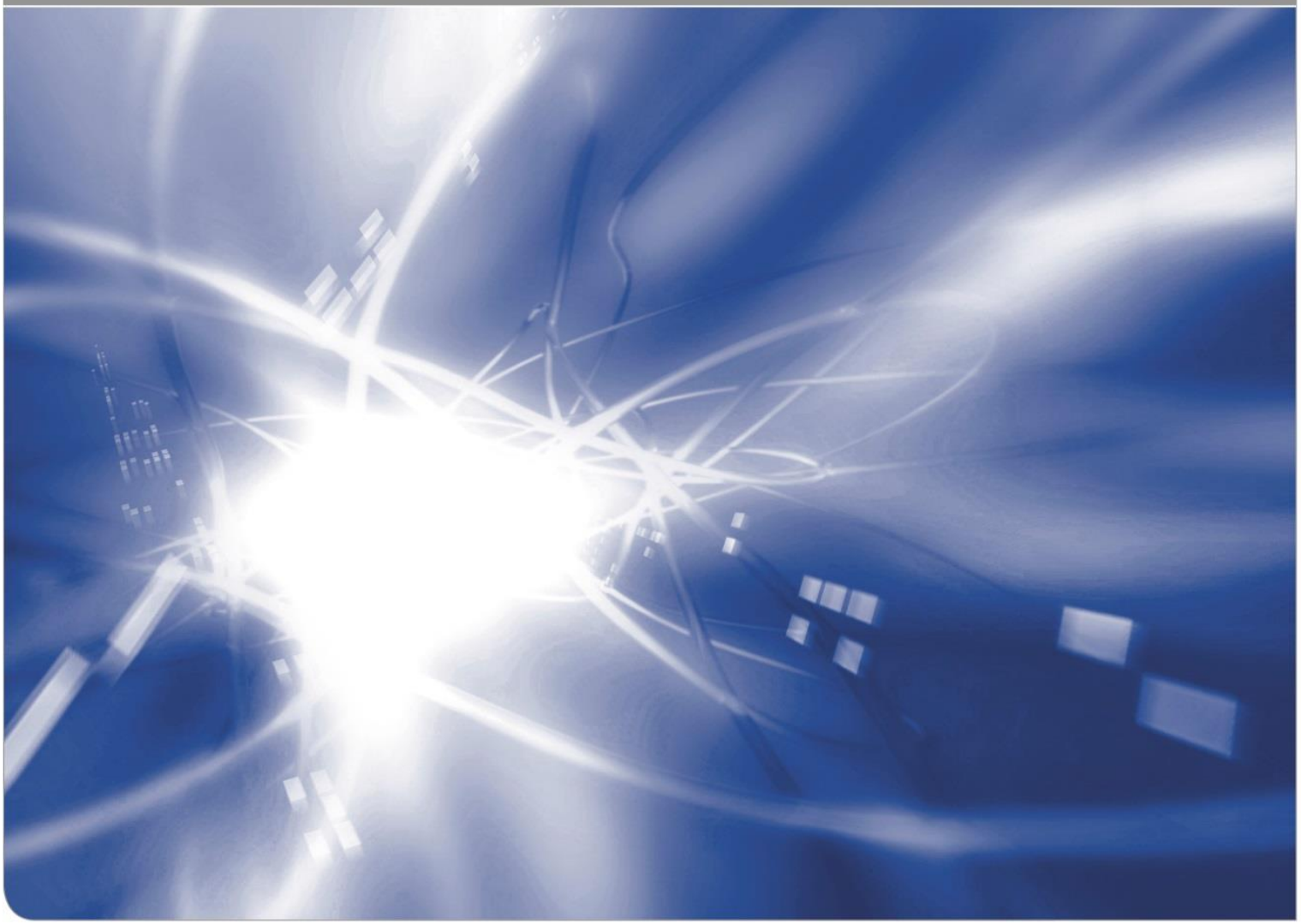
Karlsruhe Institute of Technology

Sensitivity-Based Optimization of Unsupervised Drift Detection for Categorical Data Streams

by Martin Trat, Janek Bender, Jivka Ovtcharova

KIT SCIENTIFIC WORKING PAPERS 208



Impressum

Karlsruher Institut für Technologie (KIT)
www.kit.edu

2023

ISSN: 2194-1629

SENSITIVITY-BASED OPTIMIZATION OF UNSUPERVISED DRIFT DETECTION FOR CATEGORICAL DATA STREAMS

Martin Trat, Janek Bender
Intelligent Systems and Production Engineering
FZI Research Center for Information Technology
Karlsruhe
{trat, bender}@fzi.de

Jivka Ovtcharova
Institute for Information Management in Engineering
Karlsruhe Institute of Technology
Karlsruhe
jivka.ovtcharova@kit.edu

ABSTRACT

Real-world data streams are rarely characterized by stationary data distributions. Instead, the phenomenon commonly termed as concept drift, threatens the performance of estimators conducting inference on such data. Our contribution builds on the unsupervised concept drift detector CDCStream, which is specialized on processing categorical data directly. We propose a cooldown mechanism aiming at reducing its excessive sensitivity in order to curb false-alarm detections. Using practical classification and regression problems, we evaluate the impact of the mechanism on estimation performance and highlight the transferability of our mechanism on other detection methods. Additionally, we provide an intuitive means for tuning the sensitivity of drift detectors. While only marginally improving the unaltered form of the detector on publicly available benchmark data, our mechanism does so consistently in almost all configurations. In contrast, within the context of another real-world scenario, almost none of the tested drift-detection-based approaches could outperform a baseline approach. However, potentially false-alarm detections are reduced drastically in all scenarios. With this resulting in a cutback in signals for refitting estimators, while maintaining a better or at least comparable performance to vanilla CDCStream, compute infrastructure utilization could be economized further.

***K*eywords** unsupervised concept drift detection · data stream mining · productive artificial intelligence · categorical data processing

1 Introduction

Technologies such as the internet of things, implemented as interconnected machines and sensors, demand analytical and predictive approaches to process data streams of eminently high speed and volume. The assumption that distributions associated to such data are stationary does often not hold, given the volatile nature of the processes generating it. Closely intertwined with non-stationary processes, the phenomenon commonly termed as concept drift can arise with different magnitudes, able to be defined on arbitrary scales, or dynamic profiles, such as incremental, abrupt or recurring drift (Webb et al., 2016). Especially estimators, being core components of data-driven services based on artificial intelligence, are under immense pressure to continuously adapt to the ever-changing environment of e.g. a shop floor and the degenerating impact of drift. Such adaptation needs to happen in-pipeline, hence highly automated and efficiently.

Therefore, it is of utmost importance to detect drifts and to mitigate their impact on the performance of estimators. Many prominent approaches assume the availability of ground truth information for detecting drift. However, this is often not realistic, with such information being either expensive to obtain or arriving with substantial delay (Gemaque et al., 2020). Unsupervised drift detection approaches target this problem. Using such approaches, one can only detect data drift (also known as virtual concept drift) (Oikarinen et al., 2021), which nonetheless threatens the performance of estimators (Oliveira et al., 2021). The reason for this is that data drift might develop into real concept drift (Khamassi et al., 2018) or other data-distribution-related circumstances such as previously underrepresented classes gaining importance.

In order to tackle these challenges, we contribute by augmenting an unsupervised drift detector, **which heavily suffers from false-alarm detections** in various data contexts. More specifically, we (i) suggest a mechanism for varying the drift detection sensitivity, which is transferable to other detection approaches as well, and (ii) outline a means to analyze unsupervised drift detection performance in detail without explicit knowledge on the presence and location of drifts. The gist of our proposed mechanism is the suppression of redundant subsequent detections, enabling the detector to readapt itself to a new concept after a drift detection. It is simple yet shows to be highly effective and does not tamper with the unsupervised detection character. Furthermore, instead of having to alter detector parameters requiring profound method knowledge, our proposed mechanism facilitates abstract and intuitive sensitivity tuning.

This work is structured as follows: A concise review of the body of related work and the methodical basis are given in section 2 and 3, respectively. Our contribution is introduced in section 4 and thoroughly evaluated in section 5. Finally, section 6 concludes this work and provides an outlook.

2 Related Work

The field of unsupervised (or at least semi-supervised) drift detection reveals itself as still under-researched, especially within the stream processing context. However, several recent works aim at closing that gap. Those could be categorized based on whether they assess every single incoming data instance or gather multiple ones in batches before assessing these for the occurrence of drift (Gemaque et al., 2020).

Within the drift detection and also related contexts, it is important to note that the nontrivial processing of categorical data lacks research as well. Reddy Madhavi et al. (2020) and Li et al. (2014) highlight the importance of drift-adaptive, iterative and efficient techniques for clustering such data. Reddy Madhavi et al. (2020) suggest a sliding-window-based method for that purpose but only briefly touch upon a means of making it more robust against drift, based on cluster distribution characteristics. Li et al. (2014) design several novel similarity measures and a drift detection approach. The latter is based on the occurrence of outliers in clustering results and similarity of pairs of such and constitutes only a part of their multi-faceted contribution.

A more distinct focus on drift detection however can be attributed to a series of works contributed by Cao and colleagues on clustering of categorical data streams (Cao et al., 2010; Cao and Huang, 2013; Cao et al., 2014). Based on rough set theory, Cao et al. (2010) propose a means of formulating the distance between two concepts, representable as clusters. By maintaining multiple sliding windows, this distance between two contiguous ones can be calculated. If it exceeds a predefined threshold, a drift detection is triggered and the involved windows are interpreted as containing different concepts. Based on a similar foundation, a revised distance formulation is defined by Cao and Huang (2013), additionally enabling the calculation of speed and magnitude of occurring changes. The solution proposes to discern drift via comparing the latter with a threshold separating non-drifting and drifting magnitudes. Cao et al. (2014) further extend both previous works by adding approaches enabling to link drifts to possible causes and to conduct trend analysis of drifts.

Ienco et al. (2014) also strive for the goal of facilitating drift detection on categorical data streams and introduce their unsupervised detector algorithm *CDCStream* (Change Detection in Categorical Evolving Data Streams). This approach is based on summarizing categorical data by calculating a statistic, based upon a distance function introduced in their previous work. Two different severity levels of drift, tied to bounds defined using Chebyshev's inequality, can then be determined via monitoring the difference between subsequent summary statistics. D'Ettorre et al. (2017) show how this algorithm can be slightly adapted by maintaining an ensemble of detector instances working on partially overlapping copies of the buffered stream data. Also, they focus on a vote-based strategy for discerning drift. They claim that this way, abruptly occurring changes can be detected with higher accuracy and shorter delay.

Plasse and Adams (2019) are also faced with the challenge of rapidly evolving categorical data streams. The goals of their work are improving the accuracy for determining the timing of drifts and accurately and continuously estimating distribution characteristics of the data stream by means of a forgetting mechanism. Within this context, they stress the importance of a grace period, during which their drift detector is not assessing data for the presence of drift and thus allowing it to readapt to new concepts.

In this work, we intend to augment the unaltered form of *CDCStream* (Ienco et al., 2014) by introducing a mechanism aiming at reducing its detection sensitivity in the presence of incremental drifts. We found this to be relevant, given that the detector reacts overly sensitive in various use cases and parametrizations we examined. Our mechanism is similar to above-mentioned grace period to the effect that the detector shall be granted time to readapt to newly developing data characteristics. However, we focus on evaluating the performance of estimators when using detections emitted by *CDCStream* as occasion to refit to data.

3 Drift Detection Algorithm

In this section, the basics for understanding the drift detection mechanism of *CDCStream* (Ienco et al., 2014) are briefly introduced[1].

3.1 Batch Summary Computation

For enabling the drift detection algorithm to process categorical data, several computations are done. Initially, a distance function suitable to factor in correlation and distribution characteristics of the categorical data is defined. A summary is then calculated for all data points in a batch and subsequently passed to the drift detector.

We start by assuming that, within the context of a streaming scenario, each entry in a batch of data, arriving at time t, D_t is defined over a set X of m categorical features X_i, denoted by $X = \{X_1, ..., X_m\}$. Each feature X_i can attain several discrete categories x, counted by its cardinality denoted as $|X_i|$.

The employed function Distance Learning for Categorical Attributes (DILCA) (Ienco et al., 2012) requires a context for calculating a distance. One or more features $X_i \in X$ can be selected as context for one other target feature $Y \in X$, formalized as $\mathcal{C}(Y)$, with $\mathcal{C}(Y) \subseteq X \setminus Y$. Based on all entries of a batch D_t, DILCA then assigns each pair of target feature categories y_u and y_v a scalar distance computed as

$$d_t(y_u, y_v) = \sqrt{\frac{\sum_{X_i \in \mathcal{C}(Y)} \sum_{x_k \in X_i} (P_t(y_u|x_k) - P_t(y_v|x_k))^2}{\sum_{X_i \in \mathcal{C}(Y)} |X_i|}}. \tag{1}$$

Equation (1) shows how the squared differences of the conditional probabilities P_t of each target feature category pair, inferred from batch D_t given a category x_k, are summed up for all x_k of a feature X_i. This step is then repeated within the context of a sum over all features belonging to the respective context $\mathcal{C}(Y)$, which ultimately equates to calculating a Euclidean distance. Before calculating the square root, the result is normalized based on the sum of the cardinalities of all features belonging to the context $\mathcal{C}(Y)$. Depending on the problem size, the selection of a suitable context can be difficult, which is why a correlation-based filter (Yu and Liu, 2003) is employed for this purpose. DILCA is then repeatedly applied in order to assemble a matrix M_{X_i} containing the distances computed for all category pairs of a feature X_i (and consequently having row and column size equal to $|X_i|$). With this process being repeated for each feature, a set M containing the m matrices M_{X_i} is assembled, subsequently serving as input for the calculation of the scalar summary statistic z_t representing data characteristics of D_t:

$$z_t(M) = \frac{\sum_{M_{X_i} \in M} \frac{2 \cdot \sqrt{\sum_{r=0}^{|X_i|} \sum_{s=r+1}^{|X_i|} M_{X_i}(r,s)^2}}{|X_i| \cdot (|X_i| - 1)}}{|M|} \tag{2}$$

The numerator on the right hand side of (2) is a compact formulation of summing up all squared values above the zero-valued diagonal, dividing the square root of the sum by the quantity of said values and repeating this step within the context of a sum over all matrices in M. Finally, the result is divided by the number of matrices $|M|$, leading to all z_t values always lying within the 0-1 range.

3.2 Detection of Warnings and Changes

The core component of this algorithm is based on Chebyshev's distribution-agnostic inequality (Alsmeyer, 2011) for random variables, borrowed from probability theory and commonly used for, in simplified terms, defining bounds on probabilities. With P representing a probability, it can be formulated as

$$P(|S - \mu| \geq k\sigma) \leq \frac{1}{k^2} \quad \text{for } k > 0. \tag{3}$$

Applied to our problem of statistical nature, (3) states that at most a fraction of $\frac{1}{k^2}$ of samples of a random variable S are expected to lie beyond the region enclosed within k standard deviations around the mean of S.

Based on (3), the algorithm *CDCStream* continuously evaluates whether incoming batches deviate statistically from insights of previously probed batches. Instead of memorizing all probed data, summary statistics z_t are computed for

[1]Our implementation, including the proposed mechanism, is available at https://github.com/fzi-forschungszentrum-informatik/cdcstream.

each batch D_t based on (2) and saved in an array-like continuously augmented history L. With a new batch arriving, z_t is computed and the content of L is checked. As it is empty exclusively on the initial invocation of *CDCStream*, subsequent calls check whether L contains either one or more than one element. In the latter case, mean μ and standard deviation σ are computed from the sample of past data represented by all $z_t \in L$, as required by (3). Also, extrema σ_{min} and σ_{max} are continuously maintained and updated as standard deviation σ is calculated. For k, following Ienco et al. (2014), two different instances k_w and k_c are used and kept at values 2 and 3 throughout this work, representing the bounds of the drift magnitude levels warning and change, respectively. Note that, based on (3), k adopting these two values and without further assumptions on the distribution of summary statistics z_t, one would expect at least 75% and roughly 89% of z_t to lie within the region bound by two and three standard deviations from the mean, respectively. This way, two differently tight and thus differently strict bounds are realized with k_w and k_c for determining discrete drift severity magnitudes. Next, in case of an issued change, no probing for a warning is done and history L is wiped. However, in the former case of L containing exactly one summary statistic z_t (computed from the previous batch), μ adopts this value and σ is calculated as arithmetic mean of σ_{min} and σ_{max}. If these extrema are not set, we are within the first cycles after *CDCStream* has initially been invoked and postpone drift probing to a later stage, when more batches will have been assessed. Finally, z_t of the currently probed batch is appended to L, leaving the latter never empty once started.

3.3 Shortcomings

As mentioned in section 2, we tested *CDCStream* in several scenarios using different parametrizations beforehand. As exemplarily shown in section 5.2, it can exhibit a very high detection sensitivity and often emits detections in a cascaded fashion, i.e. in short intervals, or even contiguously. We argue, that this might be a result of history L containing only a low number of summary statistics z_t after a change detection, which causes a history wipe as explained in section 3.2 above. Therefore, in this post-change period, *CDCStream* maintains a history containing characteristics of a potentially newly developing data distribution with very limited evidential value. Nevertheless, it still continues to assess data batches and is not blocked from emitting subsequent warning and change detections.

4 Cooldown Mechanism

Within the context of incremental drift scenarios, we argue that cascaded drift detections emitted by a temporarily unnecessarily sensitive *CDCStream* might be redundant, as new data distributions evolve slowly when compared to e.g. abrupt drift. We further expect that if the detector is granted time to readapt its history L in post-change periods and to gather characteristics on a developing data distribution without interruption, unnecessary detection cascades would be suspended.

From a set of possible solutions to achieve this, we choose to limit the sensitivity of *CDCStream* using a cooldown mechanism. This way, suffering from the lack of evidential value in a recently cleared history L, *CDCStream* is not forced to continue assessing data batches for the presence of drift. In principle, the cooldown mechanism causes *CDCStream* to skip drift (warnings and changes) assessment for a certain number of times. Therefore, each time a change is detected, a counter corresponding to a predefined number of invocations of *CDCStream* (thus each time a data batch is passed) is initiated. Then, for each subsequent detector invocation, applying (3) for comparing the statistics of history L with the summary statistic z_t that would have been computed from the current data batch is skipped if the value of the counter is greater than 0. After that, the counter value is decreased by 1. Once the counter value reaches 0, assessment for the presence of drift is resumed. For the remainder of this paper, the initial value of the counter is referred to as cooldown length. With it being set to 0, our augmentation of *CDCStream* corresponds to its unaltered form. As an example, the proposed mechanism is schematically depicted in Fig. 1 with the cooldown length being set to 3, represented as hatched area. In this figure, one can see how a change is being detected at time t_0, represented as a solid vertical line. Potential subsequent change and warning detections for an immediate successor batch arriving at $t_0 + 1$ and another one at $t_0 + 3$ are suppressed, represented as dashed vertical lines. At the end of the detector invocation at $t_0 + 3$, afore-mentioned counter reaches 0. Hence, drift detections are again possible with one exemplarily occurring at $t_0 + 4$.

The impact of this mechanism would mean less interference with adapting productively employed estimators due to unnecessary detection signals, without missing out on drift events of potentially disruptive consequences. Consequently, the risk of deploying under-fitted estimators, if training data for such were tied to the repeatedly small history of *CDCStream*, could potentially be mitigated. Also, omitting redundant estimator fitting reduces utilization of compute infrastructure accordingly. Without the cooldown mechanism, the need for downstream measures against false refitting signals would increase.

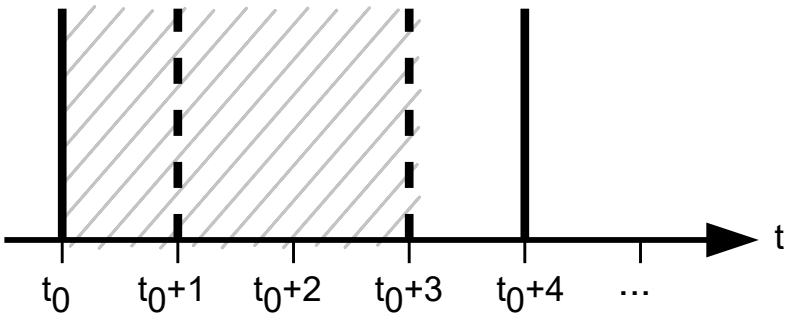


Figure 1: Exemplary depiction of the change detection behavior at cooldown length 3.

One might expect that an effect similar to that of our mechanism can be obtained by increasing the parameters k_w and k_c. Initially, we conducted a series of experiments in which we varied these parameters with small increments. Following this approach, we did not achieve productive results. Sensitivity control can only be exercised via adjustments within very narrow value ranges. However, the relation between the parameter and the sensitivity space is difficult to track, which renders tuning activities very cumbersome. In contrast, the suggested cooldown mechanism offers to do so intuitively without having to deviate from parameter values recommended by the detector's inventors. As we focus on the suggested approach, an extensive analysis of the impact of varying k_w and k_c is not documented in this work.

5 Evaluation

We consider supervised classification and regression problems and observe how the performance of the respective productive estimator is impacted by drift adaptation using *CDCStream* in a warning-change strategy context (Ienco et al., 2014), given various magnitudes of our proposed cooldown mechanism. An issued warning leads to a new background estimator being created and trained in parallel, whereas a change results in the substitution of the productive with the current background estimator. Each change additionally leads to the creation of a new background estimator. Note that, unshaken from the supervised character of the estimation problems, the drift detector is applied in an unsupervised fashion as no ground truth information or estimator performance valuation is utilized. Across all experiments, the parameters k_c and k_w, as defined in section 3.2, are kept at 3 and 2, respectively.

5.1 Employed Datasets and Estimators

Firstly, we evaluate our approach using the popularly employed open dataset comparing electricity market prices of the Australian state New South Wales with those of its neighboring state Victoria (*ELEC2*) (Harries, 1999). It roughly covers the years 1996 to 1998 and has 45,312 entries, eight features and a binary class variable indicating an increase or decrease in the price in New South Wales with respect to the mean price of the preceding 24 hours. All features, except the only categorical one indicating the weekday, are normalized to a 0-1 range. In order to compare the performance of our treatment in this scenario, we proceed as done by Ienco et al. (2014). Given all eight features, we estimate the class using a Naive Bayes classifier and discretize the non-categorical features into five identically wide bins for drift detection preprocessing. Furthermore, drift might occur due to weather fluctuations and the intrinsic price volatility over days, weeks and seasons, as well as the market's expansion into neighboring areas, also covered by the data (Harries, 1999).

Secondly, a lead time prediction case study (Bender et al., 2022) (labeled as company A in cited reference), conducted in a medium-sized enterprise, shall serve as a more practically oriented evaluation context (henceforth termed *LEADTIME*). Here, the goal is to predict the duration of various process types occurring within the manufacturing context of highly customised forging products. From a total of 99,526 entries, covering the years 2019, 2020 and several months of 2021, 5,186 ones representing standstill/waiting operations are filtered out and subsequently sorted by increasing values of the feature `date_registered`. Before dividing it into batches, the first 10% of the data is held out for initial estimator fitting. In contrast to the original case study, no inter-quartile-range-based filtering is applied. As estimator, we use a stacked estimator containing a least angle regressor and an ensemble of randomized decision trees, designated as the original case study's best result. Regarding this data, based on the available information and careful visualization-based analysis, we do not know whether drift is present. However, due to various factors, such as evolving customer needs,

we initially assume it. More details, e.g on the estimator parametrization, which we do not vary, can be obtained from Bender et al. (2022).

For regression estimator fitting, we resort to employing 21 different features, as done by Bender et al. (2022). All categorical features are label-encoded and then passed as numerical features, alongside genuinely numerical ones. Before being passed to the drift detector, the afore-mentioned feature set is augmented by the also available but previously unused features `customer_id`, `run_id`, `work_sequence_id_planned`. In contrast to estimator-related preprocessing, raw categorical features are used while numerical ones are discretized into five equally wide bins, as this work focuses on categorical data for drift detection.

5.2 Drift Detection Behavior

As mentioned in section 3.3, *CDCStream* has been observed to emit a high count of detections on certain parametrizations, phrased as detection cascades in this work. Therefore, we compare the drift detection evidence observed across all experiments in the following. As a metric, we focus on the change rate, which describes the number of detected changes as a fraction of the total number of possible drift detections and is consistently comparable across multiple experiment settings. Possible detections correspond to the number of seen data batches reduced by 1 as the first data batch cannot trigger a detection. For the sake of brevity and clarity, we provide absolute numbers for certain characteristic findings only. Even if it is not our goal to lower detection counts, we aim at suspending detection cascades, great numbers of detections are a symptom of.

Table 1 shows the change rate when applying *CDCStream* onto *ELEC2* data. Stream data of the four different batch sizes also used by Ienco et al. (2014) is processed. We limit ourselves to only documenting the results associated to these batch sizes here. Also, as warning detection behavior exhibits a trend highly similar to that of changes, we omit showing details here, which is further backed up by the fact that estimator substitutions are performed solely based on change evidence.

Table 1: Change rate observed when applying *CDCStream* onto the *ELEC2* dataset.

Batch size	Cooldown length							
	0	1	2	3	4	5	7	10
50	0.9680	0.0177	0.0110	0.0099	0.0099	0.0033	0.0077	0.0077
100	0.6460	0.0111	0.0111	0.0111	0.0088	0.0088	0.0088	0.0088
500	0.5506	0.0337	0.0225	0.0225	0.0225	0.0225	0.0225	0.0225
1000	0.4545	0.0682	0.0455	0.0682	0.0682	0.0682	0.0682	0.0682

The column for cooldown length set to 0 corresponds to a vanilla configuration of *CDCStream*, hence without our treatment. Note that we can reproduce the results from Ienco et al. (2014) with relatively small deviations. We deem it not necessary to analyze these further, as they might be artifacts from e.g. rounding errors in detection routines, resulting in marginally different change detection counts.

Detection cascades are especially apparent for batch size 50, resulting in 876 detected changes corresponding to a change rate of approx. 0.9680. The temporal dispersion of change detections can be described using the average distance measured in batches between such, with a distance of 0 indicating consecutive detections. In this case, an average distance of roughly 0.2121 batches, thus less than one batch, can be observed. However, applying the cooldown mechanism with length 1 and 2 yields a striking reduction to merely 16 and ten change detections, corresponding to change rates of approx. 0.0177 and 0.0110, respectively. In these two configurations, an average distance between detected changes of 56.4000 and 38.0000 batches can be observed, respectively. This precisely represents the effect the mechanism is intended to induce. Applying the cooldown mechanism with length 1 onto the 100, 500 and 1000 batch size configurations leads to change rates of approx. 0.0111, 0.0337 and 0.0682, respectively. When compared to the associated vanilla *CDCStream* findings, these translate into reductions by approx. 0.6350, 0.5169 and 0.3864, respectively. Increasing cooldown length from 1 to higher values does merely cause slight variation in change rate but are documented nonetheless. Apart from that, contrary to what is observed for vanilla *CDCStream*, our treatment within the context of batch sizes 500 and 1000 results in higher change rates than batch sizes 50 and 100.

In Table 2, observed change rates in the *LEADTIME* scenario are shown. We evaluated the same configurations regarding cooldown length while probing more different batch sizes than in the *ELEC2* scenario. The latter is done as we want to analyze detector behavior on a broader spectrum of batch sizes on this dataset, *CDCStream* has not been used on before. Regarding warnings, their associated trend is also highly similar to that of changes, which is why we do not show details in that regard.

Table 2: Change rate observed when applying *CDCStream* onto the *LEADTIME* dataset.

Batch size	Cooldown length							
	0	1	2	3	4	5	7	10
5	0.1265	0.0004	0.0003	0.0002	0.0002	0.0002	0.0002	0.0002
50	0.0094	0.0024	0.0024	0.0024	0.0024	0.0024	0.0024	0.0024
100	0.0542	0.0012	0.0012	0.0012	0.0012	0.0012	0.0012	0.0012
250	0.0118	0.0089	0.0059	0.0059	0.0059	0.0059	0.0059	0.0059
500	0.5357	0.0417	0.0179	0.0179	0.0179	0.0179	0.0060	0.0060
750	0.0179	0.0089	0.0089	0.0089	0.0089	0.0089	0.0089	0.0089
1000	0.9759	0.0241	0.0241	0.0241	0.0241	0.0241	0.0120	0.0241
1500	0.2727	0.0545	0.0364	0.0364	0.0364	0.0364	0.0182	0.0182

When analyzing change rates associated to vanilla *CDCStream*, one might notice more fluctuation for increasing batch sizes. A minimum of approx 0.0094 is found at batch size 50, whereas a maximum of approx. 0.9759 is found at batch size 1000, with several local maxima and minima in between. When considering our proposed cooldown mechanism, a drastic reduction in detected changes can be observed for batch size 5 and cooldown length 1. Here, the change rate drops by 0.1261 from approx. 0.1265 to 0.0004, which corresponds to going from 2148 detected changes down to merely six. This also shows the cooldown mechanism's impact in a rather extreme case regarding absolute numbers. Comparable to what is observed on *ELEC2* data, constant non-zero cooldown lengths generally result in higher change rates for higher batch sizes, given only few exceptions. Also, change rate variation for increasing non-zero cooldown lengths is observed to be rather low for constant batch sizes.

5.3 Drift Adaptation Performance

In all scenarios, we evaluate performance in the style of prequential evaluation: On arrival of each data batch, it is used for evaluating the estimator's performance and, subsequently, for incremental estimator fitting[2]. In order to enable comparable results across all experiments, we decided to consistently analyze the mean of the respective scenario's performance metric over all batches. As a strong baseline, we decided to fit the respective estimator incrementally and batch wise, without any forgetting of historical data. In other words, estimator adaptation is constantly done without taking any drift detection into account. Note, however, that this is computationally expensive and, due to the immediate necessity of ground truth, rarely possible in practical data streaming contexts.

The mean of the estimator's accuracy, achieved on all batches of the *ELEC2* scenario, serves as performance indicator[3]. The results are presented in Table 3. As a reminder, note that a cooldown length of 0 corresponds to applying *CDCStream* without our treatment. Accuracy values in this column coincide with and reflect the observations from Ienco et al. (2014) to the effect that batch sizes 50 and 100 result in better estimator performance than higher ones. This also holds when applying our treatment, regardless of its magnitude.

Table 3: Estimator accuracy observed during experiments on the *ELEC2* dataset. Bold values mark row maxima.

Batch size	No drift adaptation	Cooldown length							
		0	1	2	3	4	5	7	10
50	0.7250	0.7537	0.7506	**0.7545**	0.7543	0.7541	0.7496	0.7542	0.7542
100	0.7245	0.7472	0.7513	0.7513	0.7513	**0.7522**	**0.7522**	**0.7522**	**0.7522**
500	0.7247	0.7192	0.7411	0.7403	**0.7419**	**0.7419**	**0.7419**	0.7403	0.7403
1000	0.7196	0.7281	0.7285	**0.7291**	0.7285	0.7285	0.7285	0.7285	0.7285

The second column, indicating the absence of drift adaptation, shows that vanilla *CDCStream* outperforms the baseline for all batch sizes except 500. In contrast, when applying our cooldown mechanism, it outperforms the baseline for all batch sizes and, moreover, this is true for each probed non-zero cooldown length.

The configuration with cooldown length set to 2 and batch size 50 results in the greatest accuracy observed across all experiments on *ELEC2* data, valued at approx. 0.7545. This corresponds to an increase of approx. 0.0008, when

[2]Following Gama et al. (2014), for estimators not supporting incremental/partial fitting, we apply a workaround by fitting from scratch, providing the same data as bulk instead of batch wise, which leads to the identical estimator knowledge gain.

[3]Note that a different scoring metric is employed by Ienco et al. (2014).

compared to vanilla *CDCStream*. For batch size 100, cooldown lengths 4, 5, 7 and 10 equally result in the greatest accuracy with a value of 0.7522. This is explainable, as four changes were detected in these settings, with an average distance of 56 data batches between each other. In other words, change detections are spread out widely enough, so higher cooldown lengths do not impact subsequent detections. For batch sizes 500 and 1000, similar results are achieved. Here, one can highlight the greatest increase in accuracy of approx. 0.0227 being observed for batch size 500 and cooldown lengths 3, 4 and 5. Generally, across all probed batch sizes, we note that the highest estimator accuracy is always achieved with the cooldown treatment being applied (cooldown length greater than 0). Even further, the cooldown treatment always outperforms vanilla *CDCStream* for batch sizes 100, 500 and 1000. For batch size 50, this is the only case for five out of seven probed non-zero cooldown lengths.

Table 4 provides an overview of the batch-wise mean absolute error one can observe for the estimator's inference on the *LEADTIME* dataset. With this use case focusing on duration predictions, the advantage of this performance metric is that it is immediately intuitively comprehensible as its unit is given in minutes.

Table 4: Mean absolute estimation errors observed during experiments on the *LEADTIME* dataset. Bold values mark row minima.

Batch size	No drift adaptation	Cooldown length							
		0	1	2	3	4	5	7	10
5	**96.70**	122.06	115.46	115.34	115.15	115.15	115.15	115.15	115.15
50	**99.09**	114.77	113.65	113.70	113.70	113.70	113.70	113.70	113.70
100	**99.91**	110.74	108.90	108.90	108.90	108.90	108.90	108.90	108.90
250	**101.13**	107.93	104.25	104.30	103.77	103.77	104.20	104.20	104.20
500	**102.04**	113.21	105.15	106.98	106.20	106.20	106.00	105.42	105.42
750	**102.22**	107.22	103.70	106.10	106.10	106.10	106.10	106.10	106.10
1000	**102.92**	112.71	107.15	107.15	107.15	107.15	107.15	107.01	106.57
1500	103.08	109.91	106.15	106.07	106.07	103.27	103.27	**102.68**	**102.68**

In this scenario, as obtainable from the second column of Table 4, the baseline outperforms any drift-based estimator adaptation for almost all batch sizes. It achieves the lowest error across all experiments on this data at batch size 5 with a value of approx. 96.70 minutes. Solely the configuration with batch size 1500 and cooldown lengths 7 and 10, resulting in an error of approx. 102.68 minutes, marginally outperforms the baseline by roughly 0.40 minutes. Being faced with these findings, multiple explanations are conceivable. On the one hand, despite us initially assuming it, there might be no drift occurrences in the data. Given that the baseline model is trained on each batch after being tested on it, it might do so without being disrupted by changing concepts. If this is the case, one would expect to observe exactly the described result. On the other hand, as Bender et al. (2022) point out, the data acquisition practice might be error prone due to a high degree of manual intervention. In that case, any analytical approach might be limited in exploiting this data for the intended goals.

Nevertheless, regarding the configuration in which our treatment outperforms the baseline, a change rate of approx. 0.0182 can be observed as as shown in Table 2, corresponding to one change located in the first quarter of the stream. Simultaneously, this configuration also outperforms vanilla *CDCStream* by approx. 7.23 minutes, with the latter detecting 15 changes in this case, corresponding to a change rate of roughly 0.2727. The greatest improvement relative to vanilla *CDCStream*, with our treatment being applied, is achieved at batch size 500 and cooldown length equal to 1. In this case, the error is reduced by around 8.06 minutes from approx. 113.21 to 105.15 minutes.

Remarkable is that our proposed treatment (any non-zero cooldown length) strictly leads to lower errors for all probed batch sizes when compared to vanilla *CDCStream*.

6 Conclusion

We show that the suggested cooldown mechanism is able to successfully suspend drift detection cascades, i.e. detections in short intervals or contiguous ones, resulting in improved drift adaptation in all considered estimation problems. This way, we strictly outperform the unaltered version of the considered unsupervised drift detector *CDCStream*. Regarding the *ELEC2* scenario, this is evident as the best estimation performance is consistently achieved with the cooldown treatment being applied to the drift detector. Furthermore, for almost all batch sizes of streamed data, this is even the case regardless of the chosen magnitude of our suggested mechanism. On the *LEADTIME* scenario, both previous statements hold for all probed batch sizes, but only when compared with the unaltered form of *CDCStream*. On the latter dataset, the baseline, which does not take any drift detections into account but consistently fits the employed

estimator incrementally using each incoming data batch and associated labels, is only outperformed by *CDCStream* for batch size 1500 with our treatment being applied. For the remaining probed ones, the baseline exhibits better performance, which might arise from the possibility that the data does not contain any drifts. The intended reduction in drift detection sensitivity becomes apparent in our evaluation as detection counts considerably decrease in various configurations on both datasets. Moreover, the proposed mechanism is not limited to the considered drift detector but can potentially be employed for other detection methods that maintain a memory on evaluated data.

Based on the entirety of these observations, one might argue that a great number of originally detected drifts are no reliable indicators for estimator adaptation or forgetting of supposedly drifting data. Consequently, this benefits productive applications consuming drift detections. In terms of, for instance, drift adaptation, less estimator refitting would be triggered, hence unnecessary labelling costs and occupation of compute infrastructure could potentially be avoided. Similarly, valuable data could be prevented from being falsely flagged as outdated.

In future works, we intend to explore further research directions for drift detection approaches and means for evaluating such. Specifically, regarding the considered *LEADTIME* scenario, we remain uncertain about the existence of drift. In case it contains drift, Bifet (2017) argues that exclusively considering the performance within the context of a supervised estimation problem can lead to ambiguous insights. This circumstance could be mitigated using a set of detection-related metrics, which, however, would require ground truth information on the presence of drift. If we cannot obtain or infer such for our use case, synthesizing it is an avenue we plan to pursue, in order to evaluate our approach more profoundly, also in terms of i.a. detection accuracy and delay. Apart from that, being beyond the scope of this work, we will examine solutions on how to identify features exhibiting large portions of detected drift, by e.g. employing varying feature subsets.

Acknowledgements

We declare that this work was funded by the German Federal Ministry of Education and Research (grant number: 02K18D033, SEAMLESS project).

References

Alsmeyer, G. (2011). Chebyshev's inequality. In Lovric, M., editor, *International encyclopedia of statistical science*, Springer reference, pages 239–240. Springer, Berlin.

Bender, J., Trat, M., and Ovtcharova, J. (2022). Benchmarking automl-supported lead time prediction. *Procedia Computer Science*, 200:482–494.

Bifet, A. (2017). Classifier concept drift detection and the illusion of progress. In Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L. A., and Zurada, J. M., editors, *Artificial Intelligence and Soft Computing*, volume 10246 of *Lecture Notes in Computer Science*, pages 715–725. Springer International Publishing, Cham.

Cao, F. and Huang, J. Z. (2013). A concept-drifting detection algorithm for categorical evolving data. In Pei, J., Tseng, V. S., Cao, L., Motoda, H., and Xu, G., editors, *Advances in Knowledge Discovery and Data Mining*, volume 7819 of *Lecture notes in computer science Lecture notes in artificial intelligence*, pages 485–496. Springer, Berlin.

Cao, F., Huang, J. Z., and Liang, J. (2014). Trend analysis of categorical data streams with a concept change method. *Information Sciences*, 276:160–173.

Cao, F., Liang, J., Bai, L., Zhao, X., and Dang, C. (2010). A framework for clustering categorical time-evolving data. *IEEE Transactions on Fuzzy Systems*, 18(5):872–882.

D'Ettorre, S., Viktor, H. L., and Paquet, E. (2017). Context-based abrupt change detection and adaptation for categorical data streams. In Yamamoto, A., Kida, T., Uno, T., and Kuboyama, T., editors, *Discovery science*, volume 10558 of *Lecture notes in computer science Lecture notes in artificial intelligence*, pages 3–17. Springer, Cham.

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

Gemaque, R. N., Costa, A. F. J., Giusti, R., and Santos, E. M. (2020). An overview of unsupervised drift detection methods. *WIREs Data Mining and Knowledge Discovery*, 10(6).

Harries, M. (1999). Splice-2 comparative evaluation: Electricity pricing.

Ienco, D., Bifet, A., Pfahringer, B., and Poncelet, P. (2014). Change detection in categorical evolving data streams. In Cho, Y., Shin, S. Y., Kim, S., Hung, C.-C., and Hong, J., editors, *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 792–797, New York, NY, USA. ACM.

Ienco, D., Pensa, R. G., and Meo, R. (2012). From context to distance: Learning dissimilarity for categorical data clustering. *ACM Transactions on Knowledge Discovery from Data*, 6(1):1–25.

Khamassi, I., Sayed-Mouchaweh, M., Hammami, M., and Ghédira, K. (2018). Discussion and review on evolving data streams and concept drift adapting. *Evolving Systems*, 9(1):1–23.

Li, Y., Li, D., Wang, S., and Zhai, Y. (2014). Incremental entropy-based clustering on categorical data streams with concept drift. *Knowledge-Based Systems*, 59:33–47.

Oikarinen, E., Tiittanen, H., Henelius, A., and Puolamäki, K. (2021). Detecting virtual concept drift of regressors without ground truth values. *Data Mining and Knowledge Discovery*, 35(3):726–747.

Oliveira, G., Minku, L. L., and Oliveira, A. L. I. (2021). Tackling virtual and real concept drifts: An adaptive gaussian mixture model approach. *IEEE Transactions on Knowledge and Data Engineering*, page 1.

Plasse, J. and Adams, N. M. (2019). Multiple changepoint detection in categorical data streams. *Statistics and Computing*, 29(5):1109–1125.

Reddy Madhavi, K., Vinaya Babu, A., Sunitha, G., and Avanija, J. (2020). Detection of concept-drift for clustering time-changing categorical data: An optimal method for large datasets. In Raju, K. S., Senkerik, R., Lanka, S. P., and Rajagopal, V., editors, *Data Engineering and Communication Technology*, volume 1079 of *Advances in Intelligent Systems and Computing*, pages 861–871. Springer Singapore, Singapore.

Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., and Petitjean, F. (2016). Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994.

Yu, L. and Liu, H. (2003). Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, volume 2, pages 856–863.

KIT Scientific Working Papers
ISSN 2194-1629

www.kit.edu