Analysis of Mixed Concept Drift Detectors in Deployed Machine Learning Models

Toma Zamfirescu

Supervisors: Jan Rellermeyer, Lorena Poenaru-Olaru

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

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

Name of the student: Toma Zamfirescu
Final project course: CSE3000 Research Project
Thesis committee: Jan Rellermeyer, Lorena Poenaru-Olaru, Jesse Krijthe

Abstract

Label-independent concept drift detectors represent an emerging topic in machine learning research, especially in models deployed in a production environment where obtaining labels can become increasingly difficult and costly. Concept drift refers to unforeseeable changes in the distribution of data streams, which directly impact the performance of a model trained on historical data. This paper initially focuses on two *mixed* label-independent drift detectors, SQSI and UDetect, which are implemented and evaluated on a specific setup using synthetic and real-world data sets. Next, multiple label-dependent drift detectors are evaluated on real-world data sets, and the results are compared to those of the label-independent detectors. This paper presents a framework for comparing multiple concept drift detectors on different data sets and configurations, checking whether they can be reliably used in a production environment.

1 Introduction

Machine learning is taking over the technology world, allowing us to get valuable insights, mathematically predict outcomes, and classify data. In practice, machine learning models are trained on some data and then deployed in a production environment. A model initially trained on some data may no longer reflect reality after some time - the distribution of the real-world data changes continuously, so the model's accuracy decreases. The recent COVID-19 pandemic influenced many aspects of human behavior, so models that, for instance, predicted the number of hospital admissions, the number of clients in a restaurant, or sales of drug stores cannot rely on pre-COVID-19 data - this is an extreme example of concept drift [8]. Because of this, there needs to be a way to spot those changes or inconsistencies, and based on those, the model should be retrained to produce accurate results at all times. To do this, we introduce the notion of **concept drift**, which describes unforeseeable changes in the underlying distribution of streaming data over time [13]. To spot concept drift, we need to implement some algorithms called **concept drift detectors**. A general framework for drift detection, separated into four stages is shown in Fig. 1 [13].

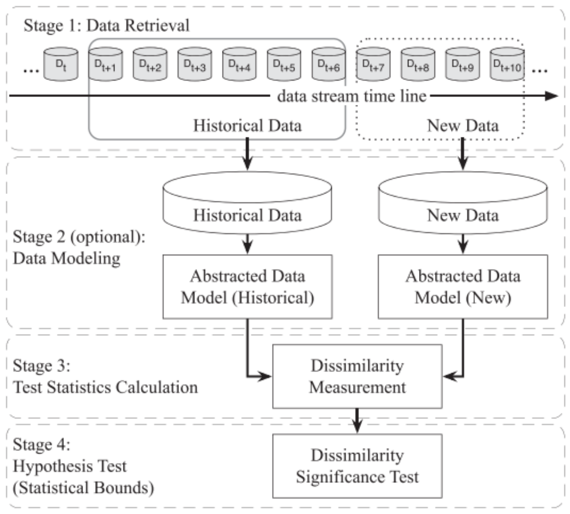

Stage 1: Data Retrieval
D_t
D_{t+1}
D_{t+2}
D_{t+3}
D_{t+4}
D_{t+5}
D_{t+6}
D_{t+7}
D_{t+8}
D_{t+9}
D_{t+10}
data stream time line
Historical Data
New Data
Stage 2 (optional): Data Modeling
Historical Data
New Data
Abstracted Data Model (Historical)
Abstracted Data Model (New)
Stage 3: Test Statistics Calculation
Dissimilarity Measurement
Stage 4: Hypothesis Test (Statistical Bounds)
Dissimilarity Significance Test

Figure 1: Framework for drift detection [13]

This paper emphasizes the analysis of **label-independent** concept drift detectors, meaning that we assume we do not have any labels other than those for the training data. Labeling data can become increasingly difficult, costly, or sometimes impossible in a production environment. Despite this problem, there is little research done for label-independent detectors, and code is rarely shared with the scientific community [17]. Label-independent concept drift detectors are commonly categorized as follows: based on data distribution, error-rate drift, multiple hypotheses [13], or mixed [12], which is a combination of the previous ones. This paper will focus on the latter, the **mixed concept drift detectors**.

The **research questions** that this paper will aim to answer are the following:

- **RQ1**: How well do mixed concept drift detectors identify concept drift in the case of synthetic and real-world data?
- **RQ2**: How do mixed concept drift detectors perform compared to other label-dependent detectors?

The main focus is to implement two label-independent detectors, evaluate them using some metrics on both synthetic and real-world data, and compare the results with some already-implemented label-dependent detectors available in different Python libraries.

The main contributions of this paper are the following:

- Implementation of two label-independent concept drift detectors, with the code publicly available on GitHub at https://github.com/tzamfirescu/concept-drift-detection.
- Replication of the experiments of two label-independent concept drift detectors, SQSI [15] and UDetect [4].
- Evaluation of label-independent and label-dependent concept drift detectors on real-world data.

The paper is organized as follows: Section 2 presents the related work, the foundation of this research. Sections 3 and 4 define the methodology and results, respectively. Section 5 is the discussion, where some interesting insights are highlighted. Section 6 tackles responsible research, while Section 7 concludes the paper.

2 Related Work

Concept drift detection is a necessity in deployed machine learning models, so extensive research was performed on this topic, especially as part of data stream monitoring [13]. There are multiple categories of label-independent concept detectors, as presented in [13]. *Error-rate-based* detectors represent the largest category, and they track the changes in the online error rate of base classifiers. Then, depending on the significance of the change, a drift alarm is signaled. The second largest category is *data distribution-based* detection, where algorithms use a distance metric to quantify the dissimilarity between training and testing data. *Multiple-hypothesis test* algorithms apply similar principles to the previous two and can be divided into parallel and hierarchical multiple-hypothesis tests. *Mixed* detectors exhibit hybrid characteristics of previous types. Concept drift can be categorized as **abrupt** (or sudden, instantaneous) and **gradual** [20], the difference being the time interval over which drift occurs. As shown in Fig. 2, in the case of abrupt drift, the context (or concept) changes suddenly, and the data becomes stationary right away. In the case of gradual drift, it takes more time until the context changes from A to B. This transition period is also called **drift width**.

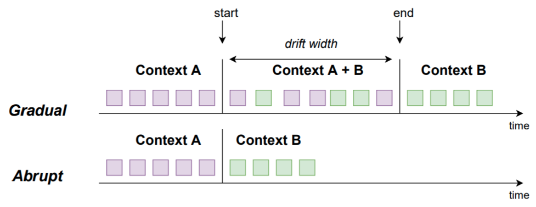


Figure 2: Difference between drift types [17]

The two concept drift detectors that are implemented and analyzed next are called *Data Stream Quantification by Score Inspection* (SQSI) [15], and the second *Unsupervised Change Detection for Activity Recognition* (U-Detect) [4]. SQSI is a mixed detector because the algorithm presented in the paper performs a two-step test, checking if the distribution of the classifier errors has changed. If drift is detected using the first statistical test, the classifier is retrained, and the same statistical test is performed again. Similarly, UDetect is considered mixed because it defines a two-level distribution-based framework [11], and it does not need training labels at all - it only computes a summary value for the training features, which is then compared to the summary values of each batch. Other examples of mixed concept drift detectors are part of the multiple-hypothesis detectors category. Linear Four Rate drift detector (LFR) keeps track of changes in the true positive, true negative, false positive, and false negative rates [21], performing parallel multiple hypothesis tests. Hierarchical Linear Four Rate (HLFR) is a hierarchical framework with LFR as the detection layer, followed by the zero-one loss over the train-test split as the validation layer [23].

3 Methodology

The main goal of this paper is to show the accuracy of label-independent detectors and compare them to other more reliable label-dependent ones already available in Python. To do this, we need to define the specific data setup in 3.1, some data sets suited for that in 3.2, the label-independent detectors 3.3, the supervised detectors we will run our experiments with in 3.4, and the evaluation metrics used for all experiments in 3.5.

3.1 Data Setup

For all the experiments conducted as part of this research paper, we process each data set as a data stream, separating training data from testing data. The testing data is then divided into equal-sized batches. The algorithms will check in which testing batch we have concept drift, considering only the training data as a reference. This data setup was proposed and reviewed by [17] and is summarized in Fig. 3.

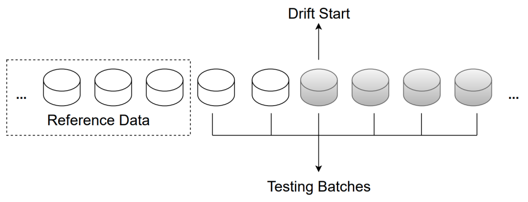


Figure 3: Data Setup [17]

3.2 Data Sets

Within this paper, we use both synthetic and real-world data, with the scope of observing the performance of the detectors in as many setups as possible. All data is assumed to come in a data-streaming manner, meaning that samples are ordered by time.

Synthetic Data Sets
The synthetic data sets are generated by two data generators available in the MOA framework, SEA, and Agrawal [17]. Therefore, we have three types of data sets: SEA, AGRAW1, and AGRAW2. SEA contains three numerical features, ranging from 0 to 10, while the AGRAW data sets contain six numerical and three categorical features, with each sample representing the data of a person requesting a loan. The difference between the two data sets generated by Agrawal [1] represents the subsets of features where data is shifted. Since concept drift can be either abrupt or gradual, for each of the three types, we have six data sets - one with abrupt drift and five others with gradual drift, where the concept drift occurs over 500, 1000, 5000, 10000, and 20000 samples (drift width). In other words, the data distribution gradually drifts over several samples until it becomes stationary again.

Each synthetic data set has 100k samples. For training, we use 30k samples, and the 70k remaining ones are divided into seven equal batches of 10k. For all these data sets, the drift was set to start at sample 55000 in the data generation process. Considering this, the goal is for our detectors to signal all batches **from 3 to 7** as drifted.

Real-World Data Sets
For our experiments, we used four real-world data sets - *ELECT2*, *Airlines*, *Weather* and *spam* [7], [5]. Each has a different number of samples, so the data setup differs.

ELECT2 predicts whether electricity prices go up or down and has 45 312 samples, with 15 104 used for training and 30 208 for testing, which are divided into batches of 365 samples. *Weather* predicts whether it is raining or not and has 18 159 samples, with 6 053 as training data and the rest as testing data. The evaluation will be done with batches of 30 samples (monthly prediction) and 365 samples (yearly prediction). *Airlines* predicts if a flight is delayed or not and has 539 383 samples, with 179 794 training samples and the rest as testing. Here, the data is divided into batches of 17 000, representing the number of flights per day. Finally, *spam* predicts whether an email is spam or not and contains 4 405 samples, with 1 468 as training and the last 2 937 as testing. The testing data is divided into batches of 20, 50, and 100.

3.3 Label-Independent Concept Drift Detectors

First Label-Independent Detector - SQSI

Data Stream Quantification by Score Inspection, also called SQSI [15], is a complex algorithm that, besides detecting the moment where concept drift occurs, also retrains the model to keep it up-to-date. The main focus of this algorithm is not concept drift but quantification, which is the task of computing the distribution of a data stream. This detector can be categorized as mixed because it is data distribution-based but does not directly compare the distribution of the labels but the distribution of cross-validation accuracies. The pseudo-code of this algorithm, excluding the retraining part, which we do not need, is shown in Fig. 4.

```
Algorithm 1
    Input: path, training size, testing batch size, encoder, scaler
    Output: testing batches with concept drift
 1: procedure GetConceptDrift
 2:     data ← ReadFile(path)
 3:     training-features, training-labels ← GetTrainingData(data)
 4:     classifier ← TrainClassifier(training-features, training-labels)
 5:     scores-training ← CrossValidation(classifier, training-features, training-labels)
 6:     detected-batches ← [ ]
 7:     for each testing batch do
 8:         scores-batch ← PredictProbabilities(classifier, testing-data).
 9:         p-value ← KolmogorovSmirnov(scores-training, scores-batch)
10:         if p-value <= 0.001 then
11:             We have found drift!.
12:             add batch-number to detected-batches
13: return detected-batches
```

Figure 4: SQSI algorithm [15]

The algorithm first trains a classifier based on the training data, using *SVM* or *Random Forest*. *SVM* would take significantly more time on each data set, so all experiments were conducted using only *Random Forest*. Then, based on the trained classifier, *k-fold cross-validation* is performed on the training data, where the probability of each class occurring is computed for each event. For the cross-validation, leave-one-out is recommended in [15], but $k = 20$ was used because of time constraints. The distribution of scores generated by the cross-validation should be compared with similar scores computed on the testing data. For each testing batch, we calculate the distribution similarly - for each event, we want the probability of each class occurring. We do this by using the same classifier we initially trained. As soon as we have the two scores, we need to apply a statistical test to check if they are from the same data distribution. To do that, we use the Kolmogorov-Smirnov test, with a significance of 0.001. If the p-value of the test we applied is lower than 0.001, we found concept drift.

Second Label-Independent Detector - UDetect

UDetect is an unsupervised concept drift detector proposed by [4]. The algorithm is structured as a two-level architectural framework that includes an offline and online component and assumes no presence of ground truth - it does not need the labels of the samples. Similarly to SQSI, UDetect also performs the retraining of the model after drift is found, which we will not include in the implementation.

The offline component is separated into multiple layers and involves the training data. In the windowing and segmentation sub-layer, data is partitioned into fixed-size data chunks. Each chunk is then summarized into a single parameter, called *summary value*, representing the mean distance from each point to the centroid. In the change parameter computation layer, the algorithm calculates the Shewhart individual control chart parameters [22]. The change parameters are the lower control limit (LCL_T) and the upper control limit (UCL_T) for the individual window summary values.

$$LCL_T = T - A_2 * R \tag{1}$$

$$UCL_T = T + A_2 * R \tag{2}$$

where T is the mean of the window summaries, R is the mean range of the subgroups, and A_2 is a constant specific for $\tilde{X}$-Chart [22].

Finally, in the model generation sub-layer, a classifier prototype is built to recognize the current activity pattern in the system. In the online component of the algorithm, the algorithm signals when a significant change has occurred in the new classified instances by determining if the data classified to a class is widely divergent from the change parameters calculated in the offline component. If the null hypothesis is rejected, we have found the concept drift. The pseudo-code of this algorithm, excluding the part that deals with retraining the algorithm, is shown in Fig. 5.

```
Algorithm 2
    Input: path, training size, testing batch size, encoder, scaler
    Output: testing batches with concept drift
 1: procedure GetConceptDrift
 2:     data ← ReadFile(path)
 3:     training-features, training-labels ← GetTrainingData(data)
 4:     $T$ ← ComputeWindowSummary(training-features)
 5:     $UCL_T$, $LCL_T$ ← ComputeChangeParameters(training-features)
 6:     detected-batches ← [ ]
 7:     for each testing batch do
 8:         $w_s$ ← ComputeWindowSummary(batch)
 9:         if $w_s <= LCL_T$ or $w_s >= UCL_T$ then
10:             We have found drift!
11:             add batch-number to detected-batches
12: return detected-batches
```

Figure 5: UDetect Algorithm [4]

UDetect was implemented and evaluated on a completely different setup than ours - the detector with the same constants as defined in [4] never detects drift for the synthetic data sets we defined in 3.2. In the same paper, the subgroup size of the training data is 3, which does not return accurate results for big data, proven by no detection over all the synthetic data sets. Also, the constants used for the $\tilde{X}$-Chart in UDetect are not clearly explained, so finding suitable constants based on different subgroup sizes was difficult. The subgroup size was increased to 25, and the constant A_2 for the control charts was taken from [18]. Besides this, R in a control chart typically represents the range of a window summary, so the difference between the largest and smallest elements [18]. On the other hand, UDetect takes a different approach, with R being the difference between consecutive window summaries, which is different than other definitions of $\tilde{X}$-Charts [18].

3.4 Label-Dependent Concept Drift Detectors

Since the data sets provide labels for the testing data, we can compute the drifted batches using label-dependent (supervised) concept drift detectors and compare the results to the reference drifted batches we defined in Section 4.2.

The concept drift detectors used for the evaluation of real-world data are *error rate-based drift (ERB)* detectors and are implemented in the *river* Python library. We will apply the most popular *ERB* detector DDM (*Drift Detection Method*) [10], which uses statistical tests to signal drift, along with its improved version EDDM (*Early Drift Detection Method*) [2], which additionally checks the distance between error rates when signaling drift. ADWIN (*Adaptive Windowing*) [6] uses a window-based method to store the samples based on time, and when the mean of the stored values decreases, drift is detected. Finally, we will use the *drift detection method based on Hoeffding's inequality*, using an A-test (HDDM_A) and W-test (HDDM_W), respectively [9]. These label-dependent detectors will be evaluated using the same setup mentioned in 3.1 and using the same evaluation metrics.

3.5 Evaluation Metrics

To evaluate the concept drift detectors, we need different metrics for the synthetic and real-world data sets. In the case of synthetic data sets, the assumption is that as soon as drift occurs, all subsequent batches are also drifted. In this case, we only need to check if the detector finds the first drifted batch at the right time. In the case of real-world data sets, we cannot hold this assumption since drift may happen at any batch in the testing set. Because of that, we define two metrics for each type of data set, proposed by [17] - false positive rate (FPR_S) and latency (L) for the synthetic data sets, and false positive rate (FPR_R) and drift detection rate (DDR) for the real-world data sets.

FALSE POSITIVE RATE for the **synthetic data sets** (FPR_S) shows the ratio of non-drifted batches detected as having drift. If no batch that does not contain drift is signaled as drift, then $FPR_S = 0$, but if all batches that do not contain drift are signaled, then $FPR_S = 1$. Otherwise, the formula for FPR_S is the following:

$$FPR_S = \frac{|F|}{|B_B|} \tag{3}$$

where $|F|$ is the number of batches detected as having drift earlier than supposed, and $|B_B|$ is the number of batches **before** the drift.

LATENCY L ranges between 0 and 1 and shows how late the detector managed to detect drift. If drift is detected earlier or at the right time, latency is 0, but if it is detected later, the formula for L is the following:

$$L = \frac{k-j}{|B_A|}; b_j, b_k \epsilon B \tag{4}$$

where b_j is the batch corresponding to the beginning of concept drift, b_k is the batch detected as drift, and $|B_A|$ is the number of batches **after** the drift.

FALSE POSITIVE RATE for the **real-world data sets** (FPR_R) is similar to FPR_S, but the rate is computed over all batches, not only by considering batches before the actual drift. The formula for FPR_R is the following:

$$FPR_R = \frac{|F|}{|B_N|} \tag{5}$$

where $|F|$ is the number of non-drifted batches that were detected as drifted, and $|B_N|$ is the number of non-drifted batches.

DRIFT DETECTION RATE (DDR) is the ratio between the number of batches that were correctly signaled as drift and the total number of batches with drift. The formula for DDR is the following:

$$DDR = \frac{|C|}{|B_D|} \tag{6}$$

where $|C|$ is the number of drifted batches that were **correctly** detected as drifted, and $|B_D|$ is the number drifted batches.

3.6 Implementation

The synthetic and real-world data sets, implementation of the two detectors, and the code for finding the reference drift for real-world data are publicly available at https://github.com/tzamfirescu/concept-drift-detection. The code was written in Python on the *Jupyter Notebook* development environment, using *pandas* and *numpy* for data modeling, and *Scikit-learn* for most of the machine learning algorithms [16].

Some experiments were also run on the TU Delft's DAS-6 (The Distributed ASCI Supercomputer 6) cluster [3], where GPU nodes could be used in parallel for faster computation. A guide on how to connect to DAS-6, run a *Jupyter Notebook (or Lab)* on multiples nodes, and access the notebook from your machine can be found on GitHub. The problem is that the Python libraries initially used to implement the algorithms cannot use GPU resources. The *cuml* and *cupy* libraries can replace *pandas*, *numpy*, and *Scikit-learn* so that the code works on a single GPU node. Using multiple GPU nodes in parallel meant adapting the whole code using different *RAPIDS* [19] libraries, which was not feasible given the short time of the research.

4 Results

This section outlines the accuracy of all detectors on both synthetic and real-world data sets and also presents a framework for computing the *reference drifted batches* in the real-world data sets. To observe their general behavior and ensure that the two concept drift detectors SQSI and UDetect work, they were initially implemented and tested on synthetic data in 4.1. In 4.2 we run the detectors on real-world data, and finally evaluate the label-dependent concept drift detectors in 4.3. In Tables 1, 2, 3, and 4, in case no concept drift was detected for a specific configuration, we mark it as *ND (Not Detected).*

4.1 Synthetic Data Sets

Since we deal with both categorical and numerical features, what first needs to be done in the data modeling phase is to apply encoders to categorical data; both algorithms only work on numerical features, so not using an encoder would mean leaving some features out, which of course lowers the accuracy of the model. *One-hot*, *ordinal*, and *target* encoders are used for the categorical features, but also *no encoder* by removing all categorical features. Also, features are both scaled using the *min-max scaler* and not scaled. The following experiments show whether the detectors work, their accuracy, and whether they are data set, encoder, or scaler dependent.

When experimenting with the SQSI detector during the implementation process, the inconsistency in results was obvious: there were often different results with the same configuration. The randomness of the cross-validation most likely causes this, so to solve this and have reliable results, the algorithm was run 50 times on each data set. The false positive rate (FPR_S) and latency (L) for each data set is the average of those metrics over all the runs. Both detectors were run with all the encoders mentioned earlier, and with data both scaled and not scaled. Over 1200 runs, SQSI had $FPR_S = 0.102$ and $L = 0.02$, as shown in Table 1. UDetect had an average $FPR_S = 0$ and $L = 0.15$ over 29 different configurations; it detected drift slightly later than supposed but never earlier. Considering this, results prove that SQSI and UDetect manage to accurately detect the first drifted batch in the synthetic data sets, given the low number of testing batches - only seven.

Abrupt Drift

Results in Table 1 show that SQSI performed very well on data sets with abrupt drift, with average latency close to 0 and a false positive rate of 0.13. For the SEA data set, which only contains numerical features, both metrics were 0, meaning that drift was always detected at the right batch. After observing the AGRAW data sets results, we note that SQSI is encoder-dependent; the best results are obtained using the *ordinal* encoder, and the worst using the *one-hot* encoder. Removing categorical features had better results than *one-hot* and *target* encoders. On average, scaling the data lowered the run time and FPR_S, but not by big margins.

			SQSI		UDetect	
Data Set	**Features Encoder**	**Min-Max Scaler**	**FPR**	**Latency**	**FPR**	**Latency**
AGRAW1	None	No	0.18	0	ND	
		Yes	0.14	0	0	0
	OneHot	No	0.61	0	ND	
		Yes	0.52	0	0	0
	Ordinal	No	0.16	0	ND	
		Yes	0.21	0	0	0
	Target	No	0.23	0	ND	
		Yes	0.25	0	0	0
AGRAW2	None	No	0.04	0.005	0	0.2
		Yes	0	0	0	0
	OneHot	No	0.04	0	0	0.2
		Yes	0.09	0	0	0.2
	Ordinal	No	0.01	0	0	0.2
		Yes	0	0	0	0.2
	Target	No	0.005	0.03	0	0.2
		Yes	0.01	0	0	0.4
SEA	None	No	0	0	0	0.2
		Yes	0	0	0	0.2

Table 1: Results of SQSI and UDetect on synthetic data sets with **abrupt drift**

On average for the abrupt data sets, UDetect had $FPR_S = 0$ and $L = 0.2$. Results in Table 1 prove that this detector is not necessarily encoder dependent, but performs differently depending if the features are scaled or not. For AGRAW1, UDetect never detected drift unless features were scaled, so even though not scaling the features of AGRAW2 and SEA did not influence the outcome, AGRAW1 proves that UDetect is scaler-dependent. The detector managed to signal the right batch for AGRAW1, but would mostly signal one or two batches later for the other data sets.

Gradual Drift

For data sets with gradual drift, SQSI was only evaluated using the *ordinal* encoder, the one that performed best for abrupt drift, and again, features were both scaled and not scaled. For the SEA data sets, SQSI performed perfectly, with a false positive rate and latency of both 0. In the case of both AGRAW1 and AGRAW2, the latency increases as the width of the drift increases - when the width of the drift is 10k or 20k samples, SQSI detects one batch later than supposed. In the case of AGRAW1, the overall FPR_S is 0.2 and L is 0, while for AGRAW2 FPR_S is 0 and L is 0.1. The results for both SQSI and UDetect are highlighted in Table. 2.

The results in Table 2 show that UDetect is not as sensitive to gradual drift - false positive rate is 0 and latency is quite stable, between 0 and 0.2 over most widths, and only increasing to 0.4 for AGRAW2 when the width is 20000. In the case of SEA, the width of the drift did not influence at all the results, the detector signaling the drift one batch later in all runs. Unlike SQSI, which would either predict perfectly

				SQSI		UDetect	
Data Set	Features Encoder	Width of Drift	Min-Max Scaler	FPR	Latency	FPR	Latency
AGRAW1	Ordinal	500	No	0.24	0	ND	
			Yes	0.14	0	0	0.2
		1000	No	0.21	0	ND	
			Yes	0.22	0	0	0.2
		5000	No	0.24	0	ND	
			Yes	0.36	0	0	0
		10000	No	0.14	0	0	0.8
			Yes	0.17	0	0	0.2
		20000	No	0.16	0.04	ND	
			Yes	0.19	0.04	0	0.2
AGRAW2	Ordinal	500	No	0	0.01	ND	
			Yes	0.04	0.005	0	0.2
		1000	No	0.05	0.005	0	0.2
			Yes	0.04	0	0	0
		5000	No	0.04	0	ND	
			Yes	0.03	0.05	0	0
		10000	No	0.03	0.215	ND	
			Yes	0.02	0.22	0	0.2
		20000	No	0	0.24	0	0.2
			Yes	0	0.225	0	0.4
SEA	None	500	No	0	0	0	0.2
			Yes	0	0	0	0.2
		1000	No	0	0	0	0.2
			Yes	0	0	0	0.2
		5000	No	0	0	0	0.2
			Yes	0	0	0	0.2
		10000	No	0	0	0	0.2
			Yes	0	0	0	0.2
		20000	No	0	0	0	0.2
			Yes	0	0	0	0.2

Table 2: Results of SQSI and UDetect on synthetic data sets with **gradual drift**

or slightly earlier than supposed, UDetect tends to predict one batch later. Similarly to the data sets with abrupt drift, not scaling the features produced worse results for AGRAW1 and AGRAW2 - seven configurations resulted in ND (Not Detected) in this case.

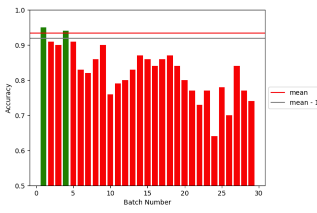
4.2 Real-World Data Sets

Finding the Reference Drifted Batches

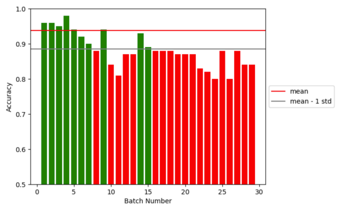
Regarding the real-world data sets, the assumption is that we do not know where drift is beforehand, so in order to perform the evaluation of our detectors we need to find a way to find the batches with drift in order to use them as a benchmark for our experiments.

To do this, we first chose the best classifier for each data set and performed cross-validation on the training data. We used two approaches for splitting the data for the cross-validation: *time-based splitting (TB)* and a variation of random-splitting, which we can call *sequential-based splitting (SB)* [14]. The time-based splitting method considers split ratios ranging from 50%/50% to 90%/10% [14], while the sequential splitting randomly chooses a testing set the same size as the testing batch and the rest of the samples as training. The mean and standard deviation were calculated based on the results of 20 splits. After that, the goal is to compute the accuracy of the classifier on each of the testing batches - this time, we do need to use the labels in the training set. If the *accuracy of a batch* is lower than the *mean accuracy of the training data minus one standard deviation*, then we assume that the batch has drifted. The batches signaled as drifted will be used as references when evaluating our detectors. The code for finding the reference drift batches for the *spam* data set is publicly available at https://github.com/tzamfirescu/concept-drift-detection.

The *spam* data set is one of the most difficult to classify because of the huge number of features - more than 10k, so few classifiers returned good results, with the best being *Random Forest*. The two splitting methods performed differently on each data set: in the case of spam, the time-based splitting signaled drift in almost all batches, while the sequential method signaled only batches with significant drift, as shown in Fig. 6, where green bars represent non-drifted batches and red bars the batches with drift. In this case, if we chose the time-based splitting method as our reference, detectors that detect all batches as drifted but may not be accurate would get almost perfect accuracy. Since the two methods produce different results, our reference drifted batches in the following experiments are computed using both the sequential and time-based splitting methods. Furthermore, in the case of *Airlines*, which contains categorical features, these experiments will include results computed using different encoders, namely *ordinal*, *one-hot*, and *target*. To compute the reference batches for the different real-world data sets, we will choose the classifier with the best accuracy for each - for *Airlines* and *ELECT2*, it is *K-Nearest Neighbors*, for *Weather*, it is *Support Vector Machines*, and for *spam Random Forest*.



(a) Time-based split method



(b) Sequential-based split method

Figure 6: Drifted batches in the *spam* data set, using the two split methods

Performance of Detectors on Real-World Data

After computing the reference drifted batches, we can evaluate our detectors using the metrics defined earlier, false positive rate FPR_R and drift detection rate DDR. The results are shown in Table 3. The two detectors, SQSI [15] and UDetect [4], managed to find the right batches at most times, as shown by DDR being close to 1, but they also had a very high FPR_R - they detected drift in batches that are not supposed to be drifted. Some experiments signaled all batches as drifted ($FPR_R = 1$ and $DDR = 1$), with *Airlines* being the only data set for which no drift was detected when using the *target* encoder on the categorical features. *Weather* is the only data set for which both detectors detected concept drift and also found non-drifted batches.

Even though both detectors performed well on the synthetic data, they performed poorly on real-world data, with

					SQSI		Udetect	
Dataset	**Classifier**	**Encoder**	**Size Batch**	**Split Method**	**FPR**	**DDR**	**FPR**	**DDR**
spam	Random Forest	No	20	TB	0.35	0.22	1	1
				SB	0.41	0.14	1	1
			50	TB	0.75	0.59	1	1
				SB	0.63	0.58	1	1
			100	TB	0.66	0.84	1	1
				SB	0.66	0.9	1	1
weather	SVM	No	30	TB	0.34	0.54	0.93	0.97
				SB	0.34	0.53	0.94	0.94
			365	TB	1	1	0.8	0.78
				SB	1	1	1	0.77
ELECT2	KNN	No	365	TB	1	1	0.85	0.85
				SB	1	1	0.92	0.81
Airlines	KNN	OneHot	17000	SB	1	1	0.33	0.05
		Ordinal			1	1	0.5	0
		Target			1	1	ND	
		OneHot	17000	TB	1	1	0.5	0
		Ordinal			1	1	0.5	0
		Target			1	1	0.5	0

Acronyms: SB - sequential-based split, TB - time-based split

Table 3: Results of different label-dependent concept drift detectors, applied on real-world data sets

an average false positive rate of 0.81 and a drift detection rate of 0.72. The main reason for this would be the way the reference drift is computed - it is difficult to find a generalized way of defining *what is drift?* which works for all types of data sets, concept drift detectors, batch sizes, etc. It is important to note that both detectors focus on retraining the classifier as soon as drift is detected, which is not done in this study. The fact that most of the batches in our experiments signal drift is expected since each batch is compared to the training data, without considering previous batches in the testing data.

4.3 Label-Dependent Concept Drift Detectors

The label-dependent concept drift detectors were only evaluated on real-world data based on the same *reference drifted batches* we computed earlier. All detectors mentioned in subsection 3.4 are error-rate based, meaning that the input of the detector relies on a classifier that should be trained on the reference data - we used *Random Forest* for all experiments. As soon as the model is trained, the detector needs to be updated with values corresponding to the samples in the testing set - 0 for a correct prediction and 1 for a wrong prediction of the classifier. The detectors are constantly updated with values over the whole testing set, meaning that previous testing batches will also be considered when checking if a batch is drifted. In comparison, SQSI and UDetect consider each testing batch independently and compare it to the training data - for example, let's consider a data set with *recurrent drift*, which starts at batch 3 and returns to normal at batch 6. Label-independent detectors (SQSI and UDetect) are expected to signal batches 3, 4, and 5 as drifted, while label-dependent detectors signal only subsequent changes in the distribution, meaning they will signal only batches 3 and 6 as drifted. The different way concept drift is computed influences the results - label-dependent drift detectors predict fewer batches with drift, so both DDR and FPR are lower. A better approach for evaluating the label-dependent detectors would be to compute the *reference drifted batches* while retraining the model on the testing data. Our setup, which simulates a machine learning production environment, does not do this.

				EDDM		DDM		ADWIN		HDDM_A		HDDM_W	
Dataset	**Features Encoder**	**Size Batch**	**Split Method**	**FPR**	**DDR**	**FPR**	**DDR**	**FPR**	**DDR**	**FPR**	**DDR**	**FPR**	**DDR**
spam	No	20	TB	0	0.05	0	0.03	0	0.01	0	0.01	0	0.01
			SB	0	0.07	0	0.03	0	0.01	0	0.01	0	0.02
		50	TB	0	0.13	0	0.07	ND		ND		ND	
			SB	0	0.19	0.04	0.08						
		100	TB	0	0.5	0	0.1	0	0.03	0	0.04	0	0.04
			SB	0.1	0.6	0.1	0.1	0	0.05	0	0.03	0	0.03
weather	No	30	TB	0.11	0.07	0.006	0.008	ND		ND		0.01	0.02
			SB	0.06	0.1	0.007	0.007					0.007	0.02
		365	TB	0.1	0.04	ND		0	0.04	0	0.04	0.1	0.26
			SB	0.5	0.03			0	0.03	0	0.03	0	0.22
ELECT2	No	365	TB	0.92	0.97	0.85	0.64	0.57	0.55	0.85	0.85	0.86	0.77
			SB	1	0.95	0.8	0.66	0.5	0.56	1	0.83	1	0.86
Airlines	OneHot	17000	SB	0	0.33	0.33	0.44	1	1	0.66	0.72	1	1
	Ordinal			0.5	0	0.5	0.42	1	1	1	0.89	1	1
	Target			ND		0.15	1	0.05	1	0.1	0	0.75	1

Acronyms: SB - sequential-based split, TB - time-based split

Table 4: Results of different label-dependent concept drift detectors, applied on real-world data sets

Despite the different ways the label-dependent detectors define drift, they signaled concept drift in most data sets using our framework. ADWIN and HDDM did not detect drift in *spam* and *Weather* for small batch sizes, while DDM failed to detect for *Weather* when the batch size is 365. It is essential to notice that the difference in results is significant, based on which detector is used: for instance, in the case of *spam*, EDDM had $FPR_R = 0.1$ and $DDR = 0.6$, but all other detectors had FPR_R and DDR close to 0. On average, the detectors tend to predict close to no drift in *Weather* and *spam* but predict almost all batches as drifted in *Airlines* and *ELECT2*. Similarly to the label-independent concept drift detectors, results prove that label-dependent detectors are also data set dependent.

5 Discussion

The two mixed concept drift detectors presented in [15] and [4] were difficult to implement and adapt to our data setup, mostly because a lot of important details were missing - a lot of time was spent into finding the right parameters and methods. This emphasizes again the importance of publicly available code in this context, which would save researchers time and resources and be able to focus on other aspects of concept drift detection.

Both detectors performed very well on synthetic data, with an average false positive rate of 0.051 and latency of 0.085. Based on the results, SQSI and UDetect are data set dependent and the features rely on scalers and encoders. These aspects were not mentioned in the papers, making the replication of the algorithms more difficult. The scope of the evaluation on synthetic data sets was to observe the behavior of the detectors and check if the assessment is worth doing on real-world data. The drawback of the synthetic data sets is that drift starts at some point, and the assumption is that all subsequent batches should be drifted. In other words, we cannot determine if the detectors accurately signal recurrent drift (when the distribution changes at some point and comes back to normal at a later point), which frequently happens in real-world data sets - we can only reliably signal when the **first drift** occurs.

The evaluation of detectors on real-world data is a challenge using this setup because it is difficult to find an algorithm that can be reliably used to answer the question *'what is drift?'*. Our experiments required us to compute the *reference drifted batches* for each real-world data set, so we defined a framework for detecting drift based on the increase in the error rate. The method behaved differently based on multiple factors: aside from the data set types, batch sizes, features encoders, or scalers, the results also relied on the type of split used for the cross-validation, time-based or sequential-based. Furthermore, each researcher evaluates their detectors by defining specific metrics and drawing multiple assumptions, which no general framework for drift detection can cover. Based on the reference drifted batches we defined in Section 4.2, the two label-independent algorithms performed modestly on real-world data, with a high average drift detection rate DDR of 0.72, but an even higher FPR_R of 0.81. One of the main causes for inconsistent results is that in their original implementations, both detectors retrain the model as soon as drift is detected, which is not done in this study.

The evaluation of label-dependent concept drift detectors requires a different data setup than the one in Section 3. The five error-rate-based detectors DDM, EDDM, ADWIN, HDDM_A, HDDM_W should be updated at every data point in the testing data, meaning that drift solely relies on the distribution of the testing set. In contrast, the framework we defined for the label-independent detectors calculates if a batch is drifted by only considering the training data.

More computational power would improve the results of this study. The code in the GitHub repository only allows CPU computation. What would help would be the code to have support for parallelization on multiple GPU nodes of the TU Delft's DAS-6 cluster. It would allow SQSI to be evaluated using *support vector machines*, improve the accuracy of all classifiers by using more computationally heavy setups and make it possible to increase the number of experiments. Many experiments are impossible to run accurately, and in a feasible time on a local machine, so more computational power would improve the overall results.

6 Responsible Research

This section aims to address the ethical standards of this project, focusing on the reproducibility of the methods and results. This is especially important in a comparative study; easier replication allows the researcher to evaluate multiple detectors and produce more results.

During this study, we learned about the ineffective way the scientific community deals with data - the implementations of algorithms and data sets are rarely available, making the replication process in this field of research unnecessarily long. Considering this, everything involving this study - the data setup, implementation of the algorithms, the method for finding the reference drifted batches, and a guide for connecting to the DAS-6 GPU nodes using SLURM, are all publicly available on GitHub. TU Delft platforms have limited accessibility for external users, so GitHub was chosen for sharing the code with the scientific community. Open data access in scientific research would save valuable time and allow researchers to focus on other new problems.

The methodology required for the experiments was clearly described in Section 3, namely the data setup, data sets, concept drift detectors, and evaluation methods. The paper also presents the pseudo-code of the two label-independent detectors with all the required formulas and parameters, aiming to simplify the replication process. The results for all experiments are available in Section 4 - even poor or inconsistent results were published and analyzed. This study aims to find a general evaluation framework for concept drift detectors; the poor performance of some detectors does not necessarily mean that they are unusable for drift detection but rather that they are not suited for our evaluation framework.

7 Conclusion

This paper proposed a comparative study in which the goal was to evaluate concept drift detectors in a setup that simulates deployed machine learning models. Two mixed label-independent concept drift detectors, SQSI and UDetect, are implemented and evaluated with different configurations on three synthetic and four real-world data sets. Not having access to the implementation made the replication process more demanding, emphasizing the importance of publicly available code in machine learning research. As soon as the two detectors are implemented, the first part of **RQ1** can be answered: in the case of synthetic data, both algorithms can reliably detect the right drifted batch, with FPR_S and L lower than 0.2. To evaluate the detectors on real-world data sets, we defined a framework for finding the *reference drifted batches* using cross-validation and two splitting methods. Based on the results, we can conclude that neither the reference drift framework nor the two detectors are reliable on real-world data sets under the setup we defined in 3.1. To answer **RQ2**, label-dependent detectors detect concept drift differently than our data setup does - our setup compares each batch to the training data, while label-dependent detectors are updated at each testing data point. Because of this, this setup does not allow for a direct comparison of the two types of drift detectors.

This study can improve its results by increasing the computational power. In this case, speed was chosen over accuracy, so adapting the code for parallelization on GPU nodes would allow much more accurate results. Also, the full implementation of the label-independent concept drift detectors, including the retraining of the classifiers as soon as drift is detected, would allow for more flexibility; for instance, label-dependent detectors could be directly compared to label-independent detectors. Future research might involve finding a generalized way of computing the reference concept drift that can be used to evaluate multiple types of concept drift detectors.

Acknowledgements

This research paper was part of the *CSE3000 Research Project* course in TU Delft's BSc Computer Science and Engineering. The project proposes a comparison study that analyzes the performances of different unsupervised concept drift detectors in a setup suited for deployed machine learning models. The responsible professor is **Jan Rellermeyer** (TU Delft, Leibniz University Hannover), and the supervisor is **Lorena Poenaru-Olaru** (TU Delft). All meetings were with our supervisor and the peer group containing three other students - **Jindřich Pohl**, **Konsta Kanniainen**, and **Baptiste André**.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, 1993.

[2] Manuel Baena-García, José Campo-Ávila, Raúl Fidalgo-Merino, Albert Bifet, Ricard Gavald, and Rafael Morales-Bueno. Early drift detection method. *Fourth International Workshop on Knowledge Discovery from Data Streams*, 01 2006.

[3] Henri Bal, Dick Epema, Cees de Laat, Rob van Nieuwpoort, John Romein, Frank Seinstra, Cees Snoek, and Harry Wijshoff. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(5):54–63, 2016.

[4] Sulaimon Adebayo Bashir, Andrei Petrovski, and Daniel Doolan. A framework for unsupervised change detection in activity recognition. *International Journal of Pervasive Computing and Communications*, 13(2):157–175, 2017.

[5] Albert Bifet, Ricard Gavaldà, Geoff Holmes, and Bernhard Pfahringer. *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press, 2018. https://moa.cms.waikato.ac.nz/book/.

[6] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. *Proceedings of the 7th SIAM International Conference on Data Mining*, 7, 04 2007.

[7] Gregory Ditzler and Robi Polikar. Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2283–2301, 2013.

[8] Christopher Duckworth, Francis P. Chmiel, Dan K. Burns, Zlatko D. Zlatev, Neil M. White, Thomas W. V. Daniels, Michael Kiuber, and Michael J. Boniface. Emergency department admissions during covid-19: explainable machine learning to characterise data drift and detect emergent health risks. *medRxiv*, 2021.

[9] Isvani Frías-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jiménez, Rafael Morales-Bueno, Agustín Ortiz-Díaz, and Yailé Caballero-Mota. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823, 2015.

[10] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In Ana L. C. Bazzan and Sofiane Labidi, editors, *Advances in Artificial Intelligence – SBIA 2004*, pages 286–295, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

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

[12] Bartosz Krawczyk and Alberto Cano. Online ensemble learning with abstaining classifiers for drifting and noisy data streams. *Applied Soft Computing*, 68:677–692, 07 2018.

[13] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2019.

[14] Yingzhe Lyu, Heng Li, Mohammed Sayagh, Zhen Ming (Jack) Jiang, and Ahmed E. Hassan. An empirical study of the impact of data splitting decisions on the performance of aiops solutions. *ACM Trans. Softw. Eng. Methodol.*, 30(4), jul 2021.

[15] Andre G. Maletzke, Denis M. dos Reis, and Gustavo E.A.P.A. Batista. Quantification in data streams: Initial results. *2017 Brazilian Conference on Intelligent Systems (BRACIS)*, 2017.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[17] Lorena Poenaru-Olaru, Luis Cruz, Arie van Deursen, and Jan S. Rellermeyer. Are concept drift detectors reliable alarming systems? – a comparative study, 2022.

[18] Jambulingam Subramani and Saminathan Balamurali. Control charts for variables with specified process capability indices. *International Journal of Probability and Statistics*, 1(4): 101-110, 01 2012.

[19] RAPIDS Development Team. *RAPIDS: Collection of Libraries for End to End GPU Data Science*, 2018.

[20] Alexey Tsymbal. The problem of concept drift: definitions and related work. Technical Report TCD-CS-2004-15, The University of Dublin, Trinity College, Department of Computer Science, Dublin, Ireland, 2004.

[21] Heng Wang and Zubin Abraham. Concept drift detection for streaming data, 2015.

[22] D.J. Wheeler. *Understanding Variation: The Key to Managing Chaos*. SPC Press, 1993.

[23] Shujian Yu, Zubin Abraham, Heng Wang, Mohak Shah, Yantao Wei, and José C. Príncipe. Concept drift detection and adaptation with hierarchical hypothesis testing, 2017.