OST SM MEMA change detection

Marcell Szabo, Ruixuan Mao, Emma Maria Sole Tosato, Zariqi Albert

December 2023

1 Introduction

1.1 Purpose and Dataset

Change detection in data streams is crucial for analyzing Cyber-Physical Systems (CPS), drawing significant attention due to its real-world implications. This process, essential in dynamic environments, involves identifying data variations over time, often leading to 'concept drift' which can greatly impact predictive models. For instance, in industrial control systems, detecting concept drift timely can avert operational failures or security breaches. Our study uses different versions of the HAI dataset,the HAI dataset is a collection of time-series data from an industrial control system simulation involving processes like heating, turbine operation, and water treatment, labeled for cybersecurity analysis. It includes data points and attack indicators, designed to study concept drift and anomalies in cyber-physical systems[1] [2], some members worked with 1.0 version and some with the 21.03. These versions are suitable to demonstrate practical change detection applications in system, providing an ideal platform to assess various algorithms.

In addition, for a deeper understanding of concept drift and its implications for project, we refered to "Learning under Concept Drift: A Review" (Lu et al., 2019)[3], which provides an overview of the various detection methods and challenges, which provides important theoretical support for this project.

The change management process is critical in data flow mining. It includes the detection of changes in the data, subsequent model tuning, management, and evaluation of the impact of the changes. The causes of these changes can be attributed to new learning environments, the presence of unknown variables, or observed changes in data attributes. The dynamics of change can be categorised according to the way they occur: abrupt, gradual, incremental or iterative. We need the adaptability of machine learning models to be tested by their ability to detect and adjust to these changes to ensure continued performance and relevance.In industrial manufacturing, change detection techniques are crucial for predictive maintenance, where they monitor equipment sensor data to identify potential failures before they occur, reducing downtime and maintenance costs. Additionally, in process control, detecting shifts in operational data helps maintain quality standards by alerting to deviations that could indicate issues like equipment wear or process inefficiencies.[4]

1.2 Overview of the Project

Our methodology encompasses several steps:.

1. We transformed the HAI dataset into a stream format, integrating it into a data stream management system using Kafka and InfluxDB
2. After that have undertaken essential preprocessing steps like normalization and missing value handling using Spark and Python, followed by a detailed examination of the dataset and real-time statistical analyses such as histograms and sampling, further enhanced by real-time statistical representations in Grafana.
3. Then We then establish the foundational framework for the error rate-based change detection algorithm, incorporate the underlying machine learning algorithm, and set up a baseline implementation based on the algorithm mentioned in the previous section.
4. The final step evaluates these algorithms, concentrating on various performance metrics like detection delay, false detection rate, miss detection rate, and overall accuracy. Also the goal here was to understand how accuracy and detected changes were related.

1.3 Technologies

In executing this project, we utilized a range of technologies including Kafka, InfluxDB, Python, Spark, and Grafana.

1.4 Error rate-based drift detection

Our findings highlight a critical aspect of concept drift research: the necessity of addressing drift in machine learning contexts. The project underscores the importance of error rate-based drift detection algorithms, which track variations in the error rate of base classifiers and initiate an update process upon statistically significant changes.

These algorithms are broadly categorized into error rate-based and data distribution-based types. Error rate-based methods, like DDM (Drift Detection Method)[5], focus on model error rates to signal changes. Other notable methods include HDDM[6], EDDM[7], ADWIN[8],PageHinkley[9], and CUSUM[10], each offering unique drift detection approaches. These methods, combined with strategies for model retraining and adaptation, are critical in maintaining predictive model accuracy in evolving data streams.

It's important to distinguish between 'change' and 'drift', with 'drift' specifically referring to alterations affecting the target variable's statistical properties. This distinction is key for effective model adaptation strategies. Our comprehensive study of these change detection algorithms not only examines their individual merits but also explores broader implications in managing concept drift in system. Through this analysis, we contribute to a deeper understanding of maintaining robust, reliable predictive models in dynamic environments.

2 System Architecture and project workflows

In this section we are discussing the architecture and infrastructure that we set up for the change detection project. On the below diagram the detailed architecture can be seen. Furthermore, the pipelines can also be observed. It is rather important, that this is a generic overview of our workflows. The a high level blueprint was designed together but the implementations differ by the members. For comprehensive information please refer to the documentation of individual GitHub branches of the group members.

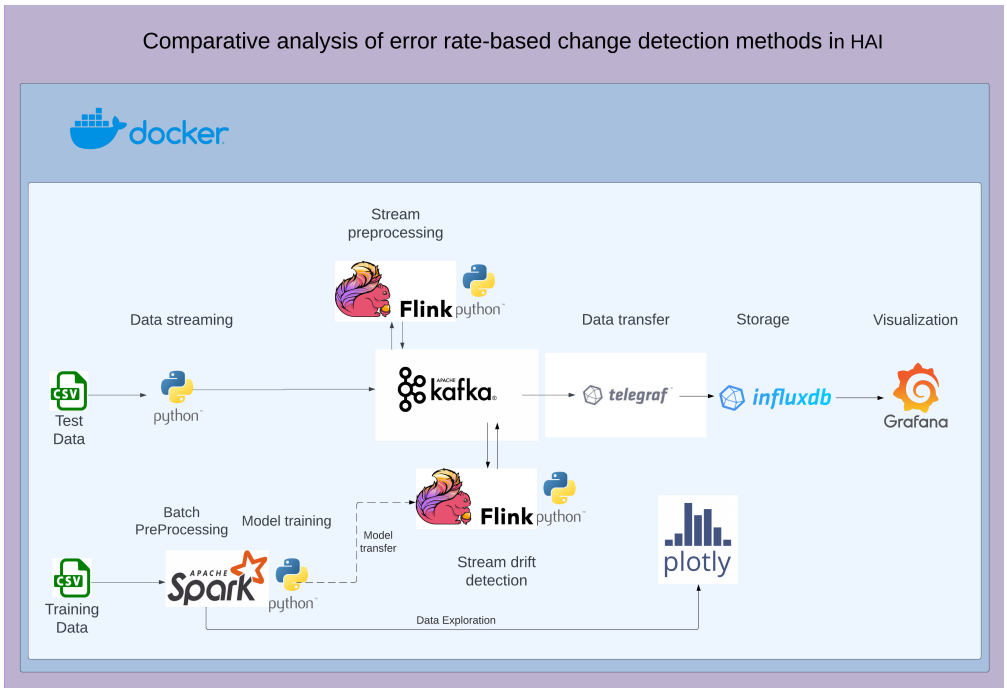

Figure 1: Architecture and pipelines

2.1 Utilized technologies

2.1.1 Docker

Docker is an open source platform that enables developers to build, deploy, and manage applications in isolated environments called containers. It provides a lightweight, efficient way to package applications and their dependencies into a single object.Docker's containerisation technology enables consistent operations across environments and is suitable for deploying our projects across different runtime environments.[11]

2.1.2 Python

Python is a high-level interpreted programming language known for its simplicity and ease of reading. python is widely used for web development, data analysis, artificial intelligence, and scientific computing, and supports a wide range of programming paradigms, with a large set of libraries and frameworks. the versatility and ease of use of the language has made it our programming language of choice.[12]

2.1.3 Kafka

Apache Kafka is a powerful open-source platform for distributed event streaming, essential for constructing real-time data pipelines and streaming applications. Its architecture, comprising Producers, Brokers, and Consumers, facilitates efficient data publishing, storage, and retrieval. In our project, Kafka's robust streaming capabilities align perfectly with the requirements of handling and processing large-scale data streams in real-time scenarios.[13]

2.1.4 Spark

Apache Spark is an open source unified analytics engine for large-scale data processing. Spark's main feature is in-memory computation, which provides high speed for iterative algorithms, making it particularly suitable for the big data processing and machine learning tasks in our project.[14]

2.1.5 Flink

Apache Flink is an open source stream processing framework for distributed, high-performance, always-available, and accurate data streaming applications. It excels at handling both unbounded and bounded datasets and runs in all common cluster environments. In addition, Flink's ability to provide accurate, real-time analytics makes it a key technology for the stream processing portion of our project.[15]

2.1.6 Telegraf

Telegraf is an open source server agent for collecting and reporting metrics and data. It supports a variety of plugins that allow it to collect metrics from a variety of sources, process them, and write them to a range of outputs.Telegraf is used for monitoring in DevOps practices, and its easy integration with time-series databases makes it easier to work with time-series data.[16]

2.1.7 Influxdb

InfluxDB is an open source time series database designed to handle high write and query loads. It is a NoSQL database optimised for fast, high-availability storage and retrieval of time-series data in areas such as operations monitoring, application metrics, IoT sensor data, and real-time analytics.InfluxDB's ability to efficiently store and process time-series data makes it our first choice for storing data.[17]

2.1.8 Grafana

Grafana is an open source analytics and interactive visualisation web application. When connected to supported data sources, it provides charts, graphs and alerts for the web, focusing primarily on time series data.Grafana is widely used for monitoring metrics and data visualisation, and allows us to create dashboards with a variety of visualisation options to monitor data in real time.[18]

2.1.9 Plotly

As it is stated in the introduction, the basis of error rate-based change detection usually is that a static data set is given which is used for data analysis and the training of one or more initial models. Then the live data is gathered and sent to servers. It is preprocessed, analyzed and then labels are predicted by the previously

trained model(s). If there is feedback available, the accuracy / error rate of the models can be calculated. The drift detectors analyze the error rate in a timely manner (real-time if possible) and if a longer and decent drop is observed in the error rate, the drift detector sends alerts and possibly triggers various adaptation mechanisms. In some cases multiple models are used depending on the state of the stream or the model is retrained.

In the following we explain our implementation of the above system. In the pipelines it is common that we integrated Docker to containerize the project and facilitate its execution.

2.2 Static pipeline

The first part is batch analysis. Data is loaded, preprocessed and analyzed using Spark and Python. Preprocessing involves data cleaning (e.g. dropping or filling Na values), features and label(s) separation, feature selection, scaling and normalizing along with numerous other actions. The data is further explored. We worked with the HAI training data. The trained model is saved so it can be used by the multiple drift detectors.

2.2.1 Data analysis on static data

In summary, the code performs data preprocessing, exploration, and visualization, providing valuable insights into the static dataset. It sets the foundation for further analysis and modeling in the context of anomaly detection in a cyber-physical system.

The use of Spark in data loading and preprocessing is critical for efficient handling of large datasets. Spark's batch processing capabilities enable parallel computation, making it well-suited for big data scenarios. This is especially important in the context of cybersecurity data, where large volumes of information need to be processed for anomaly detection.

The code also included Python libraries like Matplotlib and Plotly for for data visualization. The plots show some distributions of the features of the dataset in different time intervals and sets.

Data analysis steps:

- Data Loading and Insights: for understanding the data structure and characteristics, providing an initial overview of what the data looks like. Two dataset were used and explored, both composed of train, test and labels.
- Data Preprocessing: handling missing values, duplicates, and normalization. Some missing values were found but they were few. There no duplicated values.
- Data Visualization: plotting the distribution of variables and observe any patterns or anomalies. Subplots are used to display multiple variables in a single figure.

2.3 Streaming pipeline

- In our case the incoming stream is simulated by loading the test csv files and streaming them row-by-row to Kafka with a Python Kafka producer. The data format is changed during the process, i.e. the csv rows are sent as an encoded JSON, the keys representing the column names and the values the data values. Note that this way the data points are easily interpretable, but this comes with a slight overhead as the keys have to be transfered and stored for each data point. In real world screnarios more scalable representations may be considered. Kafka consumer input is included in Telegraf, and the streamed data is outputted to InfluxDB. This is not an indispensible part, since the raw data is not required (and often not possible) to be consisted for long. As the test part of our dataset e.g 21.03 version had around 400'000 points, saving it was feasible.
- The raw data is consumed and preprocessed the actions are similar to those of the static pipeline. Methods include filling NA values, separation of the features and labels and others. The preprocessed data is produced back to Kafka, this way the different drift detectors are able to use them and the preprocessing is only has do be done once. Marcell tried out the encapsulation of this step with Flink.
- At the core of the system sits the drift detector module. In the beginning it loads the trained model and the drift detector algorithm. Then it starts to consume the preprocessed data. Label(s) for each data point are predicted. Then the accuracy score / error rate is calculated and passed to the drift detector. The accuracies, warnings and change detection alerts are observed. In Marcell's code these metrics are produced to another Kafka topic and saved to InfluxDB through Telegraf. In Ruixuan's

and Albert's code the data is dirctly saved to influxdb. Each member created Python modules for this step. While Ruixuan and Albert experimented also with Spark orchestration, Marcell did that with Flink. Wrapping these stream processing steps in a Spark or Flink job is quite necessary for scalability.

- InfluxDB is added to Grafana as a data source. In Grafana, dashboards were created to visualize the drift detection results. The relationship between the attacks and accuracy is presented along with the drifts detected. Data features, distributions were plotted as well. Alerts were configured to notify users upon a change is detected.

3 Change Detection Algorithms

In this section introduce the algorithms that we studied and compared. As the main focus of the research is error rate based change detection, the basis of our change detection logic was to predict labels of the streamed data and compute the error rate or accuracy using the real labels, then detect changes based on the the error rate. Note that other measures of model performance would been also sufficient with the correct modifications, e.g. precision, AUC-ROC etc.

Most methods utilized have basic functionalities in common.

1. Initialization: parameters are passed (e.g. warning and change levels, window size (except ADWIN)) the variables, e.g. statistical counters, sums are declared.
2. Element addition: for each data point observed and predicted, a performance measure is computed and added to the method. The algorithm maintains the required statistical values. The input value is usually binary, either 1 or 0, where 1 means error.
3. Warning, change detection: After each added element the method checks the value of the maintained variables and triggers warning or change detection if the corresponding condition is met.
4. Reset: After change detected, the algorithm resets itself to its initial state.

3.1 Drift Detection Method (Marcell)

Drift Detection Method (DDM) is one of the older fundamental algorithms. It is designed to monitor and identify changes in the statistical properties of the streaming data, in our case the error rate, which is a derived measure of that. The method does not use windows, therefore it is not as responsive to smaller changes. When the detected increase in the error rate exceeds the warning or change threshold, warning or change will be detected.

The detection threshold is calculated as the minimum of $p_i and s_i$, where p_i is the ith recorded error rate, and s_i is the ith recorded standard deviation. The condition for entering the warining zone is $n = 2$ and for the change detection $n = 3$

$$p_i + s_i \geq p_{min} + n \cdot s_{min}$$

3.2 Hoeffding Drift Detection Method with exponentially Weighted moving average test (Marcell)

Note that there is another version HDDM-A using normal moving averages. HDDM-W is a more complex algorithm based on Hoeffding's bound. The Hoeffding Inequality for the sample mean $\bar{X}$ is given by:

$$\Pr\left(\left|\bar{X} - \mu\right| \geq \epsilon\right) \leq 2e^{-2n\epsilon^2}$$

Where Pr denotes the probability. $\bar{X}$ is the sample mean. μ is the true mean. ϵ is the deviation from the true mean. n is the number of samples.

This inequality provides an upper bound on the probability that the sample mean deviates from the true mean by more than a certain amount ϵ.

The algorithm maintains an exponential weighted moving average (EWMA) of the error rates passed in each step. The EWMA takes the newer values with more weight then the olders.

$$\text{EWMA}_t = \lambda \cdot x_t + (1 - \lambda) \cdot \text{EWMA}_{t-1}$$

The method checks if the error rate increase exceeds the corresponding bound and warns or detects change.

3.3 Basic moving average test (Marcell)

A very simple moving average based method was developed by Marcell. At initialization window size and thresholds have to be set. When adding elements, the algorithm maintains the average. If the moving average surpasses a certain value, warning and drift detection alert is fired. This is not a too flexible algorithm but it was suitable to help in the understanding the basics of change detection. However if well parametrized, this method can be as effective as others. The moving average is reacalculated as

$$\mathrm{SMA}_t = \mathrm{SMA}_{t-1} + \frac{1}{N} \cdot (x_t - x_{t-N}).$$

3.4 Early Drift Detection Method (Mao)

The Early Drift Detection Method (EDDM) is an algorithm designed for detecting concept drift in data streams. EDDM monitors the performance of a learning model, particularly focusing on the distances between errors in prediction.[7] It is particularly adept at recognizing gradual drifts, where changes occur slowly over time.[7]

The core formula of EDDM is defined as follows:

$$P_{max} = \max_{i=1}^{n} p_i, \quad S_{max} = \max_{i=1}^{n} s_i \tag{1}$$

where P_{max} and S_{max} represent the maximum distance and the maximum standard deviation between two errors, respectively, over n observations.

3.5 ADaptive WINdowing (Mao)

The ADaptive WINdowing (ADWIN) algorithm is a pivotal tool for change detection in data streams. ADWIN is unique in its ability to automatically adjust the size of its sliding window, making it adept at detecting both abrupt and gradual changes in data.[8] ADWIN works by maintaining a variable-sized window of recent data. If it detects a significant change in the data's distribution within this window, it will adapt the window size accordingly. This adaptability makes it particularly effective in environments with evolving data characteristics.[19] The core principle of ADWIN can be described as:

$$|\bar{X}_1 - \bar{X}_2| > \epsilon \tag{2}$$

3.6 PageHinkley (Albert)

The PageHinkley methodis a widely recognized algorithm for change detection in data streams, particularly effective in detecting shifts in the mean value of a sequential process. It is especially useful in scenarios where the change manifests as a gradual but consistent deviation over time.[9] The PageHinkley method operates by calculating a cumulative sum of the differences between the observed values and their mean. When this cumulative sum exceeds a predefined threshold, it signals a potential change.[20]The formula for the PageHinkley test is as follows:

$$PH_t = \max_{0 \le k \le t} \left(\sum_{i=k}^{t} (x_i - \mu - \delta) \right) \tag{3}$$

where x_i is the observed value at time i, μ is the estimated mean, δ is a sensitivity threshold, and PH_t is the PageHinkley statistic at time t.

3.7 Kolmogorov-Smirnov Windowing (Albert)

The Kolmogorov-Smirnov Windowing (KSwin) algorithm is an influential method in change detection, particularly in data streams within system. KSwin stands out for its use of the Kolmogorov-Smirnov statistic, a non-parametric test, to compare data distributions within a window, effectively identifying shifts in data characteristics.[21] KSwin operates by examining data within a defined window, using the Kolmogorov-Smirnov test to compare the distribution of data in the current window against a reference distribution. If the test indicates a significant difference, it suggests a potential change in the data stream.

The KSwin algorithm utilizes the Kolmogorov-Smirnov statistic to compare data distributions within a window. [22]The key mathematical representation of this statistic is as follows:

$$D_{KS} = \sup_x |F_n(x) - F(x)| \tag{4}$$

4 Results

Each member's experiments are presented and discussed. At the end an overall discussion can be found. Diagrams are shown presenting the error rates of the classifier and the drift detection points.

4.1 Experiment setup

A random forest classifier is used as a model for predicting if attacks happened in a given time point. HAI 21.03 and HAI 1.0 version was used for this experiment. It is worth mentioning that as labels not just the attack but the three types of attacks were also used enabling fine grained categorization. The dataset contains training and test data. In the training set there were no attacks causing the model to be biased and predict only no attacks. Therefore, when attacks occured in the test data, the model had high error rates, allowing the drift detector to find the drift. This is not a gradual drift but an abrupt change, like reoccuring concepts, which is easier to detect.

Since the test dataset contains more than 400'000 data points plotting them all is not quite informative. Hence only one full graph is showed. The error rate is a time series containing binary values, whether the classifier predicted wrong labels or the correct labels.

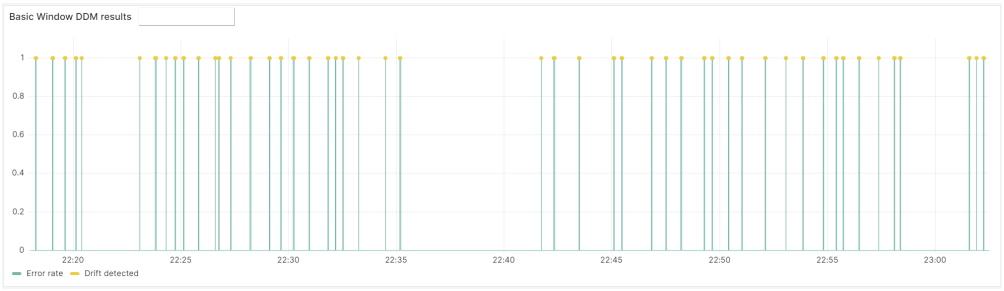

Figure 2: The full graph showing the drift detection process

4.2 Change detection experiment (Marcell)

4.2.1 DDM

The DDM proved to be the least performant. It struggled to detect very short changes. This may be also interpreted as more robust, which is positive. Because of its simplicity and efficiency, it completed around 20% faster than the other methods.

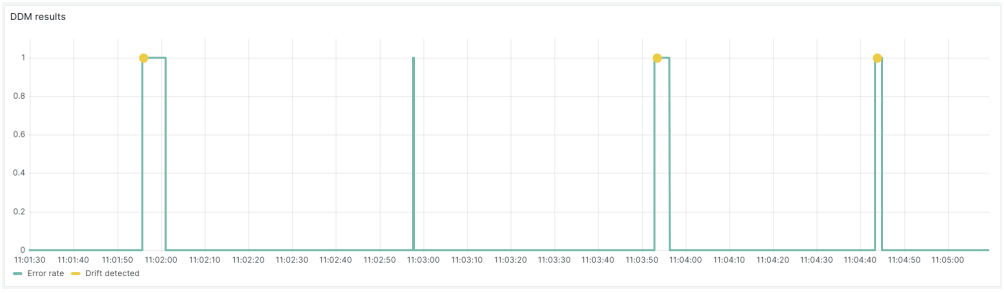

Figure 3: A zoomed in graph with the DDM results

4.2.2 HDDM-W

The HDDM-W is a very potent algorithm. It did not miss drifts, and it reacted quite rapidly. This on the other hand may cause false alarms as well.

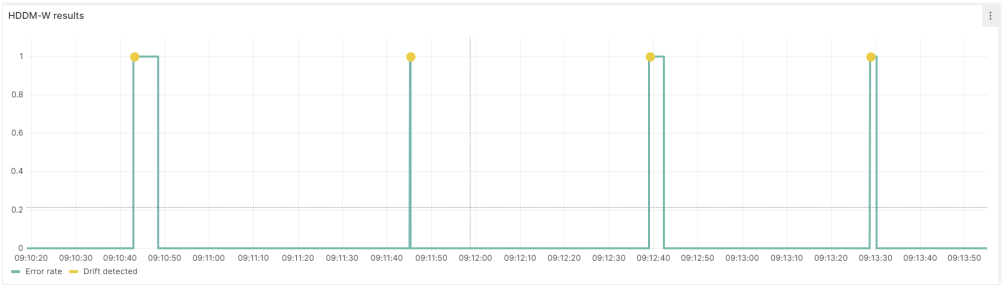

Figure 4: A zoomed in graph with the HDDM-W results

4.2.3 Basic Moving Average Test

This algorithm performed just as good as the more complex algorithms despite its simplicity. A drawback worth mentioning is that it needs to be parametrized precisely to achieve suitable results. There is no graph shown because it is very similar to the HDDM-W.

4.2.4 Change detection with adaptation and model management

Change detection is often combined with adaptation methods and model management. In this experiment 2 models are trained, one on the train set and the other is on data containing attacks. The first model provides good predictions in normal zones and the second performs well during attacks. The strategy is to use the first model and when the accuracy drops, switch to the other model. This logic enables to have high accuracies in normal conditions as well as during attacks, except in the change phase, until the change is detected.

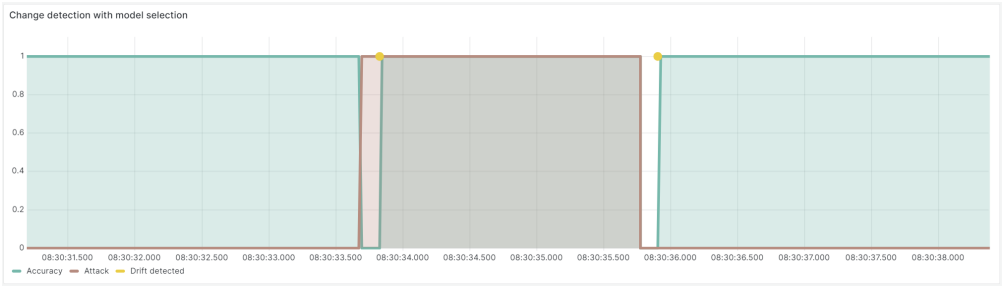

Figure 5: A zoomed in graph of the multi model setup

4.3 Change detection experiment (Mao)

4.3.1 EDDM

The EDDM algorithm shows improved performance in detecting gradual changes compared to DDM. It shows higher sensitivity to longer, more subtle changes in the data, which could explain its finer level of detection. This property is very beneficial in situations where gradual changes need to be detected as early as possible. Despite its enhanced detection capabilities, EDDM maintains a balance between sensitivity and computational efficiency, completing tasks approximately 15% faster than more sophisticated methods. This efficiency, coupled with its effectiveness in identifying gradients, highlights the utility of EDDM in situations where both timely and accurate change detection is required.

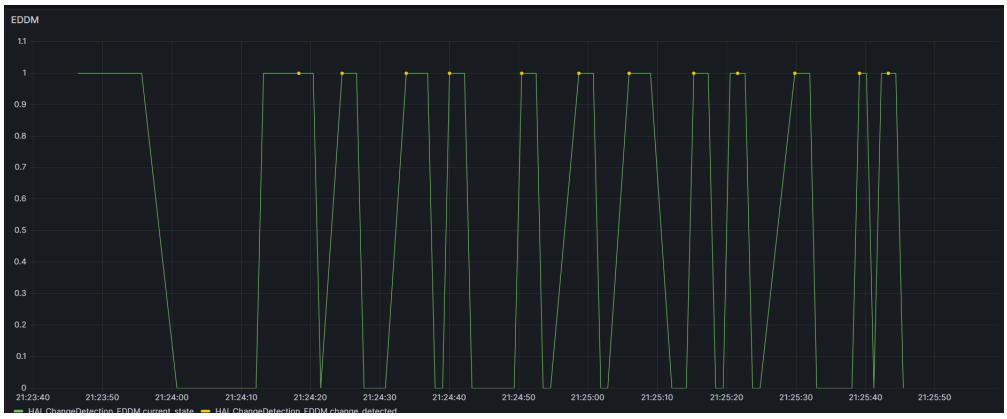

Figure 6: A zoomed-in graph with the EDDM results

4.3.2 ADWIN

The ADWIN algorithm shows remarkable adaptability in dealing with different rates of change. It performs very well in environments with frequent dynamic changes and outperforms the eddm algorithm in situations where data changes rapidly and unpredictably. However, this adaptability comes at the cost of computational efficiency, resulting in ADWIN being approximately 25% slower than the more eddm algorithm. Nonetheless, ADWIN is able to adapt instantly to changing data patterns because in some cases, detecting and responding to instantaneous changes is more important than processing speed.

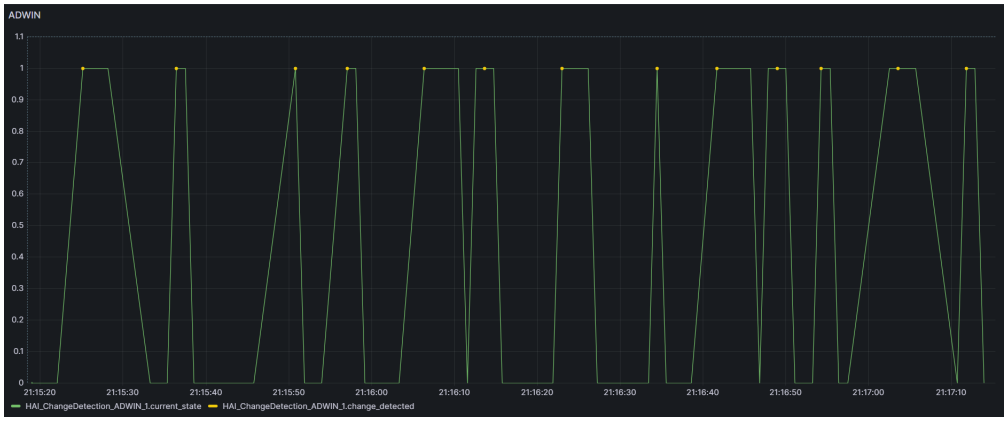

Figure 7: A zoomed-in graph with the ADWIN results

4.4 Change detection experiment (Albert)

4.4.1 PageHinkley

In our practical application, the Page-Hinkley method consists of continuously monitoring the sum of the cumulative differences between an observation and its mean. If there is a significant deviation from the mean, this indicates that drift has occurred. In our experiments, the Page-Hinkley method was implemented with the aim of improving response speed and efficiency. We set a detection threshold that balances sensitivity with avoiding false alarms. Once this threshold is crossed, the test shows drift, prompting us to check and adjust the model accordingly. As per results, we can see that in our experiments, this method has succeeded to detect the changes but sometimes with a small delay, which makes it a good algorithm to use for change detection.

Figure 8: A zoomed in graph with the PageHinkley results

4.4.2 KSWin

In our experiments utilizing the HAI dataset, we effectively implemented the KSWin algorithm, which functions by maintaining two distinct data windows. The first window captures the latest data points, while the second holds data from a period deemed stable. We then applied the Kolmogorov-Smirnov test to these windows to ascertain any significant divergence in their distributions. A change is flagged by KSWin when the disparity in distributions surpasses a set threshold, hinting at potential drift. This process is crucial, particularly in the HAI dataset, where timely and accurate identification of changes in industrial control systems' operational data is imperative. From the obtained results, we observe that KSWin demonstrates the capability to detect drifts with minimal delays. This characteristic positions it as a superior approach for change detection.

Figure 9: A zoomed in graph with the KSWin results

4.5 Overall discussion

In this chapter we compared various change detection methods. We found that there is no clear order between the goodness of the algorithms. Each has its advantages and disadvantages. The quickest models are EDDM and HDDM-W. DDM is more robust to noise. Simpler algorithms like DDM are less computationally expensive. ADWIN has the useful feature that it is not required to be parametrized as the other methods enabling more flexibility and robustness. The PageHinkley method effectively identifies gradual, long-term changes in data, making it ideal for subtle drift detection. However, its response to sudden changes is slower compared to other methods.

In contrast, KSwin excels in detecting significant distribution shifts within data windows, thanks to the Kolmogorov-Smirnov test. While highly effective in dynamic scenarios, KSwin demands precise calibration of window sizes and thresholds, making its setup more complex

References

[1] GitHub. Hil-based augmented ics (hai) security dataset. https://github.com/icsdataset/hai, 2023. Accessed: date-of-access.

[2] Hyeok-Ki Shin, Woomyo Lee, Jeong-Han Yun, and HyoungChun Kim. HAI 1.0: HIL-based augmented ICS security dataset. In *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*. USENIX Association, August 2020. Accessed: date-of-access.

[3] 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.

[4] E. Brondízio D. Lu Corresponding author, P. Mausel and E. Moran. Change detection techniques. *International Journal of Remote Sensing*, 25(12):2365–2401, 2004.

[5] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. pages 286–295, 2004.

[6] 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.

[7] Manuel Baena-García, José Campo-Ávila, Raúl Fidalgo-Merino, Albert Bifet, Ricard Gavald, and Rafael Morales-Bueno. Early drift detection method. 01 2006.

[8] 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.

[9] Hayet Mouss, M.Djamel Mouss, Kinza Mouss, and Sefouhi Linda. Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. pages 815 – 818 Vol.2, 08 2004.

[10] Thomas Flynn and Thomas Flynn. Change detection with the kernel cumulative sum algorithm. 12 2019.

[11] Docker: Empowering app development for developers. https://www.docker.com/. Accessed: date-of-access.

[12] Python programming language. https://www.python.org/. Accessed: date-of-access.

[13] Apache Kafka. Apache kafka. https://kafka.apache.org/, 2023. 2023-12-09.

[14] Apache spark - unified analytics engine for big data. https://spark.apache.org/. Accessed: date-of-access.

[15] Apache flink: Stateful computations over data streams. https://flink.apache.org/. Accessed: date-of-access.

[16] Telegraf: The plugin-driven server agent for collecting and reporting metrics. https://www.influxdata.com/time-series-platform/telegraf/. Accessed: date-of-access.

[17] Anurag Nair. Introduction to influxdb: A time-series database. https://wearecommunity.io/communities/india-java-user-group/articles/891, 2021. 2023-12-09.

[18] Munikanth. Grafana. https://medium.com/@munikanthtech/grafana-d2652ad884e7, 2023. 2023-12-10.

[19] Hayder Fatlawi and Attila Kiss. An adaptive classification model for predicting epileptic seizures using cloud computing service architecture. (appl. sci. 2022, 12, 3408. pp. 1-22). *Applied Sciences*, 12:1–22, 03 2022.

[20] Raquel Sebastião and Jose Maria Fernandes. Supporting the page-hinkley test with empirical mode decomposition for change detection. pages 492–498, 06 2017.

[21] Wikipedia contributors. Kolmogorov–smirnov test — Wikipedia, the free encyclopedia, 2023. [Online; accessed 18-December-2023].

[22] Maurras Togbe, Yousra Chabchoub, Aliou Boly, Mariam Barry, Raja Chiky, and Maroua Bahri. Anomalies detection using isolation in concept-drifting data streams. *Computers*, 10:13, 01 2021.

[23] Guang Li, Jie Wang, Jing Liang, and Caitong Yue. The application of a double cusum algorithm in industrial data stream anomaly detection. *Symmetry*, 10(7), 2018.