SEA* is a dataset of three numerical features ranging from 0 to 10. The datasets *AGRAW1* and *AGRAW2* have the same features, but differently generated values. These are six numerical features and three categorical features describing a person, and labels are a decision of whether this person can receive a loan.

Table 1: Numerical description of real-world datasets.

Dataset	Size	# Ref	# Num. features	# Cat. features	# Experiments
Weather	18159	6053	8	0	2
Spam	4405	1468	10727	0	3
ELECT2	45312	15104	8	0	1
Airlines	539383	179794	3	1	2

Four real-world datasets were used for the evaluation: *Weather*, *Spam*, *ELECT2*, and *Airlines*. *ELECT2* [11] and *Airlines* [12] were also employed by Lorena et al. [6], while *Weather* [13] and *Spam* [14] were used additionally. In Table 1, we can see the total number of samples (Size), the first n samples used as reference data (# Ref), where $(Size - \#Ref)$ samples were then used as testing data, the numbers of numerical and categorical features, and the number of experiments done in each dataset. In *Weather*, the idea is to predict whether it is raining or not, and we did two experiments with batches of either 365 (yearly) or 30 (monthly) samples. *Spam* is a sparse representation of word frequencies in emails, for which it should be decided whether they are spam or not, and we did three experiments with batches of 100, 50, and 20 samples. *ELECT2* is concerned about rises or drops in electricity prices, and we did one experiment with batches of 365 (yearly) samples. The *Airlines* show flight details from which the aim is to predict whether a flight will be delayed or not. We did two experiments, both with batches of size 17 000, but two different encoding methods for the categorical feature: one-hot encoding and target encoding.

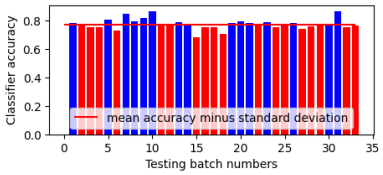

Figure 4: Concept drift definition example in the Weather dataset for yearly experiments. Batches defined as drifting shown in red.

For the Weather dataset, sequential cross-validation [15] was used to obtain the expected accuracy. Each time, a number of consecutive samples in reference data of the same size as one testing batch were taken randomly. A classifier was trained on the rest of reference data, and then its accuracy was measured on the extracted points. This was repeated until the standard error of accuracies was smaller than 0.05. The mean and standard deviation (std) of all these accuracies were then taken as benchmark. Gaussian Naive Bayes, decision trees, k nearest neighbors, SVM, and logistic regression from sklearn were tried and their hyperparameters tuned through Bayesian search[4] in both monthly and yearly experiments. In both cases, SVM[5] was the best-performing method, with C=2.09 in monthly experiments and C=11.98 in yearly experiments. We then defined testing batches to be drifting if their accuracy was smaller than the benchmark accuracy minus the benchmark std - i.e. batches below the red line in Figure 4. Drift definitions for all real-world datasets used were done in a similar way[6].

3.3 Evaluation metrics

In synthetic datasets, the drift detectors' performance was based on two metrics: false-positive rate (FPR), and latency, both adapted from Lorena et al. [6]. These reflect how often a detector signaled drift before it occurred, and how late it detected drift after it occurred. With a good drift detector, both should have values as low as possible. Each metric is described in more detail in the following paragraphs.

Synthetic False-Positive Rate (FPR_S) is the percentage of testing batches before drift happened where a detector signaled drift. It takes on values between 0 and 1, where 0 means that no drift was signaled before it was supposed to be signaled, and 1 means that all batches before the first batch with drift were wrongly signaled as containing drift. The formula, adapted from [6], is

$$FPR_S = \frac{k^F}{|B_{ND}|} : b_k^F \in B, \tag{1}$$

where B is the list of batches, b_k^F is the batch erroneously signaled as drift, and B_{ND} is the total number of batches without drift out of the total list of batches.

Latency (LTC) is the percentage of testing batches required to detect concept drift after it occurred. It is therefore only applicable to the first signaled batch at or after the first batch with actual concept drift. It takes on values between 0 and 1, where 0 means that drift was detected at the exact location where it was supposed to be detected, and 1 means that drift was detected only at the last batch or not detected at all. The formula, adapted from [6], is

$$LTC = \frac{k - j}{|B_{AD}|} : b_j, b_k \in B, \tag{2}$$

where B is the list of batches, b_n is the n^{th} batch in B, b_j is the batch corresponding to the beginning of concept drift, b_k is the batch detected as drift, and B_{AD} is the number of batches after the first batch with drift.

In real-world datasets, the drift detectors' performance was based on **real-world FPR** (FPR_R), and **detection accuracy** (ACC). The metrics are:

$$FPR_R = \frac{|B_{DS} \cap B_{DD}^C|}{|B_{DD}^C|},$$

$$ACC = \frac{|B_{DS} \cap B_{DD}|}{|B_{DD}|},$$

[4] skopt.gp_minimize()

[5] sklearn.svm.SVC()

[6] https://github.com/konstaka/drift-definitions

where B_{DS} are testing batches where drit was signaled by a detector, B_{DD} are batches where drift was defined, and regular set complement, intersection and size notation is used.

3.4 Implementation of concept drift detectors

The clustering used in the two drift detectors shares three common characteristics. Firstly, the k-means algorithm is used, because this clustering algorithm was also used in the papers from which the algorithms are adapted, is well-known, is relatively fast, and its implementation is publicly available in the widely used scikit-learn Python library[7]. Secondly, in this imported KMeans, the kmeans++ method is used to obtain the initial centroids. This method chooses initial cluster centroids progressively, through weighted probabilities based on distances from already obtained centroids. This way, the convergence of KMeans was faster and results were more consistent with different random states than with purely randomly selected initial centroids. Finally, data was always scaled through min-max scaling[8]. This scaling does not assume any particular data distribution, and ensures that features with larger value ranges do not unreasonably contribute more to the distance between data points in k-means. Note that this scaling is built into MSSW by default.

The two papers describing the algorithms also do not mention anything about categorical features, which are present in some of our datasets, cannot be directly handled by k-means, and therefore had to be adapted to the algorithms. We tried both omitting these features and converting them to numerical ones through one-hot encoding[9] and target encoding[10]. These two encodings were suitable, because the notion of distance between data points, necessary in k-means, remained consistent, as opposed to e.g. ordinal encoding, where the distance between two different encoded categorical values depends on these values without conveying any meaning. In the following headers, the algorithm-specific implementation adaptations to the given setup are described for each algorithm. Parameters not described here were kept exactly the same as in the original papers.

UCDD

For some UCDD-specific functionalities, we imported additional libraries. Firstly, it must be possible to find the k nearest neighbors. This is why we used `sklearn.neighbors.NearestNeighbors` for the analysis. We are always using 1-NN, since that is what the original UCDD paper is also using. UCDD also uses the CDF of the Beta function, for which we chose `scipy.stats.beta.cdf`.

Another adaptation concerned the split of reference data to batches. When UCDD was run with all reference data - 30k samples in synthetic data - and one testing batch - 10k samples in synthetic data - then drift was always signaled in each such testing batch. This is because the algorithm works by comparing the nearest neighbors of one reference cluster to another reference and a testing cluster. If there are much more points in the reference cluster than in the testing cluster, then much more neighbours will be found in the reference cluster than in the testing cluster, even if they both come from the same distribution. For this reason, reference data was always split to batches of the same size as a testing batch, and each testing batch was compared to all reference batches individually, as can be seen in Figure 5.

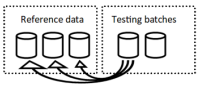

Figure 5: Comparing one testing batch to all reference batches in UCDD.

For the evaluation, we created a parameter named `min_ref_batches_drift` (`MRBD`) to determine the minimum fraction of reference batches that must signal drift for UCDD to signal the current testing batch as drifting. Because there were always three synthetic reference data batches, we tried values of 0.3, 0.6 and 0.9 (i.e. requiring either 1/3, 2/3, or all batches) for synthetic datasets.

One important drawback of the original algorithm is that it was designed to only detect drift when classes shift further away from each other, but not the one where they shift towards each other. This is due to a one-ended comparison of the numbers of nearest neighbours, obtained by $CDF_{Beta(n_1,n_2)}(0.5) < 0.05$. In this formula, either $n_1 =$ number of nearest neighbours in ref plus and $n_2 =$ number of nearest neighbours in test plus, or n_1= number of nearest neighbours in ref minus and $n_2 =$ number of nearest neighbours in test minus. However, by making this comparison double-ended, so by additionally also checking whether $CDF_{Beta(n_2,n_1)}(0.5) < 0.05$ holds, drift could also be detected in the case where classes shifted towards each other, which is certainly not less of a drift than when classes shift away from each other. The (promising) performance of this additional check to the algorithm can be observed in Section 4.

The choice of clustering parameters for KMeans - `n_clusters`, `tol`, and `max_iter` - was complicated, because clustering is done between a reference and a testing batch, but in the real world, when setting these parameters, only reference data is known. The number of clusters, `n_clusters`, was set to two, because the UCDD paper argued that our aim is to estimate classes through clustering, and all our datasets had two classes. `Tol` controls whether the change in centroids is small enough to stop KMeans, while `max_iter` is the maximum acceptable number of iterations until the algorithm stops. We cannot know how demanding will be the clustering of a new dataset made of one reference and one testing batch. However, since half of such a dataset is made of a reference batch, we could try finding suitable parameters by clustering datasets made of reference batches. This was done by picking each possible consecutive pair of reference batches, joining them, and performing clustering 100 times. In all datasets, we concluded that strict convergence (`tol`=0) could be obtained in each of the runs in less than a second, so `tol`=0 was used for the final analysis. Out of

[7] sklearn.cluster.KMeans()

[8] sklearn.preprocessing.MinMaxScaler()

[9] sklearn.preprocessing.OneHotEncoder()

[10] category_encoders.TargetEncoder()

all the combinations and clusterings, the highest number of iterations observed was picked, and the `max_iter` used for the final analysis was 1000 times this value. By using such a `max_iter` on testing data, we could be fairly confident that KMeans reached strict convergence every time.

In UCDD, it is possible that less than four clusters are created. This is because joined reference and testing data forms two clusters in which reference and testing data should be separated afterwards. But if either of these two clusters is made of only reference or only testing data, then no split is happening. Since the number of nearest neighbours of an empty set is undefined, we decided to signal drift whenever less than four clusters were created, because that probably meant a significant change in the data.

MSSW

A first concern during the implementation of MSSW was the notion of "sub-windows". The original paper used a reference and a testing window, which are further divided into equal-sized sub-windows, and mentioned that drift should be detected between the two windows, but then found drift for each testing sub-window. We therefore assumed that drift detection happens for sub-windows, not for whole windows. Adapting this to our work, both the reference and the testing data were divided to same-sized batches. Then, all the reference data was considered as the reference window, and all testing data as the testing window. Batches of reference and testing data are taken as sub-windows, and the drift detector signals testing batches - i.e. sub-windows - which should contain concept drift.

The most important challenge that had to be addressed in MSSW was the custom k-means with an entropy-weighted Euclidean distance used in the original paper. The main issue was that existing implementations of k-means either only support regular Euclidean distances (`scikit-learn`), or support the injection of a custom distance function (`pyclustering`), but then become orders of magnitude slower. However, we made the following observation about data weighting. If not stated otherwise, the same nomenclature as in the MSSW paper is used.

Distance between two points - put:

$$z'_{tg} = \sqrt{W_g} x'_{tg}, \tag{3}$$

$$z'_t = [z'_{t1}, z'_{t2}, ..., z'_{tdm}], \tag{4}$$

where dm is the number of dimensions.

Then

$$EuclideanDist(z'_i, z'_j) = \sqrt{\sum_{g=1}^{dm} (z'_{ig} - z'_{jg})^2} \tag{5}$$

$$= \sqrt{\sum_{g=1}^{dm} (\sqrt{W_g} x'_{ig} - \sqrt{W_g} x'_{jg})^2} \tag{6}$$

$$= \sqrt{\sum_{g=1}^{dm} (W_g (x'_{ig} - x'_{jg})^2)} \tag{7}$$

$$= WeighedDist(x'_{ig}, x'_{jg}) \tag{8}$$

If we choose an initial set of centroids C_{init} in the original data, and run k-means with weighted Euclidean distance, we obtain some final set of centroids C_{final}. However, if we first transform all data including each centroid C_{init} with $\sqrt{W_g}$ as in equation 3, and run k-means with regular Euclidean distance, we obtain the same final set of centroids C_{final}, only weighted by $\sqrt{W_g}$. This means that, except for the different basis, the two clustering methods are equal.

The implementation outcomes of this analysis were the following. We could transform data by $\sqrt{W_g}$, and then use KMeans[11] to obtain the correct final clusters, only in a different basis. MSSW uses average distances to centroids in testing batches to signal drift, and the distance used is still the same weighted Euclidean distance. The data could consequently stay transformed after the clustering; we only needed to also transform testing data with $\sqrt{W_g}$ obtained from reference data, and use a regular Euclidean distance wherever the weighted Euclidean distance was requested.

We had to choose the `n_clusters`, `tol`, and `max_iter` parameters in KMeans for the analysis. In each dataset, we observed the clustering process for 100 different runs of the algorithm (i.e. 100 runs with different random states, so different initial centroids) through `n_clusters`=10, `n_init`=100, `tol`=0, and `max_iter`=500. For all datasets, we concluded that strict convergence (tol=0) is observed after less than 100 iterations, so we took `tol`=0 and `max_iter`=10 000 for a sufficient margin for the final evaluation of MSSW. Then, the elbow method[12] was performed for `n_clusters` ranging from two to 30, where 30 was a high enough upper boundary since the inertia was always close to zero, and the elbow point found for `n_clusters` was taken for the final analysis.

4 Results

In this section, we present resulting metric values in comprehensive tables and discuss how we obtained them and why some values are not reported in full. Because the drift detection setup and metrics differ for synthetic and real-world datasets, results of each dataset type are presented in separate sub-sections.

4.1 Synthetic data

Obtaining meaningful results with randomness stemming from random centroid initialisation was done in the following way. K-means' `n_init` was set to 100, which already guarantees a certain robustness against randomness. Furthermore, each drift detector was run independently 2 times (i.e. k-means was run with at least 200 different random states in total). If the standard error of our evaluation metrics (see Section 3.3) from these initial runs was higher than 0.05, the algorithm was re-run, again with `n_init`=100, until the standard error became smaller. The metrics reported here are the average of metrics of all drift detector runs.

In the resulting tables presented here, the "width" column specifies the drift width in 1k samples - e.g. the value 0.5 means 500 samples, 5.0 means 5000 samples and so on -

[11] sklearn.cluster.KMeans()

[12] yellowbrick.cluster.elbow.kelbow_visualizer

and width 0.0 signifies an abrupt drift. For AGRAW1 and AGRAW2, results were obtained for three encoding strategies: exclude for leaving categorical features out, and one-hot and target for encoding categorical features.

UCDD

Table 2: UCDD metrics in all synthetic datasets with different encodings and increasing drift widths. We tried three different combinations of additional parameters `MRBD` and `ADD_C`, shown as three different columns (a), (b), and (c). Metric values using the best values for `MRBD`, `ADD_C` and the best encoding(s) are highlighted.

			(a)		(b)		(c)	
		MRBD= ADD_C was	0.3 used		0.3 unused		0.6 used	
		width	FPR_S	LTC	FPR_S	LTC	FPR_S	LTC
SEA	EXCLUDE	0.0	**0.5**	**0.0**	0.5	1.0	**0.0**	**0.17**
		0.5	**0.5**	**0.0**	0.5	1.0	**0.0**	**0.25**
		1.0	**0.5**	**0.0**	0.5	1.0	**0.0**	**0.25**
		5.0	**0.5**	**0.0**	0.5	1.0	**0.0**	**0.0**
		10.0	**0.0**	**0.25**	0.0	1.0	**0.0**	**0.25**
		20.0	**0.0**	**0.25**	0.0	1.0	**0.0**	**0.75**
AGRAW1	EXCLUDE	0.0	0.0	0.5	0.0	1.0	0.0	1.0
		0.5	0.0	0.5	0.0	1.0	0.0	1.0
		1.0	0.0	0.5	0.0	1.0	0.0	1.0
		5.0	0.0	1.0	0.0	1.0	0.0	1.0
		10.0	0.0	1.0	0.0	1.0	0.0	1.0
		20.0	0.0	0.75	0.0	1.0	0.0	1.0
	ONE-HOT	0.0	**0.0**	**0.0**	0.0	0.0	0.0	1.0
		0.5	**0.0**	**0.0**	0.0	0.0	0.0	1.0
		1.0	**0.0**	**0.0**	0.0	0.0	0.0	1.0
		5.0	**0.0**	**0.0**	0.0	0.0	0.0	1.0
		10.0	**0.0**	**0.25**	0.0	0.5	0.0	1.0
		20.0	**0.0**	**0.25**	0.0	0.25	0.0	1.0
	TARGET	0.0	**0.0**	**0.0**	0.0	0.5	0.0	1.0
		0.5	**0.0**	**0.0**	0.0	0.5	0.0	1.0
		1.0	**0.0**	**0.0**	0.0	0.5	0.0	1.0
		5.0	**0.0**	**0.0**	0.0	0.5	0.0	1.0
		10.0	**0.0**	**0.0**	0.0	1.0	0.0	1.0
		20.0	**0.5**	**0.0**	0.5	0.75	0.0	1.0
AGRAW2	ONE-HOT	0.0	**0.0**	**0.5**	0.0	0.5	0.0	1.0
		0.5	**0.0**	**0.5**	0.0	0.5	0.0	1.0
		1.0	**0.0**	**0.5**	0.0	0.5	0.0	1.0
		5.0	**0.0**	**0.5**	0.0	0.5	0.0	1.0
		10.0	**0.0**	**1.0**	0.0	1.0	0.0	1.0
		20.0	**0.0**	**0.0**	0.0	0.0	0.0	1.0

In UCDD, multiple different results had to be obtained because of the `MRBD` and `additional_check` (`ADD_C`) parameters. It is important to note that FPR_S is always non-increasing and LTC is always non-decreasing for increasing `MRBD`: requiring more reference batches to signal drift must make drift signaling less likely. On the other hand, when using the additional check, LTC is always at most as high and FPR_S is at least as high as when not using this check, because the check only introduces an additional possibility of detecting drift.

Based on the analysis above, we constructed Table 2, omitting all parameter values that would always give an LTC of 1.0. This is why there are no results for exclude and target encodings in AGRAW2. For completeness, SEA `MRBD`=0.9 when using the check gave LTC higher than 0.7 for all drift widths, so this result was also excluded.

For more insights, we inspected clusters created by k-means to test for the assumption of UCDD that (k-means) clustering can accurately estimate data labels. In SEA, average label estimate accuracies of all drift widths were ranging from 79% to 81%, but in the more complex AGRAW1 and AGRAW2 datasets and all category encoding strategies, this number was at most 56%. The results for AGRAW1, AGRAW2, and real-world datasets should therefore be looked at critically, because there, drift was not detected through the way in which UCDD was designed.

MSSW

Table 3: MSSW metrics in all synthetic datasets with increasing drift widths. We tried three different encodings of categorical variables, shown as three different columns (a), (b), and (c). Values for the best-performing encoding in each dataset are highlighted.

		(a)		(b)		(c)	
	ENCODING:	EXCLUDE		ONE-HOT		TARGET	
	width	FPR_S	LTC	FPR_S	LTC	FPR_S	LTC
SEA	0.0	**0.0**	**0.25**	-	-	-	-
	0.5	**0.0**	**0.25**	-	-	-	-
	1.0	**0.0**	**0.25**	-	-	-	-
	5.0	**0.0**	**0.25**	-	-	-	-
	10.0	**0.0**	**0.25**	-	-	-	-
	20.0	**0.0**	**0.25**	-	-	-	-
AGRAW1	0.0	0.0	0.25	0.0	1.0	**0.0**	**0.0**
	0.5	0.0	0.25	0.0	1.0	**0.0**	**0.17**
	1.0	0.0	0.25	0.0	1.0	**0.0**	**0.17**
	5.0	0.0	0.25	0.0	1.0	**0.0**	**0.0**
	10.0	0.0	0.25	0.0	1.0	**0.0**	**0.25**
	20.0	0.0	0.5	0.0	1.0	**0.0**	**0.25**
AGRAW2	0.0	0.0	0.25	0.0	1.0	**0.0**	**0.2**
	0.5	0.0	0.25	0.0	1.0	**0.0**	**0.2**
	1.0	0.0	0.25	0.0	1.0	**0.0**	**0.2**
	5.0	0.0	0.25	0.0	1.0	**0.0**	**0.2**
	10.0	0.0	0.25	0.0	1.0	**0.0**	**0.25**
	20.0	0.0	0.75	0.0	1.0	**0.0**	**0.5**

Results from MSSW can be found in Table 3, and to have more understanding of these results, we also inspected plots of average centroid distances in testing batches versus drift detection boundaries made by clustering reference data. From there, for example, we observed that the method worked as expected for SEA: the missed first testing batch, characterised by the LTC of 0.25 in all drift widths, was only missed because the average centroid distance of this batch was always just below the boundary for a drift to be signaled. This same close distance to the decision boundary was observed for LTCs of up to 0.25 in other datasets for exclude (a) and target (b) encodings. We also observed that higher drift widths caused average centroid distances for the first drifting batches to not be that different from non-drifting batches, causing MSSW to detect drift later. This is why in AGRAW1 and AGRAW2, we observe increased latencies for higher drift widths - for example, in AGRAW2 target (c), width=5.0 shows an (average) LTC of 0.2, but with width=10.0, the latency increases to 0.25.

4.2 Real-world data

In real-world datasets, k-means' randomness was overcome as follows. For experiments with batches of at most 50 samples, we only run the algorithms once with k-means' `n_init` equal to twice the batch size to make sure all possible initial centroid choices were tried - twice because UCDD always clusters two batches. For experiments with larger batches, we always observed exactly the same results for two independent runs with `n_init`=100, so we always only report results of a single run.

For the *Spam* dataset, for all three experiments, all batches were signaled as drifting by UCDD and no batch was signaled as drifting by MSSW. The reason for these unsatisfactory results lies in the data sparsity: the *Spam* dataset is made of 10 000 columns, but on average, in each row, only 56 values

Table 4: MSSW real-world metrics for different datasets and experiments. FPR=FPR_R, acc=ACC.

	WEATHER		ELECT2	AIRLINES	
	YEARLY	MONTHLY	YEARLY	ONE-HOT	TARGET
acc	0.11	0.043	1.0	0.0	0.05
FPR	0.27	0.06	1.0	0.0	1.0

Table 5: UCDD real-world metrics for different datasets and experiments and an increasing MRBD parameter. FPR=FPR_R, acc=ACC

	WEATHER				ELECT2		AIRLINES			
	YEARLY		MONTHLY		YEARLY		ONE-HOT		TARGET	
MRBD	FPR	acc	FPR	acc	FPR	acc	FPR	acc	FPR	acc
1e-16	0.73	0.83	1.0	1.0	1.0	1.0	1.0	0.75	1.0	0.9
0.1	0.53	0.33	1.0	1.0	1.0	1.0	0.0	0.15	1.0	0.65
0.2	0.27	0.17	1.0	1.0	1.0	1.0	0.0	0.15	0.0	0.6
0.3	0.27	0.06	0.99	1.0	0.93	0.98	0.0	0.3	0.0	0.6
0.4	0.27	0.06	0.93	0.94	0.89	0.98	0.0	0.3	0.0	0.5
0.5	0.27	0.06	0.59	0.76	0.89	0.96	0.0	0.4	0.0	0.05
0.6	0.27	0.06	0.32	0.46	0.85	0.96	0.0	0.45	0.0	0.05
0.7	0.27	0.06	0.11	0.23	0.85	0.96	0.0	0.55	0.0	0.05
0.8	0.27	0.06	0.04	0.1	0.85	0.96	0.0	0.65	0.0	0.0
0.9	0.27	0.06	0.02	0.04	0.85	0.96	1.0	0.65	0.0	0.0

are non-zero. The distances between different rows therefore vary substantially, resulting in UCDD always observing different numbers of nearest neighbours, so always reporting drift, and in MSSW setting too wide acceptable average centroid distance boundaries, resulting in drift never detected for any testing batch. The trivial *Spam* dataset results were therefore omitted from Tables 4 and 5.

With MSSW, as is visible in Table 4, no drifts were detected accurately for real-world datasets, and these are the reasons. In *Weather*, ACC is below 12% in both experiments, because testing batches where SVM showed a low classification accuracy did not correspond to batches with significantly different average distances to cluster centroids. As expected, drift was never signaled in *Airlines* with one-hot encoding, because sparsity made boundaries for acceptable centroid distances too loose. With target encoding, drifts were defined to be in every testing batch except for the first one. FPR_R is 1.0 and the ACC is 0.05, because MSSW signaled this first batch and only one other testing batch as containing drift. In *ELECT2*, all batches were signaled as drifting, as *ELECT2* includes features representing dates whose values only increase. This increase results in testing samples always shifting further apart from reference data centroids, always increasing the total average distance to these centroids.

With UCDD, Table 5, the first column shows ten values of MRBD tried to see what influence this parameter has on results, and whether a high ACC and low FPR_R can be observed for any MRBD. The first, MRBD=1e-16, is low enough to signal drift whenever any reference batch signaled drift. For real-world datasets, we always used the additional check, because it has proven to be useful in synthetic datasets. From Table 5, we see that ACCs were either below 0.8, or higher than 0.8, but accompanied by FPR_Rs of at least 0.7. The only potentially acceptable results are in *Airlines*, where MRBD=0.3 gives 0.6 ACC and 0.0 FPR_R for target encoding, and MRBD=0.8 gives 0.65 ACC and 0% FPR_R for one-hot encoding. However, these two values of MRBD are very different, so MRBD is dataset and encoding-dependent, and finding the correct MRBD for a new drift detection problem is difficult.

5 Discussion

In this section, the results presented in the previous section are considered. First, the suitability of each algorithm for different problems under synthetic datasets is talked through, followed by a discussion of real-world results.

5.1 Synthetic data

For UCDD, Table 2, we first assess whether using the additional check is beneficial. We see that with the same MRBD=0.3, in each row, results with the check - column (a) - were always at least as good as those without it - column (b): FPR_Ses were always the same, but LTCs were often lower in column (a). Practitioners are therefore advised to always use this check for drift detection. Only columns (a) and (c) will now be considered for the analysis, and the check is used by default in real-world datasets.

We also discuss the effect of the choice of MRBD in UCDD in Table 2. Except for SEA, LTCs were always 1.0 when a higher MRBD of 0.6 was used. For the more complex AGRAW1/2 datasets, only an MRBD of 0.3 could therefore detect drift. In SEA, however, both columns (a) and (c) gave acceptable results: with MRBD=0.3, we observe FPR_Ses of 0.5 for drift widths up to 5.0, but also LTCs of 0.0, while with MRBD=0.6, FPR_Ses were always 0.0, but the maximum LTC in widths up to 5.0 increased to 0.25. The value of MRBD should therefore be set lower if late detections are a bigger problem, and higher if false positives are a bigger issue in a drift detection use case. The ideal value of MRBD for a new dataset could be found e.g. by observing the number of drift signals in reference batches for the first few testing batches, and then choosing MRBD such that both the FPR_S and LTC are low enough.

UCDD is dependent on encoding strategies. Based on the two previous paragraphs, for this analysis, we only look at column (a) in Table 2. In AGRAW2, the only encoding strategy detecting any drift was one-hot, and in AGRAW1, the two encoding strategies minimizing LTC and FPR_S for all drift widths were target and one-hot. UCDD therefore needs information from categorical variables by encoding them, and the most universal encoding that worked in both AGRAW1 and AGRAW2 is one-hot. However, in AGRAW2, the LTCs for most drift widths were 0.5, which might be too high for practitioners. Furthermore, classes were not estimated well with k-means in AGRAW1 and AGRAW2, so UCDD is not guaranteed to produce good results for more complex datasets than SEA. Further research is needed to assess whether the assumption of label estimates in complex datasets is too strong only for k-means, or for all unsupervised clustering algorithms.

As observed in Table 3, MSSW gives similar values of metrics for AGRAW1 and AGRAW2 when using the same encoding strategy. For both AGRAW1 and AGRAW2, the sparsity of one-hot encoding - column (b) where all LTCs were 1.0 - set too loose average centroid distance boundaries in reference data, so no drift was ever signaled. However, exclude (a) and target (c) both showed FPR_Ses of 0.0 and LTCs of at most 0.25 for drift widths smaller than 20.0, and the additional information from target encoding categorical variables always lead to LTCs in (c) being $<=$ LTCs in (a)

for same drift widths. These positive results might be caused by the entropy-weighted distance in k-means, which is made to lead to better clusters than plain k-means. Target encoding, so column (c), is therefore expected to always lead to the lowest values for both metrics, and because of the consistent results, based on synthetic datasets, MSSW is considered better suited for data with categorical values than UCDD.

Under abrupt drift (rows with width=0.0), UCDD - Table 2 - and MSSW - Table 3 - compare as follows. For `MRBD`=0.6, in SEA, UCDD (Table 2 column (c)) achieved lower LTC than MSSW. With a suitable `MRBD`, UCDD is therefore expected to achieve better abrupt drift results than MSSW on simple datasets such as SEA. MSSW (Table 3) showed LTCs of both AGRAW1 and AGRAW2 below 0.2 for target encoding (column (c)), while the LTC in AGRAW2 for UCDD was of 0.5. These results suggest that for more complex datasets with categorical features, MSSW with target encoding achieves consistently better results.

Based on our results, none of UCDD - Table 2 - and MSSW - Table 3 - was influenced much by drift widths smaller than 10.0, i.e. 10k samples. In UCDD SEA, the FPR_S and LTC stayed the same up to width=5.0 for `MRBD`=0.3 column (a), and LTC stayed below 0.25 for `MRBD`=0.6. In MSSW SEA, FPR_Ses and LTCs were the same for all widths. In UCDD AGRAW1 one-hot and target and AGRAW2 one-hot, FPR_S and LTC also stayed the same up to width=5.0, and in MSSW AGRAW1 target and AGRAW2 target, FPR_S was always 0.0, and LTC stayed below 0.2 up to width=5.0. An increase in LTC was observed for width=10.0 in most datasets with both algorithms: with UCDD in SEA with both `MRBD`=0.3 and `MRBD`=0.6, in AGRAW1 one-hot and AGRAW2 one-hot, and with MSSW in AGRAW1 target and AGRAW2 target. With the drift width being as high as the number of samples in one testing batch, the number of drifting samples in the first drifting batches diminished. This diminishment explains the increased LTC in both detectors. Both detectors are therefore deemed suitable for drift widths of less than the number of samples in one testing batch, but are expected to show a worse detection performance with larger drift widths.

5.2 Real-world data

While MSSW gave values of metrics of at most 0.25 in all synthetic datasets with exclude and target encodings, and the upper bound for most values of metrics in UCDD was 0.25 in SEA and AGRAW1 one-hot, if we look at Tables 4 and 5, both algorithms seem unsuitable for the real world. Since the drift detectors were not tried on real-world data before, and in the case of UCDD, the algorithm did not estimate class labels as it claims to in more complex datasets than SEA, it is therefore possible that these detectors are only suitable for datasets as simple as SEA, or for controlled, synthetically generated drifts. The low ACC could also be caused by the very small batch sizes in real-world experiments - as discussed in 5.1, detections of drifting batches of smaller size than drift width are not guaranteed.

Nevertheless, the suboptimal real-world detection performance was not necessarily caused by the drift detectors themselves. For example, defining drifting batches through a drop in a classifier's accuracy is not always consistent, as the accuracy can differ for different choices of classifiers. Additionally, the ACC metric requires drift to be detected at each exact batch where it occurred, so even with drift detected only one batch later every time, in the worst case, ACC can be 0.5. Further research could try alternative drift definition strategies and a comparison against existing label-dependent drift detectors to assess how much influence have drift definitions on the drift detection performance, and reach a conclusion of whether UCDD and MSSW are suitable in some real-world scenarios. Because of the differing results for synthetic and real-world datasets, we also conclude that evaluating new drift detectors only on synthetic datasets is not sufficient, so all upcoming research on new drift detectors is advised to include evaluation results with both types of datasets.

6 Responsible Research

This paper is concerned with a partial replication of work done in two other existing papers about drift detectors. Partial because we implement algorithms described in these papers, but evaluate them under a new setup. As such, we contribute to the scientific community by confirming that these two drift detectors are capable of detecting drift in a new scenario. Because we were also capable of detecting drift with the algorithms the replicated papers describe, we can conclude that these replicated papers are valid and their findings can be trusted. By evaluating these algorithms under a different setup and with different datasets, we also contribute by pointing out what are their limitations. This increased transparency and additional information on top of the replicated papers' findings could lead to better choices in future studies. For example, based on the newly found limitations, a future study might more easily assess whether or not to use the findings in the replicated papers.

To make sure the replication of the two algorithms is valid, we adhered to the following rules. Firstly, the papers describing them were properly cited. Secondly, the evaluation setup used in our paper, including the datasets used, was described in detail to make sure to have a proper distinction from the setup used in the replicated papers. This way, different or suboptimal results could be easily tracked down to the datasets and evaluation strategy, and could not be directly used to discredit the replicated papers. Thirdly and finally, everything concerning the implementation of the algorithms is explicitly mentioned: all our assumptions about the understanding of the replicated papers, all necessary adaptations and possible deviations from the original work, as well as the choices of technology, both in terms of programming language and in terms of libraries chosen for the implementations.

We ensured that our own results are reproducible by also including negative results, and by making our implementations publicly available. All results are either directly shown in the form of tables, or at least mentioned in text, which includes results where the algorithm did not perform well - in other words, no cherry picking was done to only show great results. This claim can be directly confirmed by consulting

the publicly available implementation repositories[13,14] and re-running our experiments. These repositories are linked to personal emails, not university or other time-limited emails, so they will remain active and open to consultation for a period of time not limited by e.g. the remaining study duration. The code for drift detection is in Python packages and .py files, which makes it accessible and easy to copy for anyone interested. There are also instructions for the technological setup needed to make sure anyone can run the code. Finally, the repositories store all datasets used for this paper.

7 Conclusion

In conclusion, this paper is an enhancement to the study by Lorena et al. [6], which is concerned with the comparison of the performances of multiple existing concept drift detectors - monitoring tools for "concept drift", which is a drastic unpredictable change in the underlying distribution of streaming data. This study is limited by the unavailability of implementations of unsupervised - i.e. label-independent - drift detectors. By implementing two existing unsupervised drift detection algorithms and making them publicly available, this and similar studies can be facilitated. Furthermore, ML practitioners can now choose to use such unsupervised drift detectors to detect drift without having access to actual labels in streaming data classification tasks. Finally, by replicating work done by other people, and by evaluating it on real-world datasets, through positive results obtained on synthetic datasets, we conclude that their work can be trusted, but might be limited to only synthetic datasets.

This paper aimed at describing the implementation process and finding out the detection performance of two unsupervised concept drift detectors: UCDD and MSSW, and the aim was to detect drift in fixed-sized testing batches. We have found out that under synthetic datasets, UCDD with a carefully chosen value for the `MRBD` parameter detected drift earlier in simple datasets of three numerical features, but k-means failed to estimate class labels for more complex datasets, which was reflected by a worse detection performance. On the other hand, MSSW detected drift consistently earlier than UCDD for two more complex datasets with six numerical and three categorical features when these categorical features were target encoded. The number of data samples from the moment a drift starts until it is fully established, i.e. the drift width, made UCDD and MSSW detect drift significantly later only when this width was at least as large as one testing batch. The results on real-world datasets were marginally worse for both algorithms than on synthetic datasets; the two methods are therefore likely not suitable for real-world problems. However, since results depend on the not necessarily consistent definitions of drifting batches in real-world data, further research with multiple drifting batch definition strategies and a comparison against supervised drift detections is advisable to confirm this conclusion.

[13]https://github.com/Jindrich455/clustering-drift-detection

[14]https://github.com/konstaka/drift-definitions

Additional remarks and acknowledgments

In computer science literature and papers, it is rarely the case for texts to be only written in passive form. In this paper, the trend of mixing passive and active form is therefore followed. Even though there is only one author, "we" is used as active form, because the usage of "I" is rare and seems unprofessional.

We would like to thank our supervisor and responsible professor for supervising our project. Our supervisor, Lorena Poenaru-Olaru, MSc has our gratitude for her guidance, her helpful advice, for her responsiveness, and for always reserving enough resources for our concerns. Furthermore, we would like to thank her for preparing and sanitizing all datasets used by this paper. Last but not least, we thank Dr. J. S. Rellermeyer for all his valuable feedback and suggestions.

References

[1] R. Rachita, "Streaming Analytics Market Size, Share and Trends." Allied Market Research, Portland, Oregon, USA, 2019. Accessed: Jan. 15, 2023. [Online]. Available: https://www.alliedmarketresearch.com/streaming-analytics-market.

[2] M. Pereira and B. Glisic, "Detection and quantification of temperature sensor drift using probabilistic neural networks," *Expert Systems with Applications*, vol. 213, p. 118884, Mar. 2023.

[3] Z. Yuan, H. Liu, J. Liu, Y. Liu, Y. Yang, R. Hu, and H. Xiong, *Incremental Spatio-Temporal Graph Learning for Online Query-POI Matching*. Feb. 2021.

[4] D. J. Kalita, V. P. Singh, and V. Kumar, "A novel adaptive optimization framework for SVM hyper-parameters tuning in non-stationary environment: A case study on intrusion detection system," *Expert Systems with Applications*, vol. 213, p. 119189, Mar. 2023.

[5] B. Gupta, A. Goyal, C. Sharma, and D. Kumar, "RERentFraud: A System to detect Frauds in rent payments for Real-Estate properties," in *Proceedings of the 6th Joint International Conference on Data Science & Management of Data (10th ACM IKDD CODS and 28th COMAD)*, CODS-COMAD '23, (New York, NY, USA), pp. 253–257, Association for Computing Machinery, Jan. 2023.

[6] L. Poenaru-Olaru, L. Cruz, A. van Deursen, and J. S. Rellermeyer, "Are concept drift detectors reliable alarming systems? - a comparative study," in *7th Workshop on Real-time Stream Analytics, Stream Mining, CER/CEP Stream Data Management in Big Data*, 2022.

[7] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under Concept Drift: A Review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, pp. 2346–2363, Dec. 2019. Conference Name: IEEE Transactions on Knowledge and Data Engineering.

[8] N. L. A. Ghani, I. A. Aziz, and M. Mehat, "Concept Drift Detection on Unlabeled Data Streams: A System-

atic Literature Review," in *2020 IEEE Conference on Big Data and Analytics (ICBDA)*, pp. 61–65, Nov. 2020.

[9] D. Shang, G. Zhang, and J. Lu, "Fast concept drift detection using unlabeled data," in *Developments of Artificial Intelligence Technologies in Computation and Robotics*, vol. Volume 12 of *World Scientific Proceedings Series on Computer Engineering and Information Science*, pp. 133–140, WORLD SCIENTIFIC, June 2020.

[10] Y. Yuan, Z. Wang, and W. Wang, "Unsupervised concept drift detection based on multi-scale slide windows," *Ad Hoc Networks*, vol. 111, p. 102325, Feb. 2021.

[11] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, *Learning with Drift Detection*, vol. 8. Sept. 2004. Journal Abbreviation: Intelligent Data Analysis Pages: 295 Publication Title: Intelligent Data Analysis.

[12] A. Liu, G. Zhang, and J. Lu, "Fuzzy time windowing for gradual concept drift adaptation," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–6, July 2017. ISSN: 1558-4739.

[13] V. Losing, B. Hammer, and H. Wersing, "KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 291–300, Dec. 2016. ISSN: 2374-8486.

[14] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Tracking recurring contexts using ensemble classifiers: An application to email filtering," *Knowledge and Information Systems*, vol. 22, pp. 371–391, Mar. 2010.

[15] Y. Lyu, H. Li, M. Sayagh, Z. M. J. Jiang, and A. E. Hassan, "An Empirical Study of the Impact of Data Splitting Decisions on the Performance of AIOps Solutions," *ACM Transactions on Software Engineering and Methodology*, vol. 30, pp. 54:1–54:38, July 2021.