Examining Computational Performance of Unsupervised Concept Drift Detection: A Survey and Beyond

Elias Werner
ScaDS.AI, CIDS, TU Dresden
Dresden, Germany
elias.werner@tu-dresden.de

Nishant Kumar
CGV, TU Dresden
Dresden, Germany
nishant.kumar@tu-dresden.de

Matthias Lieber
ScaDS.AI, CIDS, TU Dresden
Dresden, Germany
matthias.lieber@tu-dresden.de

Sunna Torge
ScaDS.AI, CIDS, TU Dresden
Dresden, Germany
sunna.torge@tu-dresden.de

Stefan Gumhold
CGV, TU Dresden
Dresden, Germany
stefan.gumhold@tu-dresden.de

Wolfgang E. Nagel
ScaDS.AI, CIDS, TU Dresden
Dresden, Germany
wolfgang.nagel@tu-dresden.de

ABSTRACT

Concept drift detection is crucial for many AI systems to ensure the system's reliability. These systems often have to deal with large amounts of data or react in real time. Thus, drift detectors must meet computational requirements or constraints with a comprehensive performance evaluation. However, so far, the focus of developing drift detectors is on detection quality, e.g. accuracy, but not on computational performance, such as running time. We show that the previous works consider computational performance only as a secondary objective and do not have a benchmark for such evaluation. Hence, we propose a set of metrics that considers both, computational performance and detection quality. Among others, our set of metrics includes the Relative Runtime Overhead *RRO* to evaluate a drift detector's computational impact on an AI system. This work focuses on unsupervised drift detectors, not being restricted to the availability of labeled data. We measure the computational performance based on the *RRO* and memory consumption of four available unsupervised drift detectors on five different data sets. The range of the *RRO* reaches from 1.01 to 20.15. Moreover, we measure state-of-the-art detection quality metrics to discuss our evaluation results and show the necessity of thorough computational performance considerations for drift detectors. Additionally, we highlight and explain requirements for a comprehensive benchmark of drift detectors. Our investigations can also be extended for supervised drift detection.

CCS CONCEPTS

• **Software and its engineering** → **Software performance**; • **Computing methodologies** → *Online learning settings*; Model development and analysis.

KEYWORDS

Concept Drift, Unsupervised Drift Detection, Computational Performance, Benchmark

ACM Reference Format:
Elias Werner, Nishant Kumar, Matthias Lieber, Sunna Torge, Stefan Gumhold, and Wolfgang E. Nagel. 2023. Examining Computational Performance of Unsupervised Concept Drift Detection: A Survey and Beyond. In *Proceedings of* . ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/xx.xxx/xxx_x

, ,
2023. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

In the last years, the amount of available data increased significantly due to the big data revolution. For instance, the collected data volume is expected to be about 175 ZB only for the year 2025 [38]. The availability of vast amounts of data and the exploit of computational resources like GPUs or TPUs led to the advent of deep learning (DL) methods in many application fields such as predictive maintenance [33], social media [29], marine photography [24] or transportation planning [16]. The effectiveness of such applications is often determined by the performance of the DL model on a different data distribution than the distribution on which the model was trained with. However, pure DL based applications work nicely on the training data distribution but do not perform well when the test data distribution is different from the training data distribution.

For example, Grubitzsch et al. [16] outlined that the reliability of such AI models is questionable for sensor data-based transport mode recognition. The reason is the variety of different context information, e.g. device type or user behavior that introduces drift into the data. Langenkämper et al. [24] demonstrated concept drift when using different gear or changing positions in marine photography and explained the effect on DL models. Hence, such applications need to be accompanied by approaches such as concept drift detection to estimate changes in the data distribution and to decide the robustness of a DL model on a given input.

Furthermore, applications must cope with large amounts of data or high-velocity data streams and react in real-time. On the other hand, applications are often bounded to certain hardware requirements or have to operate with limited computational resources. These observations should point to the necessity of thorough investigations concerning the running time, memory usage and scalability of a drift detector (DD). However, the literature focuses on the methodological improvements and evaluation on small-scale examples as outlined in the survey by Gemaque et al. [12]. Note that we refer to evaluation and metrics such as accuracy or recall as *detection quality*. We refer to metrics such as runtime or memory usage as *computational performance*. Also, theoretical computational complexities solely fail to capture the computational performance of an algorithm when deployed on real hardware and being applied to real data due to multiple factors such as implementation, compiler optimizations, data distribution and further external impacts. Only recently established machine learning benchmarks [34] emphasize the importance of computational performance evaluation in the wider data science community and offer a means to methodically study computational performance across various application domains and methodologies. While there is only one paper focusing on the computational performance of supervised DDs that require the availability of data labels, nothing is available for unsupervised DDs. Our work focuses on unsupervised DDs that operate in the absence of data labels, as we believe that the presence of labeled data is unlikely for many application scenarios. As there is no previous work investigating the computational performance of unsupervised DD, this work provides an initial examination of the available literature and the computational performance of related approaches.

The main contributions of this work are:

(1) We highlight that the previous literature lacks computational performance evaluation for unsupervised concept drift detection.
(2) We state the requirements for a comprehensive evaluation of DDs, reflecting computational performance and detection quality.
(3) We propose an initial set of metrics for a comprehensive evaluation of DDs and discuss our measurements of four related DD pipelines.

The rest of the paper consists of four parts. In section 2 we introduce preliminaries and give a definition for concept drift. In section 3 we give an outline of computational performance investigations for supervised concept drift detection. section 4 presents the prior works for unsupervised concept drift detection, discuss their important research contributions and explain the scope of previous computational performance evaluation. In section 5 we propose and discuss metrics for an evaluation that considers computational performance and detection quality. Furthermore, we highlight the necessity for thorough performance investigations based on the evaluation of four different DDs.

2 BACKGROUND

This section introduces the formal definition of concept drift and explains supervised and unsupervised concept drift detection.

In general, we follow the formal notations and definitions from Webb et al. [44] for the following illustrative equations. Moreover, we take also into account the publications by Gama et al. [11] and Hoens et al. [20] among others. Note that the next assumptions hold for the discrete and continuous realms in principle. Nevertheless, for ease of simplification, we consider only the discrete realm in our notations. Assume for a machine learning (ML) problem there is a random variable X over vectors of input features $[X_0, X_1, ..., X_n]$. Moreover, there is a random variable Y over the output that can be either discrete (for classification tasks) or continuous (for regression tasks). In this case, $P(X)$ and $P(Y)$ represent the probability distribution over X and Y respectively (priori). $P(X, Y)$ represents the joint probability distribution over X and Y and refers to a concept. At a particular time t, a concept can now be denoted as follows:

$$P_t(X, Y) \tag{1}$$

Concept drift happens when the underlying probability distribution of a random variable changes over time. Formally:

$$P_t(X, Y) \neq P_{t+1}(X, Y) \tag{2}$$

Supervised drift detection is the process where the data labels Y are always immediately available and unsupervised DDs detect drift without labeled data.

3 SUPERVISED CONCEPT DRIFT DETECTION

Many approaches for supervised drift detection have been developed in the last decades. Well-known surveys such as those by Gama et al. [11] or Barros et al. [2] summarize the work in the field. Moreover, [2] presents a large scale evaluation of related supervised DDs. However, they only consider detection quality aspects, e.g. accuracy and no computational performance aspects, e.g. running time of the compared methods. Only recently, [32] presented a benchmark of supervised DDs that considers the running time and memory usage of the related DDs besides the DDs' quality. However, they applied their procedure on small-scale datasets only and do not consider a thorough analysis of the computational performance evaluation. To the best of the authors' knowledge, there is no literature available that demonstrates the computational performance of supervised DDs on larger real-world datasets. Nevertheless, performance bottlenecks might be a problem as applications need to fulfill resource requirements when processing high data volume or high-velocity data and have to react in realtime. Thus, we believe that the computational performance aspects should also be considered as a main objective for supervised concept drift detection. One approach to deal with such resource requirements are parallel DDs. Although not investigated comprehensively, there are few works available discussing the scalability or parallelization of supervised DDs. One solution was developed by Grulich et al. [17] presenting a parallel version of the supervised DD Adwin with the ability to compute high data volumes with a high velocity. However, they are missing a comprehensive evaluation of their approach.

Even though the field of supervised concept drift detection can be investigated further, we focus on the unsupervised case in the rest of the paper. Unsupervised DDs gained a lot of attention in the last years due to their applicability in use cases where data labels are not available immediately.

4 UNSUPERVISED CONCEPT DRIFT DETECTION

In this section, we present a novel overview of unsupervised DDs that reflects on the computational performance instead of detection quality only. Our investigation is based on recently published survey articles and extends them. Furthermore, we discuss the respective computational performance and detection quality evaluation conducted by the prior work and show the necessity of thorough computational performance studies.

4.1 Existing Surveys

Gemaque et al. [12] presents an early taxonomy for drift detection that focuses on unsupervised DDs. The basis of their taxonomy is the accumulation and the updating process of windowed data that is used for detecting the drift. At the first level, they distinguish between batch-processed and online-processed drift detection before dividing the classes more specifically. A more recent survey on unsupervised DDs was conducted by Shen et al. [40]. They propose to separate the DDs into two categories based on the underlying method for drift detection. Approaches in group A are based on the differences in the data distribution. They either measure the sample density of different regions in the sample space or use statistical test methods to detect differences in the data distribution of a reference data set and a new data set. Thus, group A is further divided into regional density-based and statistical test-based methods. Approaches in Group B are based on model quality e.g. changing confidences, and detecting drift by monitoring and alerting changes in such model quality. Group B can be divided further into classifier-based methods, i.e. directly monitoring the quality of a base model or other model-based methods that use additional auxiliary means to detect drift, depending on the specific situation. Both surveys give an overview of related unsupervised DDs and highlight the versatility of the different approaches.

4.2 Computational Performance Considerations

Although both surveys mention the importance of computational performance considerations, they did not incorporate such objectives in their overviews thoroughly. Thus, we developed an overview that reflects on the computational performance of the related work by extracting the computational performance considerations which will be discussed next. Note that the evaluation concerning the detection quality might be different in the several papers. For our survey, we used the recent overview presented in [40] as a starting point, extended it with further works and aligned them to the taxonomy. Table 1 presents our survey results.

4.2.1 Investigated Features. In the first column of Table 1 we indicate whether such computational performance experiments were conducted. For approaches without such evaluation objective, it is difficult to assess the runtime, memory or energy consumption behavior in applications with vast amounts of data, high velocity data streams or limited computational resources. If computational performance experiments were conducted, we investigated three points. A) the objectives for the performance experiments. Those can be *Hyp*erparameters and their effect on the runtime of an approach. *Data* means, the approach is evaluated on different data sets and the computational performance is recorded. If the approach is compared with other DDs it is evaluated wrt to *Rel*ated approaches. B) the computational performance metrics that are investigated in the several works. This is the running time for most of the approaches and RAM-hour for one of the works. C) the data that was used for the computational performance experiments. We show the number of data samples, dimensions and the source of the data that was used for running time or RAM-hour evaluation. The last column indicates whether the source code of the approach is available.

4.2.2 Study Results. Several works [6, 8, 9, 13, 14, 19, 22, 23, 31, 35, 39, 46] do not conduct any runtime, memory, energy or scalability related performance measurements. Thus, it is difficult to assess their computational performance in real-world applications. Dasu et al. [7] and Gu et al. [18] conducted experiments concerning constructions' running time and updating their data representation. However, they do not consider the computational performance of the actual drift detection. Lu et al. [30] compared their approach with [7] but only concerning the data representation. The overall running time including the drift detection is unclear for those approaches. However, experiments by Qahtan et al. [37] showed that [7] has a linear increase in the running time of the data representation with growing window size and data dimensionality. Thus, they end up with a running time of 300 seconds for a data dimensionality of 20 and a window size of 10,000 samples. While [7, 18, 30, 37] evaluate the computational performance of their DD on synthetic data, Liu et al. [26, 27] used real-world data for the computational performance evaluation. Therefore, they ended up with data sets that contained fewer samples, but up to 500 dimensions.

Recently, [26] considered the RAM-hour metric to evaluate the memory consumption of the DD as recommended in [3]. Song et al.[41] create data sets based on real sources with 24 dimensions. They compare their method with two other approaches but with only a low amount of data and without a comprehensive experimental setup. Dos Reis et al. [10] conducted experiments on synthetic data to evaluate three different versions of the Kolmogorov–Smirnov test for streaming data. However, they did not compare their method to other approaches and considered only a small data set. Greco et al. [15] used real data for their evaluation and compared their approach with two other DDs but lacked a comprehensive evaluation setup as well. The most sophisticated running time performance evaluation was conducted by Pinagé et al. [36]. They tracked the running time of all the presented experiments on synthetic and real data. Moreover, they have leveraged the most extensive data set as per our survey, with over $4.9 * 10^6$ data samples and 41 dimensions. Experiments on this data set demonstrated high running times of several hours for related approaches.

4.3 Detection Quality Considerations

We skipped the data for detection quality or ML model quality evaluation in Table 1 since we focus on the computational performance considerations of the literature. Moreover, the evaluation of the related work in terms of the approaches' detection quality or ML model quality is mostly comprehensive and sound. Throughout the literature, many experiments investigating approaches' hyperparameters and their behavior on different data sets were conducted. There are also sporadic comparisons between different approaches in the single evaluation sections of the several works. However, there is no large-scale benchmarking across different DDs with a unified evaluation setting as it is available for supervised DD [2, 32].

4.4 Discussion

In future applications with high volume of data, high-velocity data or computational resource constraints, resource-efficient approaches and implementations are required. However, as outlined in the previous section, computational performance aspects for unsupervised concept drift detection were only investigated as a secondary objective in the literature. Only a few papers conducted running time

Table 1: Computational performance measures of the unsupervised DDs categorized as by Shen et al. [40]. Papers mentioned in *italic* were not considered by the original survey. Hyp = Investigation of hyperparameters, Rel= Comparison with related approaches, Data = evaluation on different data sets.

Paper	Experiments	Objectives	Metrics	Data for Computational Performance Evaluation			Source
				# Samples	# Dimension	Source	Code
Group A: differences in data distribution, regional density based							
Dasu et al. [7]	✓	Hyp	running time	5.000.000	4-10	Synthetic	✗
Gu et al. [18]	✓	Hyp, Rel	running time	100.000	2-10	Synthetic	✗
Qahtan et al. [37]	✓	Hyp, Rel	running time	5.000.000	2-20	Synthetic	✗
Liu et al. [27]	✓	Data, Rel	running time	9324, 18.159, 45.312	500, 8	Real	✗
Liu et al. [26]	✓	Data, Rel	running time, RAM-Hour	1500, 9324, 18.159, 45.312	99, 500, 8	Real	✗
Song et al. [41]	✓	Hyp, Rel	running time	100-7000	24	Real	✗
Group A: differences in data distribution, statistical test based							
Mustafa et al. [35]	✗						✗
Greco et al. [15]	✓	Hyp, Rel	running time	120.000	2	Real	✗
Kifer et al. [22]	✗						✗
Ditzler et al. [9]	✗						✗
dos Reis et al. [10]	✓	Hyp	running time	10.000	1	Synthetic	✓
Li et al. [25]	✓	Hyp	running time	10.000	1	Synthetic	✗
Group B: model quality monitoring, classifier output based							
de Mello et al. [8]	✗						✗
Haque et al. [19]	✗						✓
Lughofer et al. [31]	✗						✗
Sethi et al. [39]	✗						✗
Pinagé et al. [36]	✓	Hyp, Data	running time	Synthetic: 2k, 4k, 10k, 30k Real: 1901, 45.312, $4.9 * 10^6$	Synthetic: 2 Real: 20, 5, 41	Synthetic, Real	✓
Kim et al. [23]	✗						✗
Group B: model quality monitoring, other model based							
Gözüaçık et al. [13]	✗						✓
Gözüaçık and Can [14]	✗						✓
Lu et al. [30]	✓	Hyp	running time	5.000.000	6-20	Synthetic	✗
Zheng et al. [46]	✗						✗
Cerqueira et al. [6]	✗						✓

experiments in their evaluation, and only one investigated memory utilization. While the amount of data that was used in the respective evaluations, is sufficient to evaluate the DD's detection quality or the ML model's quality, it is not large enough to investigate the performance in terms of running time or memory usage. Moreover, the papers' evaluation settings are inconsistent and vary in the chosen data sets, number and dimension of data points and how the performance measurement is carried out. While this is comprehensible for the literature that presents novel methodological approaches, we need to investigate computational performance as a primary objective for productive DDs and real-world applications in future works. Although some papers consider the theoretical computational complexity of their algorithms, this can not replace such empiric measurements on real datasets and machines, e.g. as outlined by [21] highlighting the impact of computational performance bugs on the running times of implementations. Thus, we require consistent computational performance evaluations in order to assess the applicability of DDs for use cases with high volume of data and high-velocity data. Moreover, we should investigate scalable or parallel DDs and the resource-efficient deployment of the approaches in order to avoid waste of resources and to foster energy-efficient AI systems.

5 COMPUTATIONAL PERFORMANCE EVALUATION

To show the necessity of thorough computational performance considerations, we conducted several experiments with four different drift detectors on four datasets. First, we introduce the relevant computational performance metrics that we measured. Second, we introduce the evaluated pipelines and DDs, and present the datasets. Lastly, we show and discuss our experiment results. All experiments were performed five times and we report the average of the results. We also calculated the standard deviations but omitted them from the discussion as they were insignificant. All experiments were conducted on the HPC machine XXXXX at XXXXX. We used a single CPU core of an AMD EPYC 7702 CPU, fixed to 2.0 GHz frequency to improve reproducibility. Since the implementations do not run in parallel, we do not consider multiple CPUs or nodes.

5.1 Metrics

To assess the computational performance of a drift detector, we measure the overall runtime of the whole pipeline R_{Sum} and the runtime of the drift detector R_{DD}. For R_{Sum} we start the time measurement after loading the data and conducting an initial base model training. We end the measurement after processing the whole data stream. For R_{DD} we measure the time for everything, that is required to detect a drift and maintain the drift detector. We do not consider the drift handling, e.g. re-training of a base model, in case of a detected drift. Thus, we do not penalize a DD for an expensive base model or drift handling strategy. We can not compare R_{Sum} or R_{DD} across different approaches, since the default implementations are based on different programming languages and base models. Thus, we compute the Relative Runtime Overhead *RRO* to compare the DDs of the different pipelines as follows:

$$RRO = \frac{R_{Sum}}{R_{Sum} - R_{DD}}$$

Since the *RRO* is a relative measure, we can compare it across different approaches. It gives a first measure for the runtime overhead that is introduced by the DD in a pipeline. We assess the *RRO* wrt the different approaches and the initially proposed base models and pipeline components to compare the runtime overhead. It is important to mention, that the *RRO* is based only on the DD and not on the overhead that is introduced by the continuous re-training of the model. However, a DD triggers a drift handling method for each detected drift, e.g. model re-training. Thus, if a DD is too sensitive, the runtime overhead for the whole pipeline will increase, due to the low detection quality of the DD. Thus, it is important to reflect on both 1) the computational performance and 2) the detection quality of a DD. We also monitor the peak memory M_{peak} of the overall pipeline with profilers for the individual programming languages.

For real-world datasets, it is difficult to compute the detection quality of a DD directly, since the drift is always measured relative to a window of data samples that is maintained by the DD itself. Furthermore, this would require further pre-investigation of the individual datasets. Therefore, to reflect on the detection quality of the different DDs, we measure the accuracy metric *Acc* for the base ML models of the different pipelines. Accuracy is determined by $Acc = correct\ decisions / overall\ decisions$. Note that *Acc* does not represent the proportion of correctly detected drifts. Instead, it indirectly reflects the detection quality of the DD, since the accuracy of the ML base model is affected by true and false drift detections. Since a detected drift always triggers a drift handling, that might cause additional computational overhead, we count the number of detected drifts as *Detections*. Moreover, since some of the related approaches request true labels after a detected drift, we monitor the relative amount of requested data labels with the *ReqLabels* metric.

Table 2: Datasets that were used for the evaluation.

Dataset	Description	Size	Classes
Insects	Laser sensor data from flying insects	5325x50	5
Abrupt Insects	The same as Insects, but shuffled to introduce abrupt drift	5325x50	5
UWave Gesture	A set of eight gestures from accelerometers	4479x315	8
Forest Covtype	Cartographic features to determine forest cover type	581012x54	7
KDD CUP99	Network data containing intrusions and normal network traffic	$4.9 * 10^6$x41	2

5.2 Datasets

For our experiments, we use five different real-world datasets as described in Table 2. The datasets Insects [42] and Abrupt Insects [42] consist of 50 dimensions with five different classes that refer to different insects. Abrupt Insects was shuffled in a way, that it consists of abrupt drift, i.e. the sudden change in the data distribution. The UWave dataset [28] consists of 4479 samples with 315 dimensions that represent accelerometer motion data of eight different gestures. The Forest Covtype [5] dataset consists of 54 features with seven different forest cover type designations. The data in KDDCUP99 [43] consists of 41 dimensions and each sample is assigned to either the Normal or Intrusion class.

For the datasets Insects, Abrupt Insects and UWave we use 500 labeled samples initially available for the training of a base ML model. For the datasets Forest Covtype and KDDCUP99 we use 5000 labeled samples respectively. The rest of the data is used for the inference of the ML model and our experiments wrt concept drift detection and handling.

5.3 Evaluated Drift Detectors

From Table 1, six implementations are publicly available. We used the original Python implementations of [10] and [6], the original Java implementation of [19] and the original Matlab implementation of [36]. We used the default pipelines of the approaches, but adapted some hyperparameters, following the guidance of the respective papers. Moreover, we made minor changes in the codebases in order to make the implementations run with the different datasets and for conducting our measurements. The code for our experiments will be available publicly [1]. Note, for that initial investigation we skipped the implementations by Gözüaçık et al. [13, 14] since their pipelines conduct continual re-training of the ML model with the true labels. Thus, even if their drift detection is fully unsupervised, they rely on the immediate availability of data labels across their whole pipeline and a comparison with the other methods requires adaptions of their pipelines.

5.3.1 IKS. The Incremental Kolmogorov-Smirnov (IKS) was introduced by dos Reis et al. [10] and detects drift based on the changes in the raw input data distribution. Therefore, it constructs a reference window and a detection window and compares the data distributions within these windows with a Kolmogorov-Smirnov test. The default implementation for IKS operates on a single input feature. In the case of drift, we follow the strategy of model replacement that was suggested in the original paper among other drift handling mechanisms. The model replacement strategy requests true labels and trains a new model. For our evaluation, we use a Random Forest with 100 decision trees as a base classifier.

5.3.2 STUDD. Cerqueira et al. [6] presented STUDD, an approach that follows a student-teacher learning paradigm. It consists of a student auxiliary model to mimic the behavior of a primary teacher decision model. Drift is detected if the mimicking loss of the student model changes wrt the teacher's predictions. In case of a detected drift, the method updates the existing base model and student model with the requested true labels. As a basis for our evaluation, we use a Random Forest with 100 decision trees.

5.3.3 SAND. SAND is an approach for adapting and detecting novel classes over data streams and was introduced by Haque et al. [19]. The classification is done by K-means clustering, setting the value of K based on the size of the training data. Moreover, it consists of a concept drift detection that was introduced as Beta Distribution Change Point (BDCP) and operates on the confidences that ML model emits in the inference. BDCP reports drift in case of drops in the confidences. In the original paper, the authors recommend calling BDCP in cases, where the confidences fall under a certain threshold in order to reduce running time. Therefore, we used this procedure in our experiments. If a drift is detected, true labels are requested and the model is retrained.

5.3.4 PinagéDD. Pinagé et al [36] present DCS-LA+EDDM, a method that detects drift by monitoring the pseudo-error of an ensemble classifier. They use an ensemble of Hoeffding trees and select one of the ensemble members by DCS-LA [45] method to provide the pseudo ground truth for unlabelled samples. The pseudo-error is then calculated wrt this pseudo-ground truth. If the error changes, drift is detected and the ensemble is updated. DCS-LA+EDDM requires a training and a validation dataset for the internal pseudo-label generation. Therefore we separated the 500 samples into 350 samples training and 150 samples validation data for Abrupt Insects and Insects, 3500 and 1500 for Forest Covtype and KDDCUP99 respectively. Under the hood, DCS-LA+EDDM uses the supervised DD EDDM [1] based on the pseudo-labels. In case of a detected drift, the pseudo-labels are used and no true labels are required. For ease of naming, we refer to DCS-LA+EDDM as PinagéDD.

Additionally, we measure two baselines that use a Random Forest with 100 decision trees as a classifier. Baseline1 does not conduct any re-training and Baseline2 updates the classifier after 500 samples for Insects, Abrupt Insects and UWave 5000 samples for Forest Covtype, and KDDCUP99 respectively. Both pipelines are implemented with Python. The setup is described and provided by [6] and supports the assessment of the DDs' experiment results.

Table 3: Evaluation of the approaches on the different datasets. M_{Peak} is in Megabyte. A "-" for the baselines indicates, that there is no value for the metric available.

Dataset	Approach	Accuracy	Detections	ReqLabels	R_{Sum}	R_{DD}	R_{RO}	M_{Peak}
Insects	IKS	78%	5	89%	34s	5s	1.17	189
	STUDD	84%	0	0%	52s	26s	2.00	190
	SAND	99%	4	77%	4s	2s	2.00	196
	PinagéDD	78%	0	0%	143s	48s	1.50	629
	Baseline1	84%	-	-	25s	-	-	185
	Baseline2	81%	-	100%	28s	-	-	189
Abrupt Insects	IKS	85%	6	34%	32s	4s	1.14	185
	STUDD	83%	2	19%	57s	30s	2.11	193
	SAND	98%	5	68%	17s	15s	8.5	230
	PinagéDD	66%	1	0%	139s	49s	1.54	643
	Baseline1	65%	-	-	26s	-	-	182
	Baseline2	79%	-	100%	29s	-	-	187
UWave Gesture	IKS	90%	0	0%	23s	2s	1.09	221
	STUDD	90%	0	0%	43s	21s	1.95	223
	SAND	80%	11	72%	8s	2s	1.33	211
	PinagéDD	89%	0	0%	528s	4s	1.01	357
	Baseline1	90%	-	-	21s	-	-	222
	Baseline2	91%	-	100%	27s	-	-	248
Forest Covtype	IKS	85%	400	50%	5266s	1138s	1.27	704
	STUDD	68%	25	3%	10218s	6108s	2.48	1008
	SAND	94%	1493	87%	5176s	4910s	20.15	388
	PinagéDD	52%	8	0%	11970s	3248s	1.37	3756
	Baseline1	61%	-	-	4321s	-	-	446
	Baseline2	80%	-	100%	4432s	-	-	698
KDD CUP99	no termination within 60min							

5.4 Experiment Results

Table 3 shows our experiment results. We can see that the total absolute runtime of the approaches is high for all pipelines when dealing with the larger Forest Covtype dataset. For KDDCUP99, none of the approaches terminates within 60 minutes. However, we see

[1] https://github.com/

differences in the runtimes, i.e. the Java implementation of SAND is always the fastest and the Matlab implementation of PinagéDD is always the slowest. Since it is difficult to compare these absolute runtimes, the *RRO* provides a relative measure that reflects the overhead introduced by the drift detection. It ranges from 1.09 to 1.27 for the IKS. For STUDD we measure higher runtime overheads, i.e. 1.95 - 2.48 due to drift detection. The reason for this is the expensive maintenance of the student model. Furthermore, each detected drift is followed by drift handling, which increases the total runtime R_{Sum}. Also for SAND the *RRO* is high, i.e. 1.33-20.15, due to the fast inference times of the clustering base model and the relatively expensive computations for computing and maintaining the drift detection. For PinagéDD the *RRO* is lower, 1.01-1.54, because it uses a fast supervised DD under the hood based on pseudo-true labels. However, creating these can be expensive and lead the model training in the wrong direction, as the pseudo-label could still be wrong. Thus, PinagéDD does not require any true labels but has the lowest accuracy on average which can be even lower than Baseline1. The highest accuracy for all datasets, except UWave can be achieved with SAND. The reason for this could be the fast converging clustering model and the high amount of detected drift and requested true labels. Next to PinagéDD, STUDD requires on average the least amount of labeled data. It still achieves good accuracy on the Insects and Abrupt Insects datasets but drops off on the Forest Covtype data. IKS detects a lot of drift and requests many data labels. However, IKS only operates on a single input feature, and varying the input feature under investigation could result in different behaviors. Memory consumption is insignificant for all pipelines. In general, we observed a high sensitivity of all approaches to different hyperparameter settings. Thus, the interdependence of DDs, their hyperparameters, base models, datasets and underlying hardware resources should be further investigated. In addition, other methods of drift handling, such as α/β transformation described in [10], could be investigated.

5.5 Discussion

As demonstrated in the experiments, concept drift detection can be an important pillar to guarantee the robustness of AI applications. Without proper drift detection, the accuracy of the pipelines would decrease significantly as shown by Baseline1. Also, as Baseline2 shows, a pure re-training approach might not help to maintain the robustness of a ML application and might increase the runtime of the whole pipeline. However, from the publicly available approaches, there is no unsupervised DD available that fulfills the need for high detection accuracy, without many required true labels while having a low runtime overhead. A comprehensive benchmark of unsupervised DDs is not yet available but is required to assess the behavior of available DDs and to identify strengths and bottlenecks in current approaches. Hence, we propose to consider the following aspects when conducting a benchmark for unsupervised DDs.

5.5.1 Metrics. For drift detection, computational performance and detection quality are interrelated and depend on the data, the parameters of the DD and the base ML model in the pipeline, e.g. a DD that detects a high number of drift causes higher runtimes due to triggered drift handling. Therefore, the whole landscape of unsupervised DDs should be investigated from both perspectives: detection quality and computational performance. We proposed a first set of metrics to support this investigation. However, although the *RRO* provides an initial measure to compare different approaches across implementations, a fair comparison requires further steps.

5.5.2 Implementation and Methodology. We need proper implementations with the same programming language in order to make approaches comparable. Implementations should be publicly available and should be extensively documented. We also need theoretical considerations of the time and memory complexity of the approaches, and comparisons between them. Furthermore, as outlined in Table 1, some DDs operate on the pure input data, while others require the confidence or error rates of the base ML model. Thus, the benchmarking has to consider many different base classifiers and hyperparameter settings in order to obtain representative results for a comprehensive comparison between different DDs. Since the experimental space would be huge in this case, an alternative would be to simulate the output of a ML model. It would then be possible to investigate different DDs wrt requirements of a ML model and its inference quality. This would allow more specific investigations of DD and ML model combinations in an AI pipeline.

5.5.3 Datasets. In addition, benchmarks should be conducted based on different datasets from real-world domains and synthetic sources. While real-world data provide insights into the behavior in real applications, synthetic datasets are helpful in testing specific characteristics by defining patterns of concept drift in advance. In addition, it may be interesting to investigate the effect of data size per sample point, which can be reflected in experiments with various datasets.

Thus, a comprehensive benchmark supports the community in assessing available approaches and helps to develop novel solutions, including parallel and scalable implementations as proposed by Grulich et al. for the supervised DD ADWIN [17].

6 CONCLUSION

This work presents the first survey for unsupervised concept drift detection with a focus on computational performance. We highlight that computational performance is not represented comprehensively in the literature. We propose an initial set of metrics that reflect on both: detection quality and computational performance. Among others, it consists of the metric for relative runtime overhead to assess the computational performance of a DD. We show the necessity of such investigations by demonstrating the high runtime of the available approaches on larger datasets. We conclude that the computational readiness of contemporary DDs is questionable for future applications with lots of data and high-velocity streams. Thus, the scientific community requires comprehensive benchmarks across different DDs as well as scalable unsupervised solutions for concept drift detection.

For future work, we plan to extend our research by combining theoretical computational complexity with empirically measured computational performance, e.g. similar Big-O bounds as worst-case computational scenarios for algorithms, but different empirically measured computational performance of the implementation. Following this methodology helps to identify potential performance bugs as originally described in [21]. Furthermore, we want to develop a component for benchmarking DDs in the framework Massive Online Analysis (MOA) [4]. MOA is a state-of-the-art tool for analyzing and comparing data streaming tasks and online learning methods. It supports various tasks, e.g. classification, regression, out-of-distribution detection and concept drift detection. Thus, it is used to conduct the benchmarking of supervised DDs in [2] and [32]. However, it does not support unsupervised DDs yet. Nevertheless, it offers a consistent and unified benchmarking environment established in the scientific community. Based on this component, we want to conduct a comprehensive benchmark of all state-of-the-art DDs, beyond the initial experiments in this work. With the gained insights, we plan to develop parallel and resource-efficient solutions in the future.

ACKNOWLEDGMENT

This work was supported by the German Federal Ministry of Education and Research (BMBF, SCADS22B) and the Saxon State Ministry for Science, Culture and Tourism (SMWK) by funding the competence center for Big Data and AI "ScaDS.AI Dresden/Leipzig".

The authors gratefully acknowledge the GWK support for funding this project by providing computing time through the Center for Information Services and HPC (ZIH) at TU Dresden.

REFERENCES

[1] Manuel Baena-Garcıa, José del Campo-Ávila, Raul Fidalgo, Albert Bifet, Ricard Gavalda, and Rafael Morales-Bueno. 2006. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, Vol. 6. Citeseer, 77–86.

[2] Roberto Souto Maior Barros and Silas Garrido T Carvalho Santos. 2018. A large-scale comparison of concept drift detectors. *Information Sciences* 451 (2018), 348–370.

[3] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Eibe Frank. 2010. Fast perceptron decision tree learning from evolving data streams. In *Advances in Knowledge Discovery and Data Mining: 14th Pacific-Asia Conference, PAKDD 2010. Proceedings. Part II 14*. Springer, 299–310.

[4] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. 2010. MOA: Massive online analysis, a framework for stream classification and clustering. In *Proceedings of the first workshop on applications of pattern analysis*. PMLR, 44–50.

[5] Jock A Blackard and Denis J Dean. 1999. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture* 24, 3 (1999), 131–151.

[6] Vitor Cerqueira, Heitor Murilo Gomes, Albert Bifet, and Luis Torgo. 2022. STUDD: a student–teacher method for unsupervised concept drift detection. *Machine Learning* (2022), 1–28.

[7] Tamraparni Dasu, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi. 2006. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *Proc. Symposium on the Interface of Statistics, Computing Science, and Applications (Interface)*.

[8] Rodrigo F de Mello, Yule Vaz, Carlos H Grossi, and Albert Bifet. 2019. On learning guarantees to unsupervised concept drift detection on data streams. *Expert Systems with Applications* 117 (2019), 90–102.

[9] Gregory Ditzler and Robi Polikar. 2011. Hellinger distance based drift detection for nonstationary environments. In *2011 IEEE symposium on computational intelligence in dynamic and uncertain environments (CIDUE)*. IEEE, 41–48.

[10] Denis Moreira dos Reis, Peter Flach, Stan Matwin, and Gustavo Batista. 2016. Fast unsupervised online drift detection using incremental Kolmogorov-Smirnov test. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1545–1554.

[11] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.

[12] Rosana Noronha Gemaque, Albert França Josuá Costa, Rafael Giusti, and Eulanda Miranda Dos Santos. 2020. An overview of unsupervised drift detection methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 10, 6 (2020), e1381.

[13] Ömer Gözüaçık, Alican Büyükçakır, Hamed Bonab, and Fazli Can. 2019. Unsupervised concept drift detection with a discriminative classifier. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 2365–2368.

[14] Ömer Gözüaçık and Fazli Can. 2021. Concept learning using one-class classifiers for implicit drift detection in evolving data streams. *Artificial Intelligence Review* 54 (2021), 3725–3747.

[15] Salvatore Greco and Tania Cerquitelli. 2021. Drift lens: Real-time unsupervised concept drift detection by evaluating per-label embedding distributions. In *2021 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 341–349.

[16] Philipp Grubitzsch, Elias Werner, Daniel Matusek, Viktor Stojanov, and Markus Hähnel. 2021. AI-based transport mode recognition for transportation planning utilizing smartphone sensor data from crowdsensing campaigns. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 1306–1313.

[17] Philipp M Grulich, René Saitenmacher, Jonas Traub, Sebastian Breß, Tilmann Rabl, and Volker Markl. 2018. Scalable Detection of Concept Drifts on Data Streams with Parallel Adaptive Windowing.. In *EDBT*. 477–480.

[18] Feng Gu, Guangquan Zhang, Jie Lu, and Chin-Teng Lin. 2016. Concept drift detection based on equal density estimation. In *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 24–30.

[19] Ahsanul Haque, Latifur Khan, and Michael Baron. 2016. Sand: Semi-supervised adaptive novel class detection and classification over data stream. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[20] T Ryan Hoens, Robi Polikar, and Nitesh V Chawla. 2012. Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence* 1 (2012), 89–101.

[21] Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. 2012. Understanding and detecting real-world performance bugs. *ACM SIGPLAN Notices* 47, 6 (2012), 77–88.

[22] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. 2004. Detecting change in data streams. In *VLDB*, Vol. 4. Toronto, Canada, 180–191.

[23] Youngin Kim and Cheong Hee Park. 2017. An efficient concept drift detection method for streaming data under limited labeling. *IEICE Transactions on Information and systems* 100, 10 (2017), 2537–2546.

[24] Daniel Langenkämper, Robin Van Kevelaer, Autun Purser, and Tim W Nattkemper. 2020. Gear-induced concept drift in marine images and its effect on deep learning classification. *Frontiers in Marine Science* 7 (2020), 506.

[25] Bin Li, Yi-jie Wang, Dong-sheng Yang, Yong-mou Li, and Xing-kong Ma. 2019. FAAD: An unsupervised fast and accurate anomaly detection method for a multi-dimensional sequence over data stream. *Frontiers of Information Technology & Electronic Engineering* 20, 3 (2019), 388–404.

[26] Anjin Liu, Jie Lu, Feng Liu, and Guangquan Zhang. 2018. Accumulating regional density dissimilarity for concept drift detection in data streams. *Pattern Recognition* 76 (2018), 256–272.

[27] Anjin Liu, Yiliao Song, Guangquan Zhang, and Jie Lu. 2017. Regional concept drift detection and density synchronized drift adaptation. In *IJCAI International Joint Conference on Artificial Intelligence*.

[28] Jiayang Liu, Zhen Wang, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. 2009. uWave: Accelerometer-based personalized gesture recognition and its applications. In *2009 IEEE International Conference on Pervasive Computing and Communications*. 1–9. https://doi.org/10.1109/PERCOM.2009.4912759

[29] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, et al. 2022. Monolith: Real Time Recommendation System With Collisionless Embedding Table. *preprint arXiv:2209.07663* (2022).

[30] Ning Lu, Guangquan Zhang, and Jie Lu. 2014. Concept drift detection via competence models. *Artificial Intelligence* 209 (2014), 11–28.

[31] Edwin Lughofer, Eva Weigl, Wolfgang Heidl, Christian Eitzinger, and Thomas Radauer. 2016. Recognizing input space and target concept drifts in data streams with scarcely labeled and unlabelled instances. *Information Sciences* 355 (2016), 127–151.

[32] Mahmoud Mahgoub, Hassan Moharram, Passent Elkafrawy, and Ahmed Awad. 2022. Benchmarking Concept Drift Detectors for Online Machine Learning. In *Model and Data Engineering: 11th International Conference, MEDI 2022, Proceedings*. Springer, 43–57.

[33] Iñigo Martinez, Elisabeth Viles, and Iñaki Cabrejas. 2018. Labelling drifts in a fault detection system for wind turbine maintenance. In *Intelligent Distributed Computing XII*. Springer, 145–156.

[34] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. 2020. Mlperf training benchmark. *Proceedings of Machine Learning and Systems* 2, 336–349.

[35] Ahmad M Mustafa, Gbadebo Ayoade, Khaled Al-Naami, Latifur Khan, Kevin W Hamlen, Bhavani Thuraisingham, and Frederico Araujo. 2017. Unsupervised deep embedding for novel class detection over data stream. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 1830–1839.

[36] Felipe Pinagé, Eulanda M dos Santos, and João Gama. 2020. A drift detection method based on dynamic classifier selection. *Data Mining and Knowledge Discovery* 34 (2020), 50–74.

[37] Abdulhakim A Qahtan, Basma Alharbi, Suojin Wang, and Xiangliang Zhang. 2015. A PCA-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 935–944.

[38] David Reinsel, John Gantz, and John Rydning. 2018. Data age 2025: the digitization of the world from edge to core. *Seagate* (2018).

[39] Tegjyot Singh Sethi and Mehmed Kantardzic. 2017. On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications* 82 (2017), 77–99.

[40] Pei Shen, Yongjie Ming, Hongpeng Li, Jingyu Gao, and Wanpeng Zhang. 2023. Unsupervised Concept Drift Detectors: A Survey. In *Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery: Proceedings of the ICNC-FSKD 2022*. Springer, 1117–1124.

[41] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. 2007. Statistical change detection for multi-dimensional data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 667–676.

[42] Vinicius MA Souza, Denis M dos Reis, Andre G Maletzke, and Gustavo EAPA Batista. 2020. Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery* 34 (2020), 1805–1858.

[43] Salvatore Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip Chan. 1999. KDD Cup 1999 Data. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C51C7N.

[44] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. 2016. Characterizing concept drift. *Data Mining and Knowledge Discovery* 30, 4 (2016), 964–994.

[45] Kevin Woods, W. Philip Kegelmeyer, and Kevin Bowyer. 1997. Combination of multiple classifiers using local accuracy estimates. *IEEE transactions on pattern analysis and machine intelligence* 19, 4 (1997), 405–410.

[46] Shihao Zheng, Simon Zon, Mykola Pechenizkiy, Cassio Campos, Werner Ipenburg, and Hennie Harder. 2019. Labelless Concept Drift Detection and Explanation. In *NeurIPS 2019 Workshop on Robust AI in Financial Services*.