Master Computer Science

Concept Drift Detection Using the Distance to the Decision Boundary

Name: Rodi Laanen
Student ID: 2117819

Date: [27/06/2024]

Specialisation: Artificial Intelligence

1st supervisor: Annelot Bosman
2nd supervisor: Dr. Jan N. van Rijn
3rd supervisor: Dr. Bruno M. Veloso
4th supervisors: Prof.dr. Holger H. Hoos
Prof.dr. João Gama

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Einsteinweg 55
2333 CA Leiden
The Netherlands

Abstract

The growing number of sources generating vast amounts of data streams requires additional capabilities of machine learning models to process and adapt to this information. Factors such as dynamic environments, seasonality trends and changing human behaviour cause shifts in the relationship between instances and labels. This phenomenon is known as concept drift and can lead to a decrease in model performance. Traditional methods for detecting concept drift typically rely on the error rate. A disadvantage of such a drift detection method is that while concept drift is detected, the model's performance has already declined. Therefore, we propose to detect concept drift through $\tilde{\varepsilon}^*$-drift values. $\tilde{\varepsilon}^*$-drift values provide additional information compared to the regular error rate by approximating the distance to the decision boundary of misclassified instances. We evaluated our proposed method on two synthetic datasets simulating abrupt and incremental drift. We assessed three input types: error rate, features and $\tilde{\varepsilon}^*$-drift with six drift detection methods: DDM, EDDM, ADWIN, KSWIN, Mann-Whitney U test and a threshold method. The conducted experiments indicate that, while $\tilde{\varepsilon}^*$-drift are not consistently faster in detecting concept drift and are more computationally expensive than the regular error rate, they are effective in detecting concept drift and exhibit a lower average number of false positives compared to the regular error rate. Moreover, this thesis serves as a first step to investigate further the relationship between the $\tilde{\varepsilon}^*$-drift values and model performance.

Acknowledgements

I would like to express my gratitude to my supervisors: Annelot Bosman, Jan van Rijn, Bruno Veloso, Holger Hoos and João Gama for their guidance, mentorship and support throughout this thesis. This thesis provided me with the opportunity to be part of a collaborative effort between Leiden University and the University of Porto. This collaboration has enriched my research experience and offered me new perspectives. A special thank you goes to Annelot for her daily guidance and the enjoyable coffee breaks.

Table of Contents

1 Introduction **1**

2 Background Theory **3**
- 2.1 Concept Drift . . . 3
  - 2.1.1 Types of Concept Drift . . . 5
  - 2.1.2 Concept Drift Detection Framework . . . 5
  - 2.1.3 Concept Drift Detection Methods . . . 7
- 2.2 Local Robustness Verification . . . 7
- 2.3 Key Ideas . . . 10

3 Related Work **11**
- 3.1 Concept Drift Detection Methods . . . 11
- 3.2 Detecting Concept Drift Through Critical ε Values . . . 12

4 Methodology **14**
- 4.1 Lower Bound to the Critical ε Value . . . 14
- 4.2 Lower Bound to the Critical ε-drift Value . . . 21
- 4.3 Concept Drift Detection Through $\tilde{\varepsilon}^*$-drift Values . . . 22
  - 4.3.1 Binary Search for $\tilde{\varepsilon}^*$-drift Values . . . 22
- 4.4 Concept Drift Detection Methods . . . 22

5 Experiments **24**
- 5.1 Synthetic Data Generators . . . 24
  - 5.1.1 SEA . . . 24
  - 5.1.2 Hyperplane . . . 24
- 5.2 Evaluation Methods . . . 25
- 5.3 Experimental Settings . . . 26
  - 5.3.1 Dataset Settings . . . 26
  - 5.3.2 Normalization of the Input Data . . . 26
  - 5.3.3 Neural Network Configurations . . . 27
  - 5.3.4 α,β-CROWN Settings . . . 27
  - 5.3.5 Drift Detector Settings . . . 28
- 5.4 Results . . . 28
  - 5.4.1 Synthethic Datasets Results . . . 28
  - 5.4.2 Running Times on Synthetic Datasets . . . 31
  - 5.4.3 Discussion . . . 32
  - 5.4.4 Limitations . . . 33

6 Conclusion and Future Work **34**

References **40**

Appendix A SEA Results **40**

Appendix B Dataset Configurations **42**

Appendix C Neural Network Configurations **44**

Appendix D α,β-CROWN YAML configuration file **45**

Appendix E Misclassifications **46**

1 Introduction

Nowadays, various sources, such as sensors [1], financial market [2, 3], weather stations [4], electricity market [5] and Internet-of-Things (IoT) [6] provide increasing flows of information [1, 7, 8, 9]. These vast amounts of data are known as data streams [10]. Characteristics of data streams include the continuous or potentially infinite flow of incoming data, the temporal aspect ensuring sequential order and the dynamic nature [10, 11, 12]. Unlike traditional static datasets, data streams evolve over time and demand additional capabilities of models to process them [8, 10].

Dynamic environments, seasonality trends and changing human behaviour among other factors might cause shifts in the underlying data distribution [7, 13]. This change in the underlying data distribution or hidden context is referred to as concept drift [14]. As a consequence of concept drift, models processing data streams are prone to a decrease in performance. To prevent a model from becoming obsolete, it is crucial to equip the model itself or an external mechanism with the capacity to detect concept drift and act on it accordingly.

Current methods to detect concept drift rely, among others, on the error rate of the model [11, 15, 16, 17], statistical tests performed on batches of the data stream or uncertainty estimates from a deep neural networks [18, 19, 1, 20]. For instance, the Drift Detection Method (DDM) [11] compares the continuously updated error rate of the model and considers a significant increase in the error rate as an indication of concept drift. A downside of this approach is that the performance of the model needs to decrease to trigger the warning and concept drift stages. Such a method acts retroactively meaning that the performance of the model has already deteriorated. Naturally, this is undesirable since one wants to maintain a high-performing model.

To mitigate a decrease in model performance, we aimed to detect concept drift via local robustness verification. Local robustness verification [21] provides a formal approach to examine to which extent a neural network is sensitive to instances with small perturbations causing incorrect predictions. The work of Szegedy et al. [22] made clear that these small changes to the input, also known as adversarial examples, trigger a neural network to predict a wrong class. For instance, subtle manipulations of images invisible to the human eye can make neural networks generate a wrong output. For this reason, it is important to formally define the robustness of a neural network against adversarial examples, especially in safety-critical domains [23, 24].

A common approach to measure the robustness of a neural network is through a single predefined ε value. ε denotes the size of allowed perturbation on an original instance. This means that a smaller ε value encompasses a tighter region in which a neural network verifier can search for an adversarial example. However, defining this single ε value requires domain knowledge and does not provide information on a per-instance level.

Therefore, Bosman et al. [25] introduced the ε^* value representing the maximum amount of perturbation per instance such that the neural network still makes a correct prediction. Given that this problem is NP-hard [26], Bosman et al. [25] discretised the search for the ε^* values introducing the $\tilde{\varepsilon}^*$ value. The $\tilde{\varepsilon}^*$ approximates the ε^* value such that the perturbed instance is still classified correctly. In essence, the $\tilde{\varepsilon}^*$ value translates to an approximation of the distance towards the decision boundary of a neural network for correctly classified instances.

Measuring the $\tilde{\varepsilon}^*$ values for a batch of a data stream provides us with a distribution of distances towards the decision boundary. In the presence of concept drift, more misclassifications will occur; hence, the distribution of correctly classified instances will change. We think that observing changes in such distributions provides an indicator of concept drift.

However, in preliminary experiments, we observed that using $\tilde{\varepsilon}^*$ values is ineffective in detecting concept drift in an earlier stage than regular error rate-based methods. A detailed explanation

is provided in the methodology section 4 of this thesis. Nevertheless, we did make interesting observations regarding the misclassified instances before and after concept drift occurred. With a well-trained classifier, incorrect predictions typically occur for instances close to the decision boundary. In the presence of concept drift, the region encompassing incorrect predictions expands, resulting in misclassified instances being located further from the decision boundary.

To measure this distance towards the decision boundary for misclassified instances, we introduced the $\tilde{\varepsilon}^*$-drift value based on the $\tilde{\varepsilon}^*$ from the work of Bosman et al. [25]. Without concept drift, we expect very small $\tilde{\varepsilon}^*$-drift values for misclassified instances. However, these values will increase in the presence of concept drift. Significantly larger $\tilde{\varepsilon}^*$-drift values can function as an indicator of concept drift.

To the best of our knowledge, a method leveraging an approximation of the distance to the decision boundary in the form of $\tilde{\varepsilon}^*$-drift values to detect concept drift does not yet exist. For this reason, we investigated the following research questions:

1. Is it possible to utilise $\tilde{\varepsilon}^*$-drift values for the identification of concept drift?
2. Do $\tilde{\varepsilon}^*$-drift values in combination with a drift detector allow for detecting concept drift in an earlier stage than regular error rate-based methods?
3. Do $\tilde{\varepsilon}^*$-drift values in combination with a drift detector result in a lower number of false positives compared to regular error rate-based methods?

The main contributions of this thesis are the following:

- We provided explanations on why the regular lower bound to the critical epsilon value (i.e., $\tilde{\varepsilon}^*$) is not working for the detection of concept drift.
- We defined the lower bound to the critical ε-drift value (i.e., $\tilde{\varepsilon}^*$-drift value) for misclassified instances. It represents a distance measure towards the decision boundary for misclassified instances.
- We combined the $\tilde{\varepsilon}^*$-drift value with existing drift detectors to detect concept drift.
- We introduced the $\tilde{\varepsilon}^*$-drift value in combination with the Mann-Whitney U test to detect concept drift.
- We introduced the $\tilde{\varepsilon}^*$-drift value in combination with a threshold method to detect concept drift. However, the threshold method demands further investigations into the relationship between $\tilde{\varepsilon}^*$-drift values and the performance of a model.

The remainder of this thesis is organised as follows: First, we introduce the foundational concepts and techniques related to data streams, concept drift and neural network verification in Section 2. Subsequently, in Section 3, we discuss common types of input to detect concept drift as well as drift detection methods. This section also contains detailed information about the $\tilde{\varepsilon}^*$ value. Next, we explore how $\tilde{\varepsilon}^*$ values can be used for detecting concept drift. Based on a simplification of the SEA dataset simulating sudden drift, we provide justifications for why this input type is not prominent for concept drift detection. Following this, we transition to utilising $\tilde{\varepsilon}^*$-drift for identifying concept drift. In Section 5, we demonstrate and analyse our conducted experiments. Finally, in Section 6, we interpret the obtained results and share our conclusions.

2 Background Theory

In this section, we introduce the topics of concept drift and local robustness verification. First, we will differentiate concept drift from other possible types of data drift. Afterwards, we will explain four types of concept drift and discuss a general framework for detecting concept drift. Moreover, we provide a brief overview of three categories of concept drift detection methods: error rate-based, data distribution-based and multiple hypothesis test.

Secondly, we will start by defining the notion of adversarial example which cause neural networks to generate wrong predictions. We then provide a mathematical description of the task of local robustness verification, focusing on both incomplete and complete verification. Additionally, we discuss the complete verification tool α,β-CROWN [27] which leverages branch and bound techniques to efficiently verify the validity of certain properties.

2.1 Concept Drift

In a classification task, we aim to predict a categorical target $y \in \mathbb{R}^1$ based on an instance $x \in \mathbb{R}^n$ [28]. Furthermore, we assume two or more classes $c = 1, ..., C$, where C represents the total number of classes present in the dataset. Both Hoens et al. [29] and Gama et al. [28] consider Bayes' theorem to describe the probability that an incoming instance x at timestamp t originates from class c, see Equation 1:

$$p_t(c|x) = \frac{p_t(c)p_t(x|c)}{p_t(x)} \tag{1}$$

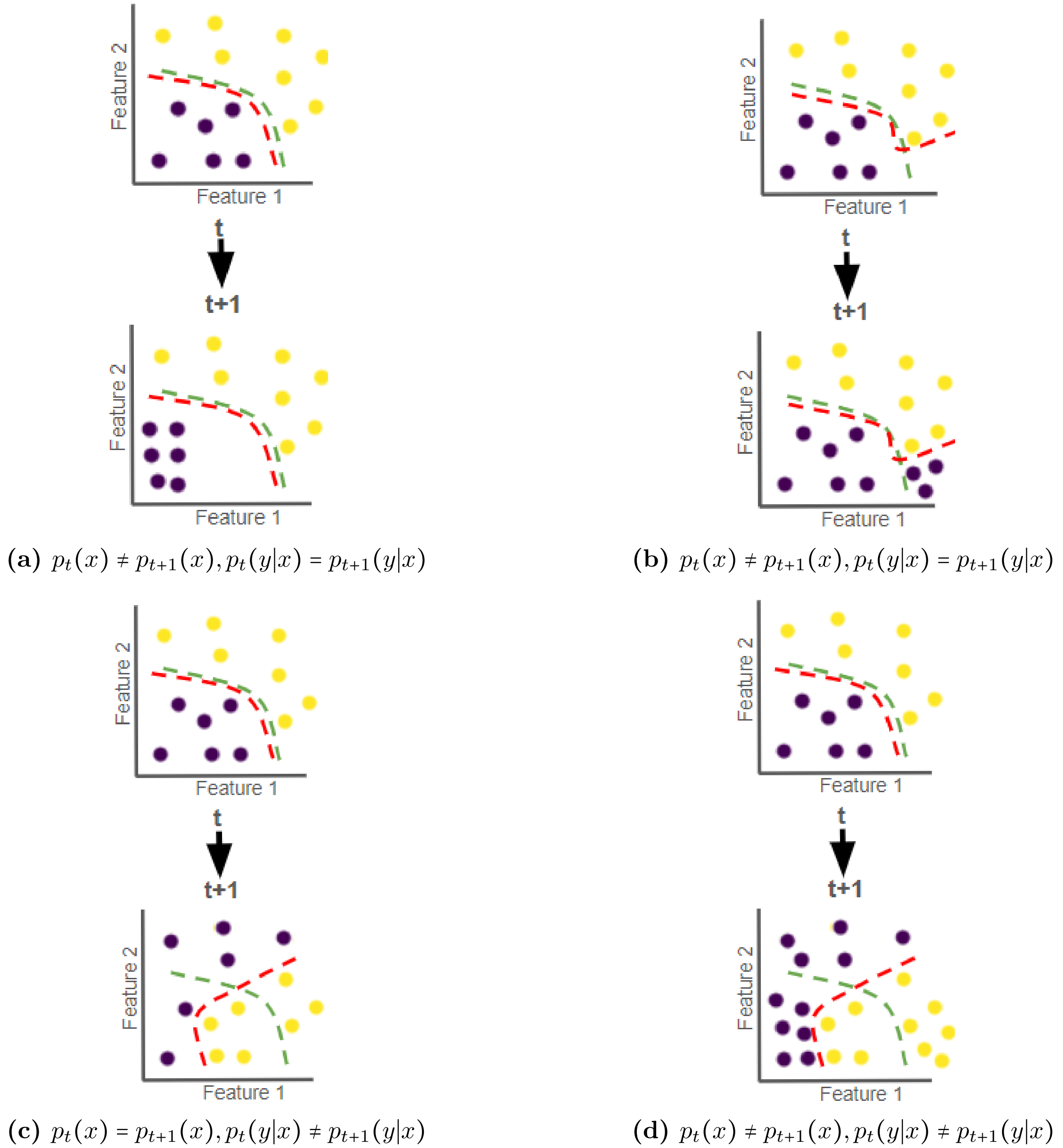
in which $p_t(c)$ represents the prior probability of observing each class at timestamp t, $p_t(x|c)$ denotes the probability that we observe x given each class c at timestamp t and $p_t(x)$ refers to the probability of observing x at timestamp t [29]. Notably, Hoens et al. [29] mention that $p_t(x)$ remains constant across all classes in C and can therefore be disregarded.

Generally, a classifier is trained and validated on a static batch of data after which the model is deployed into production. However, being in production, the classifier might deal with a continuous flow of incoming data also known as a data stream. Mathematically a data stream represents a set of samples $S = \{s_0, ..., s_t, ...\}$, where each s_i denotes a tuple (x_i, c_i) [1]. Per timestamp t a single s_t arrives. Key characteristics of data streams include the continuous or potentially infinite flow of incoming data, the temporal aspect which ensures sequential order and the dynamic nature [10, 11, 12]. In other words, data streams are a form of time series data due to the temporal component [10].

This dynamic nature of a data stream can result in changes in the underlying distribution of the observed incoming data over time, also known as *concept drift* [14, 28]. Concept drift is formally defined as follows, see Equation 2:

$$\exists x : p_t(x|c) \neq p_{t+1}(x|c) \tag{2}$$

In other words, concept drift represents a change in the joint distribution of x and c between t and $t+1$, where $t+1 > t$ [7]. Rewriting $p_t(x|c)$ into $p_t(x) \times p_t(c|x)$ [7] allows us to distinguish between the following events depicted in Figure 1:



(a) $p_t(x) \neq p_{t+1}(x), p_t(y|x) = p_{t+1}(y|x)$

(b) $p_t(x) \neq p_{t+1}(x), p_t(y|x) = p_{t+1}(y|x)$

(c) $p_t(x) = p_{t+1}(x), p_t(y|x) \neq p_{t+1}(y|x)$

(d) $p_t(x) \neq p_{t+1}(x), p_t(y|x) \neq p_{t+1}(y|x)$

Figure 1: Four possible events triggered by changes between $p_t(x)$ and $p_{t+1}(x)$, and changes between $p_t(y|x)$ and $p_{t+1}(y|x)$ [7, 28, 30]. In all figures, the red line represents the separator and the green line denotes the learned decision boundary. Figure 1a represents data drift or covariate shift, i.e., $p_t(x) \neq p_{t+1}(x)$, while the relationship between the instances and targets $p_t(y|x) = p_{t+1}(y|x)$ remains. This is also known as virtual drift since the decision boundary still holds valid. In contrast, Figure 1b represents a scenario in which the covariate shift impacts a model's performance. Figure 1c demonstrates a change in the probability of a certain instance belonging to a class, i.e., $p_t(y|x) \neq p_{t+1}(y|x)$, while $p_t(x) = p_{t+1}(x)$. Although the probability distribution of the input features remains, the mapping between these instances and classes changed resulting in an invalid decision boundary. Finally, Figure 1d depicts both changes i.e., $p_t(x) \neq p_{t+1}(x), p_t(y|x) \neq p_{t+1}(y|x)$. Figures adjusted from [29].

As can be seen in Figure 1, the scenarios presented in Figures 1b, 1c and 1d impact the performance of a model since the learned decision boundary is no longer valid. For instance in Figure 1c, the top four instances were correctly classified at timestamp t. However, at timestamp $t+1$, the top four instances originate from the purple class and will be incorrectly classified as orange instances. In this thesis, we will exclusively focus on the notion of concept drift as defined in Figure 1c, i.e., $p_t(x) = p_{t+1}(x), p_t(y|x) \neq p_{t+1}(y|x)$.

2.1.1 Types of Concept Drift

To properly define a change in the underlying distribution of the data, we adopt the concept of *context* from [11], which denotes a set of instances originating from a static probability distribution. This means that a data stream can be constructed as a concatenation of contexts. The transition from one context to the other (i.e., types of concept drift) can have several forms as demonstrated in Figure 2.

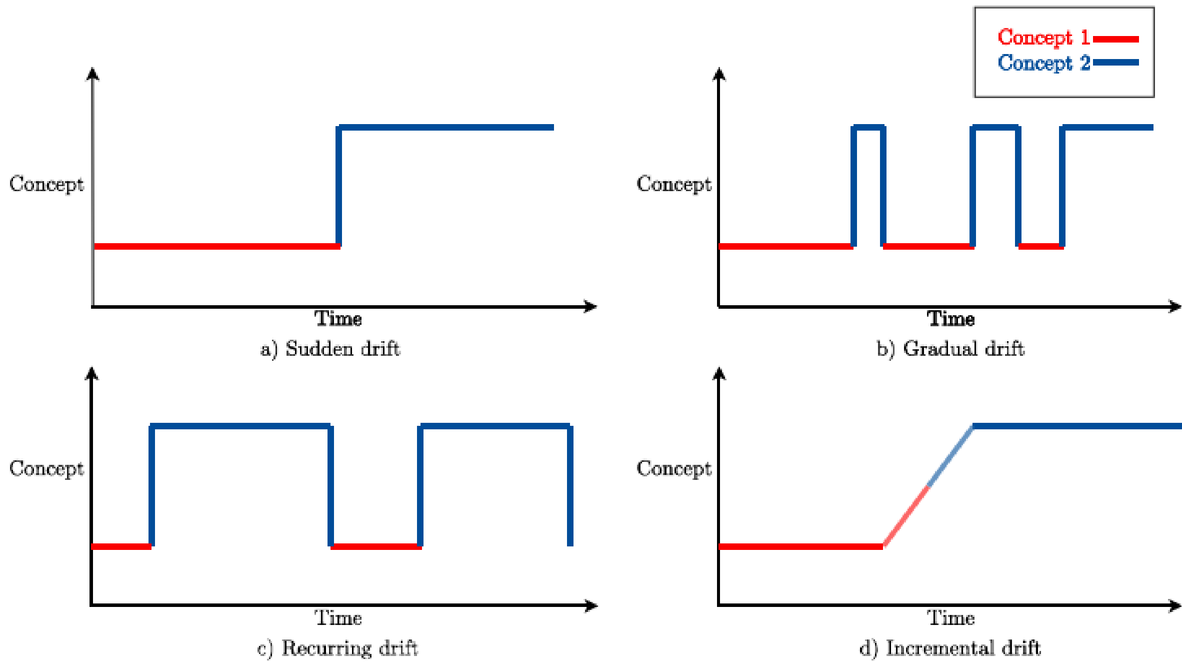


Figure 2: Four types of concept drift: a) sudden drift, b) gradual drift, c) recurring drift and d) incremental drift. Each drift is visualised over time according to two contexts: the red concept and the blue concept. Figure is retrieved from [13].

As we can see in Figure 2, four different types of concept drift are illustrated [13]: Sudden drift represented in Figure 2a refers to an abrupt change in context at one point in time. Gradual drift represented in Figure 2b denotes a change from one context to another through alternations between the two at several points in time. Recurring drift represented in Figure 2c similar to gradual drift consists of alternations between two different *contexts*. What makes recurring drift distinct from gradual drift is that the old context reappears at a certain point in time, instead of being fully replaced by the new *context* as is the case in gradual drift. An example of recurring drift is seasonality trends. Incremental drift represented in Figure 2d refers to a continuous change from one context to another without a specific point in time at which the switch occurred. Arguably, it can be considered a subtype of gradual drift or a separate type of concept drift.

2.1.2 Concept Drift Detection Framework

A general framework to detect concept drift is presented in Figure 3.

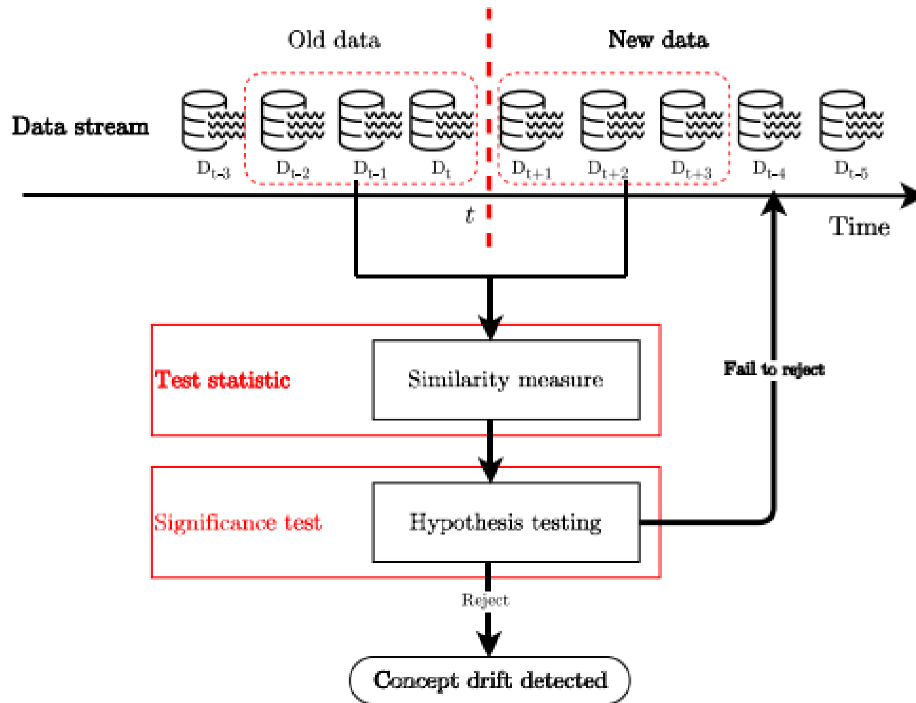


Figure 3: General outline of a concept drift detection framework. The first stage consists of defining batches of data since a single instance cannot provide enough information to detect concept drift. The second stage involves the measurement of statistical dissimilarity between two batches retrieved from the first stage. In the final stage, the severity of the computed test statistic is assessed via a hypothesis framework to determine if concept drift is present. Figure is retrieved from [13].

The general framework to detect concept drift contains three key stages: First, a reference batch consisting of old data is preserved. In general, the model is trained on this batch of data. With the incoming new data instances, a target batch is taken to compare against the reference batch. Second, by measuring the similarity between the reference batch and the target batch, a test statistic is computed that reflects the amount of dissimilarity between the two batches. Finally, a mechanism based on a certain threshold determines if the test statistic exceeds the threshold. This means that concept drift occurred.

The approach described in Figure 3 to deal with concept drift is known as adaptive learning algorithms [31]. In contrast, evolving learning models update the model on a regular frequency [31]. This means that updates might occur even if no concept drift is present. Therefore, a disadvantage of evolving learning models is the potentially high cost of retraining without knowing for sure if there is concept drift.

Adaptive learning algorithms aim to mitigate this disadvantage. However, as stated by Sebastião and Gama [32], the difficulty of designing a concept drift detection algorithm lies in effectively balancing robustness to noise while maintaining sensitivity to shifts in the underlying data distribution. Namely, it might be difficult for such an algorithm to deal with outliers potentially causing higher false alarm rates (i.e., detecting a concept drift where there is no concept drift) [32]. Furthermore, Khamassi et al. [31] mentioned the following three difficulties: “(i) how to track concept drift, (ii) which data to keep and which data to forget, and (iii) how to adapt the learner parameters and structure in order to react according to these new environment requirements.” Despite these difficulties, this thesis will only focus on adaptive learning models to deal with concept drift.

2.1.3 Concept Drift Detection Methods

There exist various methods to deal with concept drift. Lu et al. [7] identified the following three categories: error rate-based drift detection, data distribution-based drift detection and multiple hypothesis test drift detection.

The majority of the concept drift detection methods fall into error rate-based detection [7, 13]. In general, these methods monitor the error rate of the model over the incoming data instances. As soon as the error rate exceeds a predefined threshold, the model triggers a warning meaning that there is concept drift. It is important to note that error rate-based detection methods depend on the ground truth labels which are not always available [13].

Data distribution-based drift detection methods exploit a distance function to determine the dissimilarity between a reference window and a window containing the most recent instances [7]. If the two distributions statistically differ from each other, concept drift is present. Another difference between error rate-based detection and data distribution-based drift detection methods is that the latter directly takes the incoming instances as input. As a consequence, data distribution-based drift detection methods do not need the ground truth labels [20]. Additionally, according to Lu et al. [7], these methods are capable of accurately determining at which timestamp the drift occurred.

In contrast to error rate-based detection and data distribution-based drift detection, multiple hypothesis test drift detection methods use more than one hypothesis test to detect concept drift. Based on Lu et al. [7], we can categorise multiple hypothesis test drift detection methods into the following two groups: parallel multiple hypothesis tests and hierarchical multiple hypothesis tests. Parallel multiple hypothesis tests combine multiple drift detection methods which each measure the dissimilarity and verify the significance of the computed test statistics [7, 13]. On the other hand, hierarchical multiple-hypothesis methods typically involve multiple layers to detect concept drift. In general, such a method contains a detection layer responsible for identifying potential concept drift using a drift detector and a subsequent validation layer in which an additional hypothesis test is performed to validate the detected concept drift [7].

In the upcoming section 2.2, we will explain the necessary theory for the foundation of our concept drift detection mechanism.

2.2 Local Robustness Verification

In general, neural networks have proven their performance in numerous tasks such as image classification [26]. However, the integration of neural networks in safety-critical domains is limited due to concerns about their unpredictable behaviour and robustness of such neural networks [23]. Particularly in neural networks applied in safety-critical domains, it is crucial to ensure a certain degree of robustness concerning the relationship between the input and output of the neural network.

For instance, Szegedy et al. [22] demonstrated that small and oftentimes undetectable perturbations to the input of a neural network, i.e., *adversarial example*, can alter the output of a neural network to the extent that it generates a wrong prediction. Under this assumption, an adversarial example is defined as the result obtained after adding a small perturbation to the original correctly classified instance which triggers the model to misclassify it. Figure 4 illustrates an adversarial example in a medical image classification task [33] in which the manipulation of a medical image through the addition of a small amount of noise results in a misclassification. Arguably, the disparities between the original image on the left and the perturbed image on the right are invisible to the human eye.

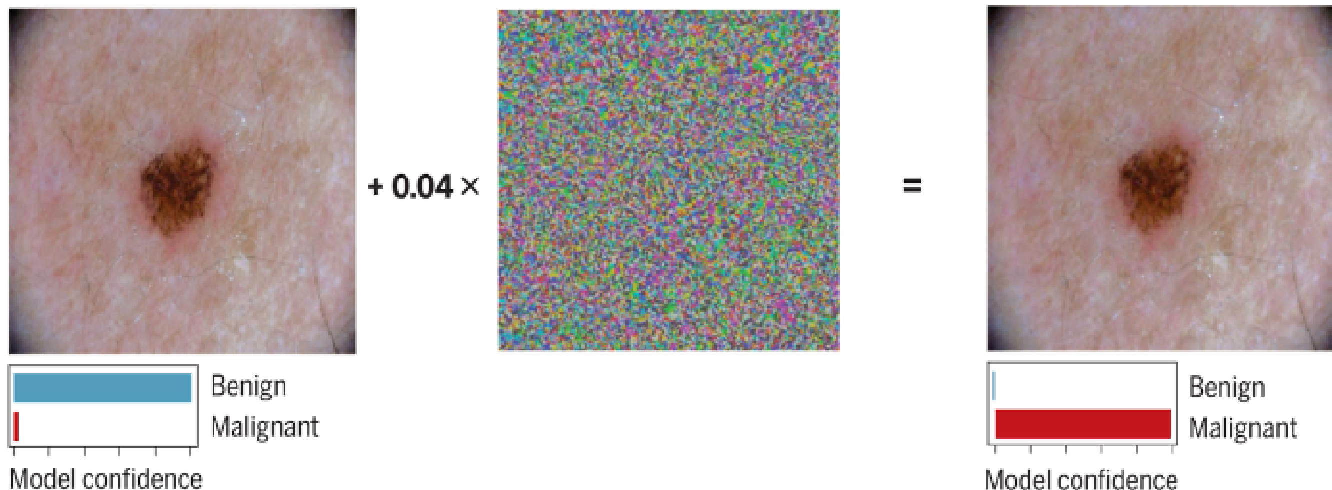


Figure 4: An example of an adversarial example applied to a deep neural network for detecting skin cancer, adapted from [33]. The image on the left side contains a spot with a high probability of being benign according to the model. On the contrary, after adding a slight amount of noise on the pixel level to this image, the model predicts the spot on the altered image on the right side to be malignant with a high confidence level.

As observed by Goodfellow et al. [21], a performance metric such as accuracy is incapable of reflecting the extent to which a model might be vulnerable to adversarial examples. Moreover, small distortions to the original instances might trigger a neural network to generate wrong predictions, whereas this neural network might yield high performance in terms of accuracy. A technique that aims to describe the vulnerability of a model to adversarial examples is *local robustness verification* [21]. Local robustness verification formally verifies if original instances combined with a varying amount of perturbation can make the neural network generate a wrong output.

In more detail, following the notation as presented in the work of Bosman et al. [25], we assume a static trained neural network classifier f_θ in $\mathbb{R}^n \rightarrow \mathbb{R}^m$, where θ represents the trainable parameters of the neural network, n denotes the size of the input vector in terms of features and m stands for the number of classes an input can be mapped to. Additionally, x_0 stands for a non-perturbed instance with correct class $\lambda(x_0)$ and x for a perturbed instance. As discussed above, local robustness verification assesses instances with a varying amount of perturbation. We can define the amount of perturbation as a region around x_0: $G_{p,\varepsilon}(x_0) = \{x : ||x - x_0||_p \leq \varepsilon\}$, where p refers to a distance metric and ε denotes the size of the perturbation. Thus, local robustness verification verifies if a neural network tolerates an adversarial example within the region $G_{p,\varepsilon}(x_0)$ without generating a wrong output.

A distance metric can be employed to ensure that adversarial examples are within the magnitude of the perturbation ε from the original input x_0 [34]. One such family of distance metrics are the L_p-norm distances. Three common L_p-norm distances applied in neural network verification are [35]:

$$\begin{aligned} L_1(\text{Manhattan distance}):& \ \textstyle\sum_{i=1}^{n} |x_i| \\ L_2(\text{Euclidean distance}):& \ \sqrt{\textstyle\sum_{i=1}^{n} x_i^2} \\ L_\infty(\text{Chebyshev distance}):& \ \max_i |x_i| \end{aligned} \tag{3}$$

where x represents an input for a neural network. A visual depiction of the three L_p-norm distances mentioned in Equation 3 is present in Figure 5.

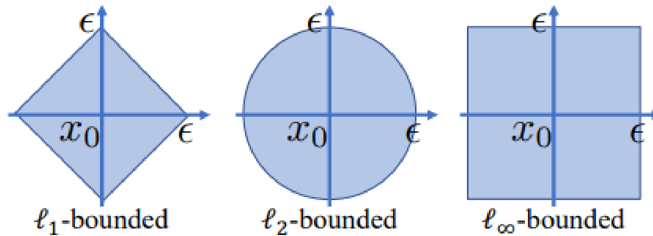


Figure 5: Common regions around the original instance x_0 with distance ε to restrict the space in which adversarial examples are generated in local robustness verification. From left to right: the L_1 norm (Manhattan distance) forms a diamond shape, the L_2 norm (Euclidean distance) forms a circular shape and the L_∞ norm (Chebyshev distance) forms a square shape. The figure is retrieved from [35].

Providing formal guarantees about neural networks proves difficult since they consist of large architectures and are non-linear as well as non-convex [26]. For instance, a neural network with ReLU as an activation function renders the verification problem NP-hard [26].

Specifically for local robustness verification, we are interested in formally assessing if a neural network f_θ holds a certain property $\mathcal{P}$ for all possible inputs x constrained to the domain defined by $G_{p,\varepsilon}(x_0) = \{x : ||x - x_0||_p \leq \varepsilon\}$ [36]. Following the example presented in [36], we assume a binary classification task for which we trained f_θ and a positive instance x_0 with correct class $\lambda(x_0) \geq 0$, i.e., $f_\theta(x_0) \geq 0$. Since we are interested in formally proving that a certain amount of perturbation to this instance leads to a correct output $f_\theta(x) \geq 0$, we set the property $\mathcal{P}$ to exclusively positive numbers $\mathbb{R}^+$. In other words, we aim to solve the optimization problem: $f^* = \min f_\theta(x), \forall x \in G_{p,\varepsilon}(x_0)$, which provides an exact solution.

Neural network verifiers are typically categorised into complete verification and incomplete verification [35] to assess whether a property holds. Incomplete verification methods generally relax the optimization problem to gain advantages in terms of computation time at the cost of not being able to guarantee an outcome for each problem instance x. Namely, by relaxing the non-convexity of f_θ, we generate a lower bound $\underline{f} \leq f^*$ which renders the verification problem more tractable to solve [27]. This results in correct verification for cases where $\underline{f} \geq 0$ since we can conclude that $f^* \geq 0$. However, in case $\underline{f} \leq 0$, we fail to verify $\mathcal{P}$.

On the other hand, complete verification guarantees to provide the exact solution f^*. Since verifying if a property holds is NP-hard, a disadvantage of complete verifiers is that they are expensive and do not scale to larger neural network sizes [35]. For this reason, oftentimes a time limit is introduced.

Remember that in our example, we are interested in finding the global minimum of $f^* = \min f_\theta(x), \forall x \in G_{p,\varepsilon}(x_0)$ to verify if $\mathcal{P} = \mathbb{R}^+$ holds. In general, this is done through means of finding counter-examples [26, 23]. For instance, if the global minimum is negative, the verifier returns satisfied meaning that the property does not hold [23]. On the contrary, if the global minimum is positive, the verifier returns unsatisfied meaning that the property holds [23]. Incomplete verifiers have a third output: unknown since the relaxation might result in cases in which we are unable to conclude if a property holds. Due to the time limit introduced in complete verifiers, they can also output timeout.

A fundamental framework allowing for the formulation of local robustness verification problems is Mixed-Integer Linear Programming (MIP) [23]. In general, MIP provides a mathematical framework to formulate optimization problems. In essence, it involves specifying a linear objective function alongside a set of linear constraints. The objective function is defined as a linear combination consisting of real- or integer-valued decision variables. The ultimate goal is to find a solution that

maximises or minimises the objective function while simultaneously satisfying the set of linear constraints.

A solver for MIP-formulated neural network verification that attracted a lot of attention is Branch-and-Bound (BaB) [23, 37]. BaB [23, 37] is considered a complete verification method. According to Li et al. [35], a majority of the submissions of the International Verification of Neural Networks Competition (VNN-COMP) included BaB [38], with α,β-CROWN [27] being the winner of the three most recent iterations.

The working principles of the BaB algorithm from a high-level perspective are as follows [23]: The algorithm iteratively divides the global MIP formulation into smaller subproblems (i.e., branches) partitioned according to the input space. Each branch represents a subspace of the feasible region which facilitates the search for a solution to the MIP. By maintaining lower and upper bounds (i.e., bounding) for each branch, BaB aims to provide an exact outcome for each branch. If the verification fails, BaB performs another iteration of dividing into subproblems and verifying the properties. Notably, subproblems are solved through incomplete verification algorithms rendering the search for a solution computationally cheaper [37]. Hence, via a divide-and-conquer approach, the algorithm efficiently narrows down the space wherein it should focus its attention on solving the neural network verification problem.

2.3 Key Ideas

In this section, we introduced the notion of a data stream characterised by the continuous flow of incoming data, sequential order and dynamic nature. A model deployed on such a data stream can exhibit a deterioration in performance due to concept drift. Specifically, we consider the following description of concept drift in this thesis: $p_t(x) = p_{t+1}(x), p_t(y|x) \neq p_{t+1}(y|x)$. There are four types of concept drift: sudden drift, gradual drift, recurring drift and incremental drift. Furthermore, methods to deal with concept drift are divided into three categories: error rate-based drift detection, data distribution-based drift detection and multiple hypothesis test drift detection. Error rate-based methods rely on monitoring changes in the error rate as an indicator of concept drift. On the other hand, data distribution-based mechanisms exploit a distance function directly applied to two input data distributions to detect concept drift. Multiple hypothesis methods utilise more than one hypothesis test to detect concept drift.

Regarding the robustness of neural networks, we have explored adversarial examples which are small and oftentimes undetectable perturbations to the input triggering the neural network to make a wrong prediction. Local robustness verification provides a formal framework to assess to which extent a neural network is vulnerable to adversarial examples. Given that solving this task is NP-hard, there is a general distinction between complete and incomplete neural network verifiers. Incomplete verification methods, while being computationally cheaper, cannot guarantee an outcome to the verification problem. In contrast, complete verification methods do provide an exact outcome to the verification problem at the cost of increased computational expenses. A state-of-the-art neural network verifier is α,β-CROWN making use of BaB.

3 Related Work

Various methods exist for detecting concept drift. In this section, we will discuss the following prominent methods: DDM, EDDM, ADWIN and KSWIN. In addition, we will explain a more recent method named UDD which uses uncertainty estimates of a neural network as input to ADWIN for detecting concept drift. We will mention several disadvantages of these methods that our proposed method aims to address. Subsequently, we introduce our method which belongs to the category of error rate-based methods. Our approach is inspired by the lower bound to the critical epsilon value defined in the work of Bosman et al. [25]. Essentially, these values provide a distance measure to the decision boundary. We hypothesise that the distributions of these values differ significantly before and after the occurrence of concept drift.

3.1 Concept Drift Detection Methods

The largest group of drift detection methods is error rate-based drift detection, as discussed in Section 2.1.3. A well-known error rate-based method is Drift Detection Method (DDM) [11]. DDM relies on the prediction of a model to update the error rate within a context window. Specifically, DDM accepts binary input in which 0 represents a correct classification and 1 denotes a misclassification. The input for DDM is assumed to follow a Binomial distribution [20]. DDM keeps track of the minimum error rate (i.e., the mean over the observed zeros and ones) and the minimum standard deviation. A warning mode is triggered as soon as the current error rate and standard deviation exceed a confidence level of 95% [11]. Suppose the current error rate and standard deviation exceed 99% [11], in that case, it is considered concept drift.

Early Drift Detection Method (EDDM) [39] focused on improving DDM specifically for slow incremental drift. Similarly to DDM, EDDM relies on binary inputs and assumes that these follow a Binomial distribution [20]. Instead of the error rate and standard deviation, EDDM exploits the average distance and variance between two consecutive misclassifications. Baena-García and colleagues [39] assume that as the model learns, this average distance should increase. This means that a decrease in the average distance indicates the potential presence of concept drift. EDDM keeps track of the maximum distance between two consecutive misclassifications and the maximum standard deviation. Similarly to DDM, EDDM first triggers a warning zone before indicating a concept drift.

A typical window-based method to monitor the error rate of a model is ADaptive WINdowing (ADWIN) and the optimised successor ADWIN2 [16]. Based on a window with a size defined by the user, ADWIN partitions the window into two large enough subwindows. Importantly, ADWIN compares the difference between the two means of the sub-windows against a defined confidence threshold. If the difference in means exceeds the threshold, the concept present in the first subwindow is different from the concept in the second subwindow. Thus, the difference in terms of the two sample means serves as the test statistic to verify whether there is concept drift [7]. ADWIN accepts binary as well as real-valued inputs without assumptions about the distribution of the input data [16].

A state-of-the-art data distribution-based detection method is KSWIN [1]. KSWIN maintains the n most recent samples and generates two distributions: one window containing the r most recent instances and another window consisting of r uniformly sampled instances excluding the ones present in the first window. The authors opted for uniform sampling since this method is parameter-free and provides no discrimination towards the selection of instances from a distribution with unknown properties [1]. As a test statistic to highlight concept drift, the authors utilise

the Kolmogorov-Smirnov test which computes the absolute distance between two cumulative distributions [1]. Table 1 summarises the discussed concept drift detection methods.

Table 1: Comparison between four concept drift detection methods.

Drift Detector	Category	Needs Labels	Input	Parameters
DDM	error rate-based	yes	binary	-
EDDM	error rate-based	yes	binary	-
ADWIN	error rate-based	depends on input	arbitrary	significance value window size
KSWIN	data distribution based	depends on input	arbitrary	detection window test statistic

As shown in Table 1, DDM, EDDM and ADWIN are originally error rate-based methods exclusively accepting the error rate as input. KSWIN being a data distribution-based method, accepts features as input.

Instead of the error rate or the direct features as input, Baier et al. [20] introduced Uncertainty Drift Detection (UDD). UDD leverages uncertainty estimates from a deep neural network as input to ADWIN to detect concept drift. For each incoming instance, UDD computes the uncertainty by performing multiple forward passes through a deep neural network using Monte Carlo Dropout [20]. During inference, Monte Carlo Dropout applies different dropout masks per forward pass to generate a distribution of predictions. It is important to note that Monte Carlo Dropout requires the network architecture to contain dropout layers. UDD applies to both classification and regression problems. For this reason, we extended the input of ADWIN to real-valued in Table 1.

To summarise, we have discussed three different inputs to drift detectors: error rate, features and uncertainty. According to Baier et al. [20], methods relying only on features as input data might be prone to an increased number of false positives due to detecting changes on the feature level instead of concept drift. As a consequence, unnecessary retraining of the model may occur. For instance, it might detect virtual drift $p_t(x) \neq p_{t+1}(x), p_t(y|x) = p_{t+1}(y|x)$ which has no impact on the performance of the model [20]. Baier and colleagues [20] addressed this problem via their proposed UDD method. However, a drawback of UDD is that the architecture of the neural network requires dropout layers.

Regarding the error rate, drift detector methods typically identify concept drift once the error rate decreases by a certain margin or the distance between two consecutive misclassifications decreases. However, a disadvantage of such methods is that by the time concept drift is detected, the model's performance has already been degraded. Therefore, we propose a method that aims at monitoring changes in the distance of instances towards the decision boundary. By using these distance measures as input to a drift detector, we expect to be able to proactively react to concept drift before a severe decline in model performance occurs. In the upcoming section, we will explain the fundamental basis on which we build our concept drift detection method.

3.2 Detecting Concept Drift Through Critical ε Values

In Section 2.2, we introduced ε denoting the amount of perturbation added to an instance. Instead of using a predefined perturbation radius, Bosman et al. [25] introduced the critical epsilon value ε^*. The critical epsilon value ε^* represents the maximum perturbation size that can be added to a specific instance such that the resulting distorted instance cannot trigger the neural network to generate a wrong prediction i.e., $x \in \{x : \|x - x_0\|_p \leq \varepsilon^*\}$ [25]. In other words, a perturbation size

larger than ε^* will generate an adversarial example and consequently trigger the neural network to generate a wrong prediction.

However, current neural network verification frameworks are expensive in terms of running time making it infeasible to evaluate all possible ε values [25]. Rather, Bosman et al. [25] propose to find an empirical lower bound $\tilde{\varepsilon}^*$ to the critical epsilon ε^* via discretization. This means that only a set of epsilon values $\mathcal{E} = \{\varepsilon_0, ..., \varepsilon_n\}$, in which $\varepsilon_i < \varepsilon_{i+1}, \forall i \in \{0, ..., n\}$ [25] is assessed. In more detail, within the range of epsilons $\mathcal{E}$, we are interested in finding two ε values next to each other such that the smaller one ε_{i-1} yields no *adverserial example* and the larger one ε_i contains an adversarial example. If such a pair exists, the empirical lower-bound $\tilde{\varepsilon}^*$ is set at ε_{i-1} and the critical epsilon ε^* exists in the range of this pair: $[\varepsilon_{i-1}, \varepsilon_i)$.

Based on this explanation, it becomes clear that $\tilde{\varepsilon}^*$ functions as a distance measure towards the decision boundary of a trained neural network. This means that for each incoming instance, we can compute its distance to the decision boundary via the method proposed in the work of Bosman et al. [25]. Specifically, we leverage this method to obtain $\tilde{\varepsilon}^*$ for a correctly classified instance, whereas we assign 0 for incorrectly classified instances. Compared to the original error rate-based input, we provide additional information (i.e., the distance towards the decision boundary) through the $\tilde{\varepsilon}^*$ values.

As specified in Section 2.1, we only consider concept drift formally defined as $p_t(x) = p_{t+1}(x), p_t(y|x) \neq p_{t+1}(y|x)$. This means that the relationship between instances and their classes changes while the probability distribution of the features remains the same. Hence the learned decision boundary by a neural network no longer holds to a certain extent in the presence of concept drift. As a consequence, we will observe an increase in misclassified instances due to the inaccurate decision boundary. We think that the distributions of $\tilde{\varepsilon}^*$ values before and after concept drift will differ significantly. To this end, we believe that a change in the distribution of $\tilde{\varepsilon}^*$ values is an indicator of concept drift. In the next section, we will explain our approach in more detail.

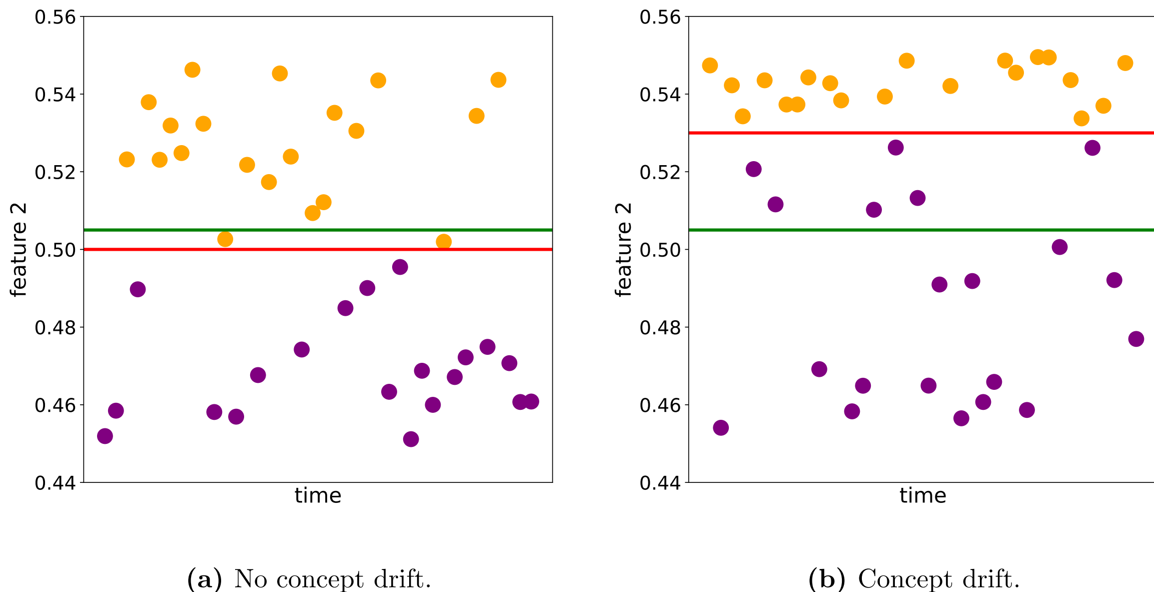
4 Methodology

This section provides a detailed explanation of our proposed method for detecting concept drift. First, we focus on the usage of $\tilde{\varepsilon}^*$ values to detect concept drift in binary classification tasks. We will explore the combined $\tilde{\varepsilon}^*$ distributions of both classes as well as the individual $\tilde{\varepsilon}^*$ distributions per class. Second, we introduce the $\tilde{\varepsilon}^*$-drift representing an approximation of the distance towards the decision boundary for misclassified instances. We provide a detailed procedure for computing $\tilde{\varepsilon}^*$-drift values. Finally, we discuss the integration of $\tilde{\varepsilon}^*$-drift values into classical drift detection methods and introduce an additional drift detector: the Mann-Whitney U test. Furthermore, we propose a threshold method as a drift detector which works exclusively with $\tilde{\varepsilon}^*$-drift values.

4.1 Lower Bound to the Critical ε Value

In Section 3.2, we introduced our idea for a concept drift detection method based on local robustness verification. To thoroughly explain our method, we make the following assumptions: I) we assume concept drift as defined by $p_t(x) = p_{t+1}(x), p_t(y|x) \neq p_{t+1}(y|x)$, II) we assume that the ground truth is always and directly accessible and III) we assume that there exists a single ultimate separator in the form of a rule to split data into two distinct groups (i.e., binary classification). We leave the extension of this method to multiclass classification and other problem types, such as regression, open for future work.

The sudden drift scenario based on a simplification of the SEA dataset visualised in Figure 6 serves as the foundation to explain our concept drift detection method:



(a) No concept drift. **(b)** Concept drift.

Figure 6: Sudden drift scenario. We assume a binary classification task in which instances consist of two features: *feature 1* and *feature 2*. The red lines denote the separator and the green lines represent the learned decision boundary by a model. The separator is a rule determining if an instance belongs to the orange or purple classes. The separator in Figure 6b changed compared to Figure 6a causing concept drift. It is important to note that the separator cannot be measured or assessed in practice.

First, we exclusively focus on Figure 6a. In this plot, we represent a binary classification task

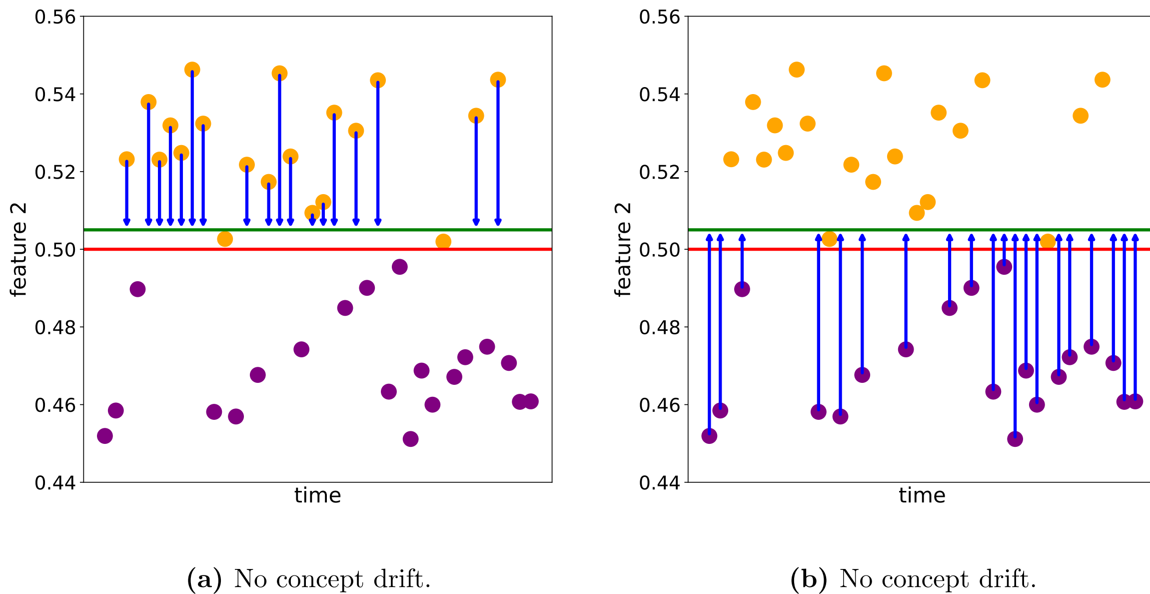
in which instances consist of two features: *feature 1* and *feature 2* in the range [0,1] and belong either to the orange class or the purple class. The red line denotes the separator representing a rule that decides if an instance belongs either to the orange class or the purple class. Specifically for the plot in Figure 6a, if *feature 2* is bigger than 0.5 an instance belongs to the orange class and otherwise it originates from the purple class. It is important to note that this separator cannot be measured or assessed directly in practice.

Furthermore, Figure 6a contains a green line representing the learned decision boundary by a neural network. Since a neural network might be incapable of perfectly learning the separator, the green line (decision boundary) is slightly above the red line (separator). As a result, misclassifications occur for instances in the area enclosed between the red and green lines.

In Figure 6b, we observe a change in the red line (separator). Notably, the red line shifted from 0.5 to 0.53 meaning that instances with *feature 2* bigger than 0.53 belong to the orange class and otherwise to the purple class. This is an example of sudden drift. Since we do not retrain the neural network, the green line (decision boundary) does not change. In other words, the area between the green line (decision boundary) and the red line (separator) increased. As a consequence, more instances fall in this area which causes the model to produce more wrong classifications compared to the situation without concept drift in Figure 6a.

As a first approach, we can compare the distributions of $\tilde{\varepsilon}^*$ values before and after concept drift. Remember that we can only measure the $\tilde{\varepsilon}^*$ value for correctly classified instances meaning that we can choose to ignore misclassified instances or set their $\tilde{\varepsilon}^*$ value equal to 0.

First, we focus on the distribution of $\tilde{\varepsilon}^*$ values prior to the occurrence of concept drift. The blue arrows in Figure 7 denote the $\tilde{\varepsilon}^*$ values for both the orange and purple classes. Although the blue arrows represent the exact distance towards the decision boundary for correctly classified instances (i.e., ε^* values), we opted for this representation of $\tilde{\varepsilon}^*$ values to simplify the explanations.



(a) No concept drift.

(b) No concept drift.

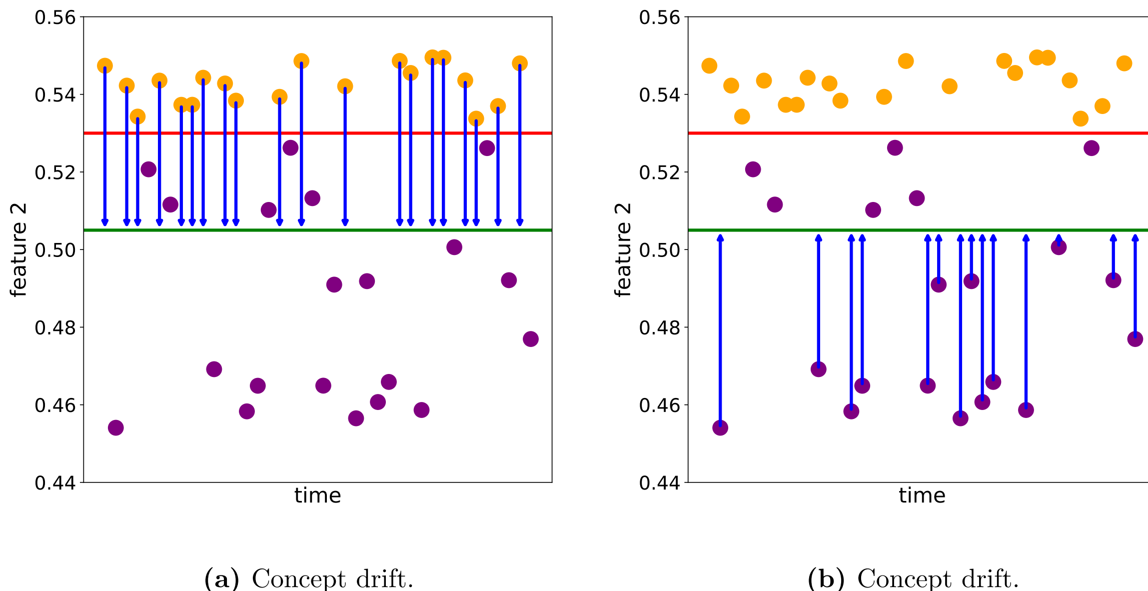
Figure 7: Distances towards the decision boundary in terms of $\tilde{\varepsilon}^*$ values in the presence of no concept drift.

Figure 7a denotes the $\tilde{\varepsilon}^*$ values as blue arrows for the orange class. The closer an orange instance is to the green line (decision boundary) the smaller the $\tilde{\varepsilon}^*$ value. Given that all instances

above the green line (decision boundary) are correctly classified as orange instances and the neural network verifier measures the distances of these instances towards the green line (decision boundary), we expect to observe all $\tilde{\varepsilon}^*$ values as defined by $\mathcal{E}$ with an infinite number of incoming instances.

A different scenario is depicted in Figure 7b. Figure 7b contains blue arrows representing the $\tilde{\varepsilon}^*$ values for the purple class. Again, the closer a purple instance is to the green line (decision boundary) the smaller the $\tilde{\varepsilon}^*$ value. However, the neural network did not learn the exact separator resulting in misclassifications in the area between the red line (separator) and green line (decision boundary). Because instances above the red line belong to the orange class, there cannot exist instances between the red line (separator) and the green line (decision boundary) originating from the purple class. As a consequence, we cannot observe $\tilde{\varepsilon}^*$ values for the purple class spanning the distance from the green line (decision boundary) to the red line (separator). Nevertheless by combining the distributions of the $\tilde{\varepsilon}^*$ values for the orange and purple class we observe all $\tilde{\varepsilon}^*$ values as defined by $\mathcal{E}$ with an infinite number of incoming instances in theory (see the pink line in Figure 10).

Similar observations can be made for the $\tilde{\varepsilon}^*$ distributions after concept drift represented in Figure 8.



(a) Concept drift.

(b) Concept drift.

Figure 8: Distances towards the decision boundary in terms of $\tilde{\varepsilon}^*$ values in the presence of concept drift.

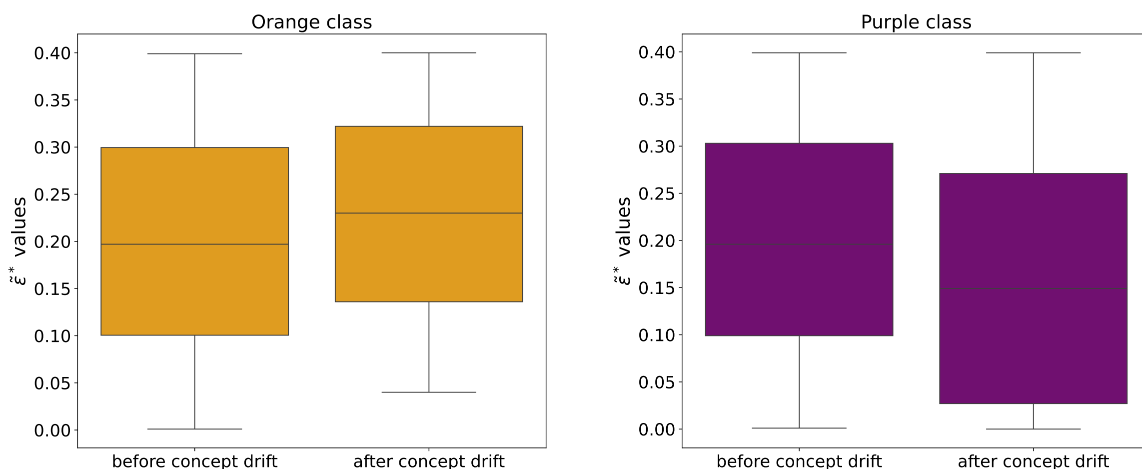
Given that the red line (separator) shifted from 0.5 to 0.53 and the green line (decision boundary) remains, Figure 8b is similar to Figure 7a. Namely, we expect to find all $\tilde{\varepsilon}^*$ values in $\mathcal{E}$ for the purple class, since all instances below the green line (decision boundary) are correctly classified.

To this extent, Figure 8a represents a more extreme example demonstrated in Figure 7b. After the concept drift, all instances above the red line (separator) belong to the orange class, whereas the instances below this line originate from the purple class. Hence, misclassifications between the red line (separator) and the green line (decision boundary) happen solely for instances from the purple class. In other words, we would observe more $\tilde{\varepsilon}^*$ values equal to 0 due to the increase in misclassified instances as depicted in Figure 9b. The distribution of $\tilde{\varepsilon}^*$ values after concept drift for

the purple class shifts downwards compared to the distribution before concept drift for the purple class.

Regarding the orange class, it is impossible to observe orange instances in the area denoted by the red line (separator) and green line (decision boundary). Consequently, $\tilde{\varepsilon}^*$ values smaller or equal to the distance between the red line (separator) and the green line (decision boundary) are non-existent for the orange class as can be seen in Figure 9a. Notably, the distribution of $\tilde{\varepsilon}^*$ values for the orange class shifts upwards compared to the distribution for the orange class before concept drift.

To clarify, the distributions of $\tilde{\varepsilon}^*$ values of the orange and purple classes before concept drift are similar to each other as seen in Figure 9. Due to concept drift, we observe larger $\tilde{\varepsilon}^*$ values for the orange class and an increase in $\tilde{\varepsilon}^*$ values equal to 0 for the purple class due to increased misclassifications. In other words, the distributions of the orange and purple classes after concept drift are drifting away from each other. However, the missing smaller $\tilde{\varepsilon}^*$ values are still present in the distribution of $\tilde{\varepsilon}^*$ values of the purple class after concept drift. Again, combining the distributions of $\tilde{\varepsilon}^*$ values of both the orange and purple classes yields all possible $\tilde{\varepsilon}^*$ values in $\mathcal{E}$ (see the grey line in Figure 10).



(a) Orange class before and after concept drift. **(b)** Purple class before and after concept drift.

Figure 9: Distributions of $\tilde{\varepsilon}^*$ values for the orange class and purple class before and after concept drift. Figure 9a represents a shift in larger $\tilde{\varepsilon}^*$ values for the orange class. An increase in misclassifications represented as a $\tilde{\varepsilon}^*$ value of 0 decreases the distribution as seen in Figure 9b. The distributions after concept drift of the orange class and purple class are moving in opposite directions.

To conclude the first approach, we expect to observe all $\tilde{\varepsilon}^*$ values in $\mathcal{E}$ before and after concept drift given an infinite number of incoming instances as demonstrated in Figure 10. As seen in Figures 7 and 8, the distributions on a class basis differ and by combining them they cancel each other out. Hence, comparing the distributions of $\tilde{\varepsilon}^*$ values before and after concept drift will not work since the distributions are expected to be similar to each other.

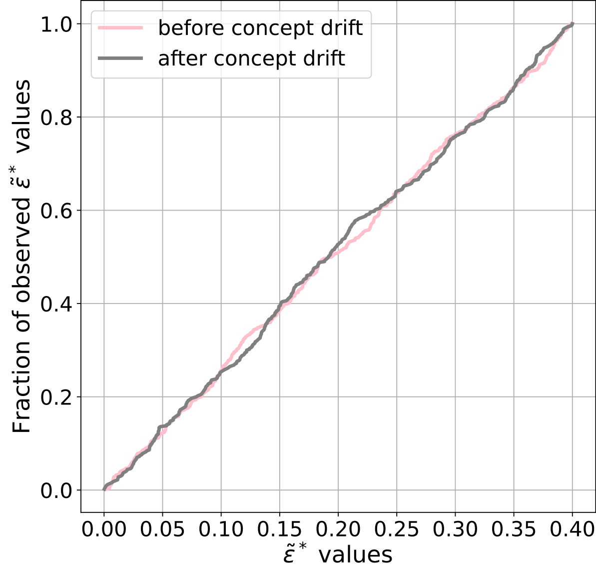
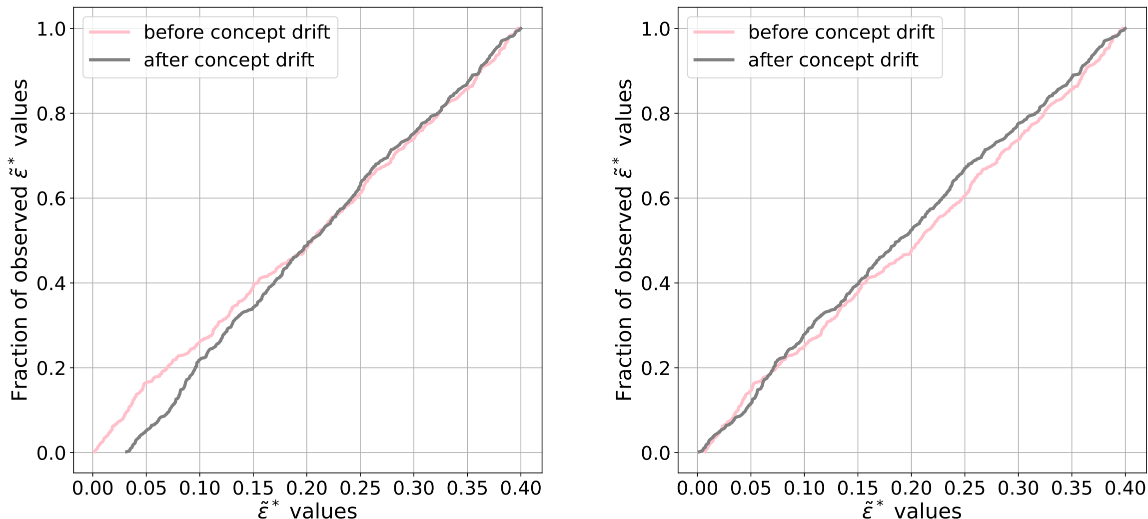


Figure 10: Distributions of observed $\tilde{\varepsilon}^*$ values before and after concept drift. Both distributions are combinations of the observed $\tilde{\varepsilon}^*$ values for the orange and purple classes as demonstrated in Figures 7 and 8.

Building on this first approach, a natural next step is to split the distributions of $\tilde{\varepsilon}^*$ values before and after concept drift based on the classes. Based on Figure 7a, we expect to observe all possible $\tilde{\varepsilon}^*$ values for the orange class before concept drift. On the contrary, we expect to observe only $\tilde{\varepsilon}^*$ values larger than the distance between the red line (separator) and the green line (decision boundary) for the orange class after concept drift as depicted in Figure 8a. Both distributions before and after concept drift are visualised in Figure 11a. Notably, the grey line representing the distribution of $\tilde{\varepsilon}^*$ values after concept drift shifted towards the right in comparison to the pink line denoting the distribution of $\tilde{\varepsilon}^*$ values before concept drift.

Regarding the purple class, we expect to observe solely $\tilde{\varepsilon}^*$ values larger than the distance between the red line (separator) and the green line (decision boundary) for the purple class before concept drift as depicted in Figure 7b. On the other hand, we expect to observe all possible $\tilde{\varepsilon}^*$ values for the purple class after concept drift as demonstrated in Figure 8b. The differences between these two distributions are minimal as depicted in Figure 11b.



(a) Orange class before and after concept drift.

(b) purple class before and after concept drift.

Figure 11: Distributions of observed $\tilde{\varepsilon}^*$ values before and after concept drift. Both distributions are split based on the class label (i.e., orange class or purple class).

The distribution shift depicted in Figure 11a contains information we could use to detect concept drift. Notably, this shift is observed in the lower part of the distributions as instances further located from the decision boundary are currently the only ones correctly classified. In other words, smaller $\tilde{\varepsilon}^*$ values are missing in the distribution of the orange class after concept drift compared to the distribution of $\tilde{\varepsilon}^*$ values of the orange class before concept drift.

For this approach to work, we depend on observing correctly classified instances with a distance towards the decision boundary equal to the $\tilde{\varepsilon}^*$ values in the lower parts of the distributions. A potential disadvantage is that it is not guaranteed to observe these specific instances while the model might already demonstrate a drop in performance due to concept drift. Namely, an exceptional scenario would be to solely observe misclassified instances and instances yielding larger distances towards the decision boundary than the $\tilde{\varepsilon}^*$ values in the lower parts of the distribution. In such a case, our approach would be unable to detect concept drift while the model demonstrates a lower performance.

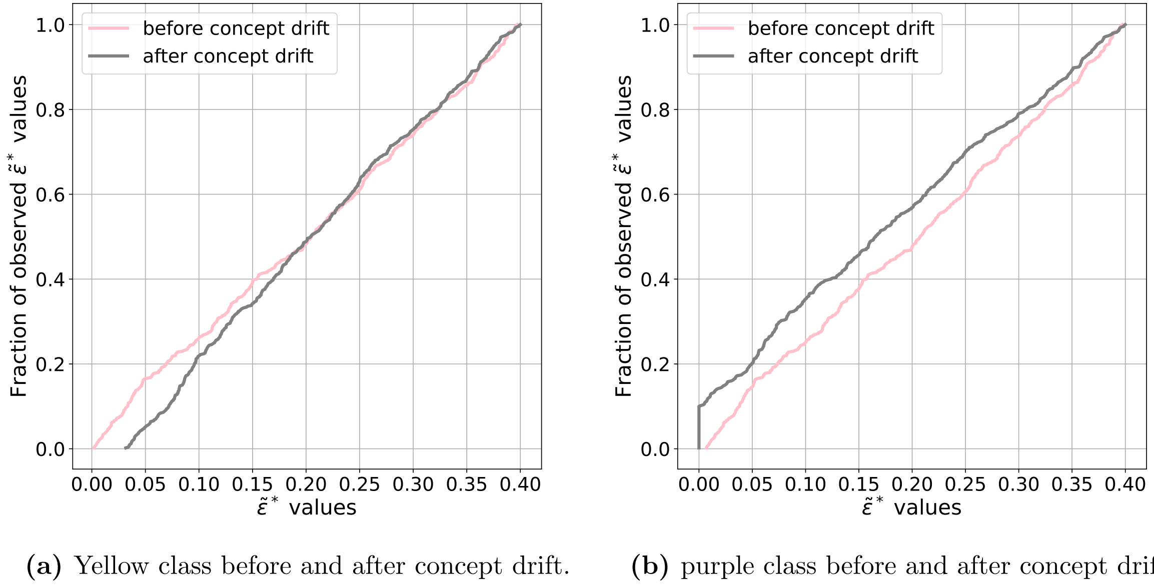
An opposite scenario might also occur in which we observe correctly classified instances with the $\tilde{\varepsilon}^*$ values in the lower parts of the distribution and zero misclassified instances. In this specific scenario, we will be faster in detecting concept drift than regular error rate-based methods.

Another drawback of this approach is the necessary split into classes. For each new correctly classified instance, we need to verify whether it belongs to the orange or purple classes. This means that we need a drift detector per class increasing the computational costs.

We conducted experiments in an early experimental phase to assess the performance of this class-splitting approach to detect concept drift. Based on preliminary results, we observed that this approach is relatively slow and produces a higher number of false positives compared to the regular error rate. Furthermore, a significant difference between the distributions of $\tilde{\varepsilon}^*$ values before and after concept drift was observed in the purple class rather than in the orange class. Despite these findings, this method does provide information in terms of a shift in the distribution of $\tilde{\varepsilon}^*$ values before and after concept drift, which we leave open for future work.

Therefore, we might consider a third approach in which we include misclassifications while still splitting the distributions based on classes. We assigned a 0 as $\tilde{\varepsilon}^*$ value for misclassified instances.

Similar to the error rate-based methods, we would observe an increase in 0 values after concept drift occurred. This difference is demonstrated in Figure 12.



(a) Yellow class before and after concept drift. **(b)** purple class before and after concept drift.

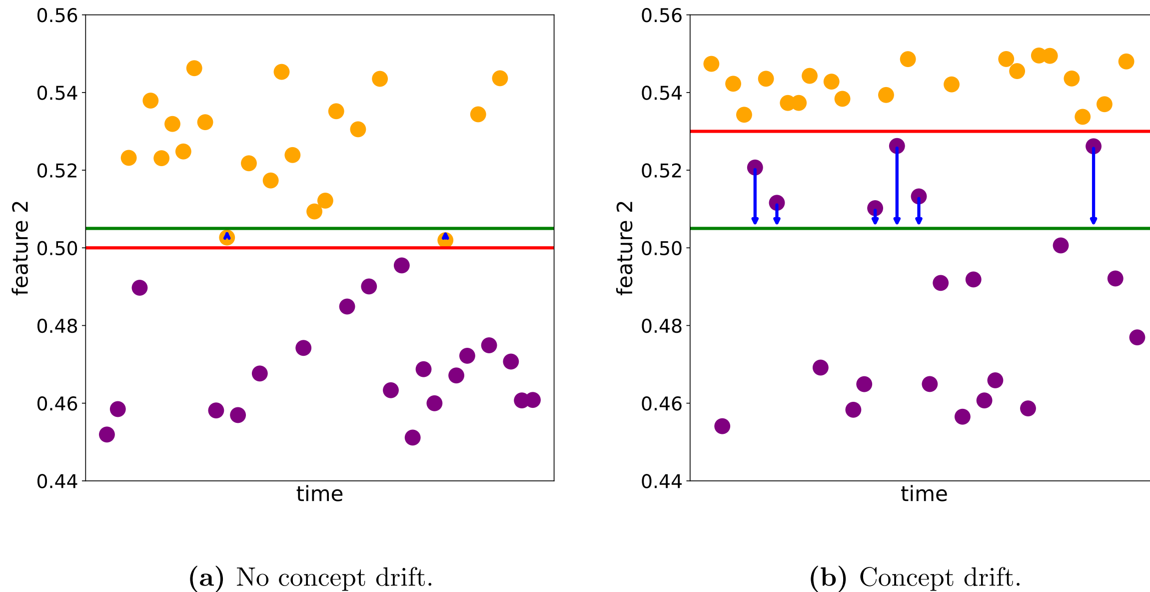
Figure 12: Distributions of observed $\tilde{\varepsilon}^*$ values including misclassifications before and after concept drift. Both distributions are split based on the class label (i.e., orange class or purple class).

As becomes clear from Figure 12b, we observe an increase in zero $\tilde{\varepsilon}^*$ values for the purple class after concept drift. However, to get this difference in zero $\tilde{\varepsilon}^*$ values before and after concept drift, we are still dependent on misclassification. For this reason, this approach defaults to the regular observation of the error rate and there is no advantage in terms of detection time.

Up to this point, we have mainly focused on the distance towards the decision boundary for correctly classified instances via $\tilde{\varepsilon}^*$ values. Simply comparing the distributions of $\tilde{\varepsilon}^*$ values before and after concept drift does not work since the classes cancel each other out. Splitting the distributions of $\tilde{\varepsilon}^*$ values before and after concept drift based on classes yields a shift in the distribution after concept drift compared to before concept drift for a specific class. However, neglecting misclassified instances and dividing the information over two concept drift detectors might result in delays in detection time compared to regular error rate-based methods that use a single concept drift detector. Namely, it is not guaranteed to receive enough of these instances to detect a significant shift in the lower parts of the distributions before and after concept drift. Including misclassified instances in the distributions results in an increase in observed 0 values after concept drift compared to before concept drift for a specific class. However, this is similar to the regular error rate and therefore not faster in detecting concept drift.

Instead of correctly classified instances, we started to focus on the distance towards the decision boundary for misclassified instances. We will explain this approach in the next section.

4.2 Lower Bound to the Critical ε-drift Value



(a) No concept drift.

(b) Concept drift.

Figure 13: Distances towards the decision boundary for misclassified instances in the presence of no concept drift (Figure 13a) and concept drift (Figure 13b).

A noticeable difference between Figures 6a and 6b is that several misclassified instances in Figure 6b are more distant from the decision boundary than misclassified instances in Figure 6a. In more detail, the two orange instances between the red line (separator) and the green line (decision boundary) are the only misclassified instances in Figure 6a. The distances towards the decision boundary of both instances are relatively small since the area between the red and green lines is small. On the other hand, we observe an increase in misclassified instances in Figure 6b due to concept drift. In other words, the area between the red line (separator) and green line (decision boundary) increased and hence we observe misclassified instances with an increased distance towards the decision boundary compared to Figure 6a. These increased distances towards the decision boundary for misclassified instances function as a sign of concept drift.

We introduce the ε-drift* value inspired by Bosman et al. [25] to compute the distance towards the decision boundary for a misclassified instance. Instead of searching for the $\tilde{\varepsilon}^*$ value of a correctly classified non-perturbed instance $\lambda(x_0)$, we are interested in doing the opposite. Namely, we aim to find the ε-drift* value for a misclassified non-perturbed instance.

Formally, we utilise a static trained neural network f_θ in $\mathbb{R}^n \rightarrow \mathbb{R}^m$ to make predictions about newly incoming instances in which θ represents the trainable parameters of the neural network and n and m stand for the input and output dimensions respectively. For a misclassified non-perturbed instance represented as x_0, a perturbed instance x within the region $x \in \{x : \|x - x_0\|_p \leq \varepsilon\text{-drift}^*\}$ triggers the neural network to predict the same class as x_0. Through discretizing the search space of possible ε-drift values (i.e., $\mathcal{E} = \{\varepsilon\text{-drift}_1, ..., \varepsilon\text{-drift}_n\}$, in which $\varepsilon\text{-drift}_i < \varepsilon\text{-drift}_{i+1}, \forall i \in \{1, ..., n\}$), we are interested in finding two ε-drift values such that the smaller one ε-drift$_{i-1}$ triggers the network to predict the exact same class as x_0, whereas the bigger one ε-drift$_i$ predicts the opposite class of x_0. If such a pair exists, we interpret ε-drift$_{i-1}$ as the lower bound to the critical ε-drift value $\tilde{\varepsilon}^*$-drift. In other words, $\tilde{\varepsilon}^*$-drift represents the distance towards the decision boundary for a

misclassified non-perturbed instance x_0. Compared to the regular error rate-based methods, our $\tilde{\varepsilon}^*$-drift method provides additional information in the form of the distance towards the decision boundary. In the next section 4.3, we will give a high-level overview of our pipeline.

4.3 Concept Drift Detection Through $\tilde{\varepsilon}^*$-drift Values

We will address an important part of our concept drift detection method. Specifically, we will cover the binary search for $\tilde{\varepsilon}^*$-drift values.

4.3.1 Binary Search for $\tilde{\varepsilon}^*$-drift Values

To search for the $\tilde{\varepsilon}^*$-drift value of a misclassified instance, we make use of a binary search approach as performed in the study of Bosman et al. [25]. Ideally, we aim to find a pair of ε-drift values such that the smaller ε-drift$_{i-1}$ is unsatisfiable (safe) and the larger ε-drift$_i$ is satisfiable (unsafe). However, we also account for the following scenarios: i) All ε-drift values in $\mathcal{E}$ might be satisfiable meaning there is no safe ε-drift value. This is expected to happen, especially when there is no concept drift, for misclassified instances that lie very close to the decision boundary. Consequently, we set the $\tilde{\varepsilon}^*$-drift value equal to 0 for such cases. ii) It is also possible that all ε-drift values are unsatisfiable for instances with a considerable distance from the decision boundary. In that specific case, we assign the highest ε-drift value in $\mathcal{E}$ to be the $\tilde{\varepsilon}^*$-drift.

Finally, a complete neural network verifier such as α-β-CROWN, is guaranteed to give a definite proof of a property given enough time [27]. Since time is precious in a data streaming context, we need to restrict the time a neural network verifier gets to verify if a property holds. As a consequence, the output of a neural network verifier can be timeout. Another possible outcome is an out-of-memory error. This raises other possibilities regarding the outcome of a $\tilde{\varepsilon}^*$-drift value for a misclassified instance.

Namely, we might find timeout or out-of-memory error outputs in between a safe and unsafe ε-drift value. As a result, the range defined by the safe and unsafe ε-drift values in which we can find the ε^*-drift becomes larger. In such cases, we still consider the safe ε-drift value as the $\tilde{\varepsilon}^*$-drift. Although the distance towards the decision boundary becomes less accurate, we do not overestimate this distance preventing potential false positives. We perform the same procedure if we encounter a safe ε-drift value and all larger ε-drift values evaluate to timeout or out-of-memory error outputs.

Finally, we might also encounter an opposite situation in which we find an unsafe ε-drift and all smaller ε-drift values to be timeouts or out-of-memory errors. In the worst-case scenario, all ε-drift values will result in a timeout or out-of-memory error output. In these cases, we have insufficient information to determine the distance towards the decision boundary and hence we discard the instance.

4.4 Concept Drift Detection Methods

Through the computed $\tilde{\varepsilon}^*$-drift values, we aim to detect concept drift. We use these values directly as input for the classical drift detection methods DDM, ADWIN and KSWIN mentioned in Table 1. Additionally, our intuition about $\tilde{\varepsilon}^*$-drift values is that we observe small $\tilde{\varepsilon}^*$-drift values with no concept drift while these values increase in the presence of concept drift. For this reason, we introduce two additional concept drift detectors: the Mann-Whitney U test [40] and a threshold method.

The Mann-Whithney U test is a non-parametric test meaning that the input data does not need to be of a specific distribution such as a Gaussian distribution. Since we do not have information

about the distribution of $\tilde{\varepsilon}^*$-drift values, we opted for this test. Intuitively, the Mann-Whitney U test compares the rankings of two independent populations. It assesses whether the values of one population tend to be larger compared to the other population. This aligns with our proposed method to detect concept drift through $\tilde{\varepsilon}^*$-drift values. Namely, we expect to observe increased $\tilde{\varepsilon}^*$-drift values with concept drift.

As for the practical implementation, we keep in memory a part of the training phase $\tilde{\varepsilon}^*$-drift values. Given that the input data's normalisation procedure might cause misclassifications due to incorrect mapping for the first batch of instances, we consider the most recent instances in the training set. With the arrival of new instances, we first fill a second window until it contains the same number of $\tilde{\varepsilon}^*$-drift values as the reference window. As soon as this is the case, we conduct the Mann-Whitney U test. If indeed $\tilde{\varepsilon}^*$-drift values from the second window are statistically larger than the ones from the reference window, we consider it as an indication of concept drift. Otherwise, we discard the oldest instance in the second window, include the most recent $\tilde{\varepsilon}^*$-drift value and repeat the described procedure.

A final approach we propose is the utilisation of a so-called threshold method. After the training phase, we could evaluate the range of $\tilde{\varepsilon}^*$-drift values. Based on these observations, we can establish criteria for acceptable $\tilde{\varepsilon}^*$-drift values. For instance, by observing enough $\tilde{\varepsilon}^*$-drift values during the training phase, we can compute the probability of observing a certain $\tilde{\varepsilon}^*$-drift value. If this probability is smaller than 0.05, we could consider it concept drift. Another possibility would be to compute the mean and standard deviation of the distribution of $\tilde{\varepsilon}^*$-drift values during the training phase. If the current $\tilde{\varepsilon}^*$-drift value is for example three standard deviations away from the mean, we could consider it concept drift.

However, these methods assume that we observe enough $\tilde{\varepsilon}^*$-drift values which is not guaranteed. Therefore, a less robust method not grounded in statistical evidence is to consider an acceptable range of $\tilde{\varepsilon}^*$-drift values. One approach could be to take the observed $\tilde{\varepsilon}^*$-drift values as this acceptable range. Concretely, if a $\tilde{\varepsilon}^*$-drift value surpasses the acceptable range, we consider it concept drift.

It is important to note that the threshold method is only compatible with $\tilde{\varepsilon}^*$-drift values. Namely, with the error rate consisting of binary values, we cannot make claims about observing larger values or values falling outside a certain range of observations. Regarding the features as input, we assume that the input distributions before and after concept drift do not change. Furthermore, the threshold method directly influences the method's sensitivity. A higher threshold means a lower number of false positives at the risk of delays or missing concept drift. Therefore, it needs further refinement, which we leave open for future work. Despite its current limitations, we believe this approach introduces a promising new research direction within the domain of neural network verification, briefly discussed in Section 6.

5 Experiments

In this section, we will explain our experimental setup and demonstrate the obtained results. To verify if our proposed method can handle different types of concept drift, we assessed the detection capabilities on two synthetic data generators. A general overview of the used synthetic data generators is present in Table 2. The code to reproduce the experiments is available via the following link: https://github.com/rlaanen/e_drift.

5.1 Synthetic Data Generators

There is a general distinction between synthetic data generators and real-world datasets in the field of concept drift [9]. Synthetic data generators provide annotations regarding the type of concept drift and the point in time at which the concept drift occurs. On the contrary, real-world datasets often lack such annotations limiting the evaluation of concept drift detection methods. Additionally, Souza et al. [9] argue whether real-world datasets truly exhibit concept drift. Namely, it is oftentimes assumed that a decrease in model performance over time indicates that a real-world dataset contains concept drift [9].

However, synthetic data generators often do not reflect a real-world dataset's complexity. Another disadvantage of synthetic data generators is the risk of committing data bias [9]. For instance, one can cherry-pick a certain configuration of a synthetic data generator, reflecting a favourable outcome for a drift detector.

Since we are interested in assessing whether our concept drift detection method is faster in identifying concept drift compared to classical error rate-based methods, it is crucial to have precise information regarding the point in time and type of concept drift. Therefore, we decided to benchmark the detection capabilities of our method on two synthetic data generators. Specifically, we utilised the SEA generator to simulate sudden drift and the Hyperplane generator to mimic incremental drift. We generated the synthetic data streams using the free and open-source River [41] framework. The upcoming subsections will provide a detailed description of the two synthetic data generators.

5.1.1 SEA

SEA [17] is a data stream generator capable of simulating sudden drift. Each instance consists of three real-valued continuous features $att1$, $att2$ and $att3$ in the range $[0, 10]$ and is linked to a binary label (i.e., False or True). Moreover, if the sum of the first two features exceeds a certain threshold (i.e., $att1 + att2 > \delta$) the instance evaluates to True and otherwise to False. SEA contains the following four thresholds: variant 0: $\delta = 8$, variant 1: $\delta = 9$, variant 2: $\delta = 7$ and variant 4: $\delta = 9.5$. Sudden drift can be simulated by changing the threshold δ. For instance, the labels of the first 1,000 instances are determined by $\delta = 8$ and the labels of the next 1,000 instances by $\delta = 9$. This abrupt change in δ simulates sudden drift. A summarised description of the SEA dataset is present in Table 2.

5.1.2 Hyperplane

The Hyperplane generator [42] initialises a hyperplane in d-dimensional space according to the following Equation 4:

$$\sum_{i=1}^{d} w_i x_i = w_0 = \sum_{i=1}^{d} w_i \tag{4}$$

in which w represents the coefficients of the hyperplane [42]. By initialising the Hyperplane generator, a random set of weights w_0 with a size equal to d is generated. Per timestep, the Hyperplane generator samples a random instance of size d with values ranging between [0,1]. If the sum of the set of weights times the instance is bigger than w_0 (see Equation 5) the associated label is positive. On the contrary, if the sum of the weights times the instance is smaller or equal to w_0 (see Equation 6), the associated label is negative.

$$\sum_{i=1}^{d} w_i x_i > w_0 \tag{5}$$

$$\sum_{i=1}^{d} w_i x_i \le w_0 \tag{6}$$

To introduce incremental drift, we can alter the original set of weights w_0 of the hyperplane. Specifically, we can increase or decrease the size of each weight in w_0 to change the position and orientation of the hyperplane [42]. The River framework [41] allows the user to set the size of the change $\nu \in [0,1]$ and the probability that the direction of change is reversed $\zeta \in [0,1]$. With these two hyperparameters, each weight in w_0 can be adjusted as follows: $w_i = w_i + \zeta\nu$ [41]. Table 2 contains summarised information about the hyperplane dataset.

Table 2: Overview of the synthetic data generators used in our experiments. Table adapted from [7].

Dataset	Drift Type	Features	Classes	Source
SEA	Sudden	3	2	River
Hyperplane	Incremental	10	2	River

5.2 Evaluation Methods

Assessing the performance of concept drift detection methods is a challenging task due to the different properties of synthetic data generators and real-world datasets. Therefore, it is important to underline whether a metric is appropriate for the given dataset. This is specifically the case for real-world datasets because they often lack annotations for exact timestamps at which concept drift occurs and for the specific type of concept drift [9, 20].

As mentioned in Section 2.1.2, concept drift detection methods might suffer from false positives. False positives are undesirable since unnecessary retraining of a model comes at the cost of resources. Thereby, we are interested in knowing if a detection strategy is capable of detecting a concept drift in a reasonable timeframe. For these reasons, the mean time to detection (MTD), the false alarm count (FAC) and the missed detection count (MDC) [43, 20] are important metrics in assessing the effectiveness of a detection mechanism. These metrics are only applicable to data streams with annotations about concept drift.

MTD represents the number of instances between the start position of the concept drift and the detected point. The lower the MTD value the better the detection mechanism is in detecting drifts early. FAC denotes the number of times the drift detection mechanism flags drift before the actual starting point of the concept drift. A lower FAC is desirable since it would save resources. MDC indicates the number of times the detection mechanism fails to detect concept drift, where a lower MDC, ideally 0, signifies a better-performing drift detector.

5.3 Experimental Settings

In this section, we outline the experimental setup. First, we will describe how we generated synthetic datasets. Second, we will provide details about the training procedures employed for the neural networks used in our experiments. Third, we will briefly mention the configuration of the α,β-CROWN neural network verifier. Finally, we will specify the parameters used for the concept drift detection methods.

5.3.1 Dataset Settings

To benchmark our proposed method, we utilised the SEA generator to mimic sudden drift and the Hyperplane generator to simulate incremental drift. Each synthetic dataset that we generated in the context of this research consists of 100,000 instances. The first 5,000 instances originate from another context than the remaining 5,000 instances. In other words, concept drift starts from instance 5,000. Furthermore, we decided not to inject noise into the synthetic data streams.

Regarding the SEA generator, we generated all possible combinations of variants resulting in a total of 6 data stream configurations. This means that the first context is another variant than the second context. For each possible combination, we generated five data stream instances with seeds 100-104 for the first context and seeds 105-109 for the second context.

As for the Hyperplane generator, the number of features as well as the number of drift features can be set by the user. We opted for the default values of the River framework which means 10 features and 2 drift features. The most important parameters to simulate drift are the size of the change v and the probability that the direction of change is reversed ζ. For the first context, both parameters are set equal to zero. Following Raab et al. [1] for the second context, we set $v = 0.001$. Since ζ influences the direction of change, we decided to set this parameter to zero for the second context as well to ensure incremental drift in one direction. Similarly to the SEA data stream configurations, we generated five instances of the data stream configuration with seeds 105-109. Detailed information about the specific configurations can be found in Appendix B.

5.3.2 Normalization of the Input Data

The majority of the work in the field of neural network verification focuses exclusively on image datasets such as MNIST and CIFAR [24]. Image data is typically mapped from [0,255] to [0,1]. Given that we did not use image data, we normalised real-valued features that are not in the range [0,1] to the range [0,1] following [44, 45]. Normalisation also ensures we can use a consistent set of ε-drift values across multiple datasets [44] to compute the distance towards the decision boundary for misclassified instances. Equation 7 demonstrates the min-max normalization which maps a single feature to the range [0,1]:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{7}$$

in which $x_{\min}$ represents the minimum value for a specific feature and $x_{\max}$ denotes the maximum value for a specific feature based on a set of instances. Namely, min-max normalization transforms a specific feature by setting the minimum value equal to 0, the maximum value to 1 and the remaining values between 0 and 1. In a setting with a static dataset, these values are trivial to compute for each feature.

However, an online setting does not allow instant access to the statistics of the evolving dataset [10]. Therefore, we adopt the min-max normalization function from the River [41] framework.

During the training phase of an algorithm, we consider a static dataset from which we compute the min and max values. We maintain and actively update the min and max values as new instances are incoming. One drawback of the River min-max normalization function lies in its inability to allow for batch updates. Consequently, the first few instances are incorrectly mapped to the [0,1] range, leading to incorrect patterns between inputs and targets. Practically, we normalised all generated SEA datasets.

5.3.3 Neural Network Configurations

Given that our proposed drift detection method relies on neural network verification, we exclusively utilised neural networks as prediction models. As for the architecture of the neural networks, we restricted ourselves to fully connected neural networks with ReLU as an activation function for the nodes. Furthermore, the neural networks and the training procedure exhibit various hyperparameters to tune such as the batch size, number of epochs, learning rate and optimizer. We performed a grid search to tune these hyperparameters. Specifically, we assessed the following values for the batch size: $\{16, 32, 64, 128\}$, the following values for the number of epochs: $\{5, 10, 25, 50, 100\}$, the following values for the learning rate $\{0.01, 0.005, 0.001, 0.0005, 0.0001\}$ and the following optimizers: {SGD, RMSprop and Adam}. To determine the best combination of hyperparameters, we combined the grid search with a 5-fold cross-validation.

Hence, on each dataset, we performed a grid search together with a 5-fold cross-validation strategy on the training set consisting of 2,000 instances with the Matthews Correlation Coefficient (MCC) [46] as an evaluation metric to assess combinations of hyperparameter values. MCC is mathematically defined in Equation 8 retrieved from [46]:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \tag{8}$$

MCC reflects a score in the interval [-1,1] in which 1 represents a perfect classifier, 0 denotes a no-skill classifier (i.e., random guessing the class label) and -1 stands for a perfect misclassified. According to Chicco and Giuseppe [46], MCC provides a more reliable score than accuracy and F1 measures since it is unaffected by class swapping and can handle imbalanced datasets. We utilised the hyperparameter values of the combination yielding the highest MCC score. The exact neural network architectures including the best-found combination of hyperparameters per dataset can be found in Appendix C.

5.3.4 α,β-CROWN Settings

To compute the $\tilde{\varepsilon}^*$-drift values within the range of 0.001 to 0.4 at intervals of 0.002 [25], we utilised the state-of-the-art GPU restricted neural network verifier α,β-CROWN [27]. The GitHub page[1] provides an extensive guide on the configuration of α,β-CROWN. Given that our neural networks consist of a maximum of 24 neurons (see Appendix C), we first opted for alpha-CROWN in combination with a MIP solver. Unfortunately, this configuration provided inconsistent outcomes based on preliminary experiments. As a next step, we tried GCP-CROWN also yielding inconsistent results on preliminary experiments. Finally, we opted for the configuration beta-CROWN with ReLU branching which provided expected results based on preliminary experiments. Specifically, we followed the configuration based on the custom model data example in combination with the configuration applied in the work of Bosman et al. [25]. Since limited to no information is given

[1] https://github.com/Verified-Intelligence/alpha-beta-CROWN/blob/main/complete_verifier/docs

about the configuration parameters by the maintainers of the α,β-CROWN GitHub page, we cannot go into detail about the parameters of the configuration. However, we decided to set a time limit of 600 seconds per ε value. The configuration is present in Appendix D.

5.3.5 Drift Detector Settings

In our experiments, we utilised the following drift detection mechanisms: DDM, EDDM, ADWIN, KSWIN, Mann-Whitney U test and Threshold. Each of these have specific parameters. For ADWIN we utilised the default parameters provided by the River framework. Since the default parameters of the River framework for KSWIN result in non-deterministic behaviour, we decided to adopt a window size of 200 and a statistic size of 100 as used in the work of Baier et al. [20]. However, Baier et al. [20] tuned the test statistic value of KSWIN on a validation set if it contains concept drift and otherwise opted for the default value. From the perspective of real-world datasets, this procedure is not solid since it is not guaranteed that part of the dataset contains concept drift. For this reason, we decided to set the test statistic equal to the default value of the River framework which is 0.005. Similarly, for the Mann-Whitney U test, we applied a reference window of size 100, a detection window of size 100 and a test statistic of 0.005.

As can be seen in Appendix E, we did not observe enough misclassifications in all datasets to set up one of the two statistical threshold methods proposed in Section 4.4. Therefore, we used the non-statistical approach and defined the threshold to be 0.015 based on preliminary experiments and to ensure consistency across the datasets. This implies that any $\tilde{\varepsilon}^*$-drift value exceeding this threshold indicates the presence of concept drift. note that the threshold method is a special case and requires further investigation to provide robust outcomes.

5.4 Results

In this section, we present the obtained results through our conducted experiments. We show the results for synthetic datasets since these allow us to compare the detection capabilities of a variety of inputs with concept drift detectors. Given that we assessed six combinations for the SEA generator (i.e., all possible combinations of the four thresholds), we present a subset of the obtained results representative of the overall findings for this synthetic data generator. The other set of results can be found in Appendix A.

5.4.1 Synthethic Datasets Results

Our objective for the synthetic datasets is to detect concept drift within a reasonable timeframe. To this end, we defined an acceptable delay window starting from the moment concept drift occurs. This means that a drift detector must identify concept drift within this delay window, otherwise, we consider it a missed detection. For both the SEA and Hyperplane datasets, we set this acceptable delay window to 1,000 [11].

Furthermore, we utilised three input sources: error rate, features and $\tilde{\varepsilon}^*$-drift values. The error rate for a single instance was obtained by performing a single forward pass through a neural network to generate the predicted target. A value of 0 was assigned for correct predictions, while a value of 1 was assigned for incorrect predictions. These binary values were then used as inputs to a drift detector. Regarding the computation of $\tilde{\varepsilon}^*$-drift values, we differentiated between correctly and incorrectly classified instances. Similar to the error rate, we assigned a value of 0 to correctly classified instances. For misclassifications, the $\tilde{\varepsilon}^*$-drift value was computed. We used the 0's for correctly classified instances and the $\tilde{\varepsilon}^*$-drift values for misclassifications as inputs to a drift detector.

Given that we used the complete neural network verifier α,β-CROWN to compute $\tilde{\varepsilon}^*$-drift values, we risked obtaining timeouts or out-of-memory errors. Fortunately, for both the SEA and Hyperplane datasets, we did not encounter timeouts or out-of-memory errors as presented in Appendix E.

Regarding the drift detectors, we applied DDM, ADWIN KSWIN and Mann-Whitney U test in combination with the three inputs. The implementation of EDDM in River explicitly accepts zeros and ones as input. For this reason, we only combined the error rate with EDDM. Additionally, we included the threshold drift detection method restricted to $\tilde{\varepsilon}^*$-drift values as input. Despite the current limitations of this drift detector, we think the preliminary results offer a valuable comparison to other combinations of inputs with drift detectors.

Table 3: Mean time to detection (MTD), false alarm count (FAC) and missed detection count (MCD) for different input with drift detector combinations on 5 seeded configurations of the SEA dataset with variants 0 (8) and 1 (9). Mann-Whitney U test is abbreviated as MWU. A dash represents that the input with drift detector combination was unable to flag concept drift within the acceptable delay window. A cross indicates that the input is not compatible with the drift detector. Bold and underlined entries represent the fastest and second-fastest combinations respectively. An asterisk stands for the fastest combination with 0 FAC and 0 MCD.

Drift Detector	MTD			FAC			MDC		
	error rate	features	$\tilde{\varepsilon}^*$-drift	error rate	features	$\tilde{\varepsilon}^*$-drift	error-rate	features	$\tilde{\varepsilon}^*$-drift
DDM	**37±27**	-	<u>57±89</u>	3±4	-	2±2	0	-	0
EDDM	239±52	x	x	1±1	x	x	0	x	x
ADWIN	394±80	-	-	0	-	-	0	-	-
KSWIN	-	445±295	-	-	3±3	-	-	1	-
MWU	167±131	511±82	165±132*	0	22±16	0	0	2	0
threshold	x	x	15±14	x	x	0	x	x	0

Table 3 denotes the results on the SEA dataset with variants 0 (8) and 1 (9). Based on this table, the fastest drift detector in combination with error rate is DDM, with an MTD of 37. When combined with features, KSWIN is the fastest, with an MTD of 445. For the combination with $\tilde{\varepsilon}^*$-drift DDM is again the fastest with an MTD of 57. Overall, error rate with DDM as a drift detector yields the fastest MTD. However, $\tilde{\varepsilon}^*$-drift with DDM exhibits a slightly lower FAC. The fastest combination with no false positives or missed detections is $\tilde{\varepsilon}^*$-drift with Mann-Whitney U test with an MTD of 165. Notably, the combination of $\tilde{\varepsilon}^*$-drift with the threshold drift detector achieves the fastest combination with an MTD of 15 and no false positives or missed detections.

Table 4: Mean time to detection (MTD), false alarm count (FAC) and missed detection count (MCD) for different input with drift detector combinations on 5 seeded configurations of the SEA dataset with variants 1 (9) and 3 (9.5). Mann-Whitney U test is abbreviated as MWU. A dash represents that the input with drift detector combination was unable to flag concept drift within the acceptable delay window. A cross indicates that the input is not compatible with the drift detector. Bold and underlined entries represent the fastest and second-fastest combinations respectively. An asterisk stands for the fastest combination with 0 FAC and 0 MCD.

Drift Detector	MTD			FAC			MDC		
	error rate	features	$\tilde{\varepsilon}^*$-drift	error rate	features	$\tilde{\varepsilon}^*$-drift	error-rate	features	$\tilde{\varepsilon}^*$-drift
DDM	67±57	-	**29±30**	7±3	-	5±3	0	-	0
EDDM	379±89*	x	x	0	x	x	0	x	x
ADWIN	707±180	-	-	0	-	-	1	-	-
KSWIN	-	445±295	-	-	3±3	-	-	1	-
MWU	355±159	511±82	353±153	0	22±16	0	1	2	1
threshold	x	x	65±56	x	x	0	x	x	0

Table 4 contains the results on the SEA dataset with variants 1 (9) and 3 (9.5). As becomes clear from this table, the fastest drift detector using error rate is DDM with an MTD of 67. When combined with features, KSWIN is the fastest, achieving an MTD of 445. For the combination with $\tilde{\varepsilon}^*$-drift DDM is against the fastest with an MTD 29. Thus, $\tilde{\varepsilon}^*$-drift with DDM exhibits the fastest MTD. Error rate with EDDM is the fastest combination with an MTD of 379 yielding 0 FAC and 0 MDC. As for the combination of $\tilde{\varepsilon}^*$-drift with the threshold drift detector, it obtains an MTD of 65 with no false positives or missed detections.

Table 5: Mean time to detection (MTD), false alarm count (FAC) and missed detection count (MCD) for different input with drift detector combinations on 5 seeded configurations of the SEA dataset with variants 2 (7) and 3 (9.5). Mann-Whitney U test is abbreviated as MWU. A dash represents that the input with drift detector combination was unable to flag concept drift within the acceptable delay window. A cross indicates that the input is not compatible with the drift detector. Bold and underlined entries represent the fastest and second-fastest combinations respectively. An asterisk stands for the fastest combination with 0 FAC and 0 MCD.

Drift Detector	MTD			FAC			MDC		
	error rate	features	$\tilde{\varepsilon}^*$-drift	error rate	features	$\tilde{\varepsilon}^*$-drift	error-rate	features	$\tilde{\varepsilon}^*$-drift
DDM	18±24	-	**10±15**	5±2	-	3±2	0	-	0
EDDM	121±22*	x	x	1±1	x	x	0	x	x
ADWIN	183±48	-	876	0	-	0	0	-	4
KSWIN	-	445±295	-	-	3±3	-	-	1	-
MWU	32±10*	511±82	32±10*	0	22±16	0	0	2	0
threshold	x	x	2±3	x	x	0	x	x	0

Table 5 represents the results on the SEA dataset with variants 2 (7) and 3 (9.5). From this table, it is evident that the fastest drift detector in combination with error rate is DDM, with an MTD of 18. When combined with features, KSWIN is the fastest, with an MTD of 445. For the combination with $\tilde{\varepsilon}^*$-drift DDM is again the fastest with an MTD of 10. Consequently, $\tilde{\varepsilon}^*$-drift with DDM is the fastest combination overall. However, the combinations of error rate and $\tilde{\varepsilon}^*$-drift both with Mann-Whitney U test, with an MTD of 32, are the fastest that exhibit no false positives or missed detections. Notably, the combination of $\tilde{\varepsilon}^*$-drift with the threshold drift detector achieves the fastest combination with an MTD of 2 and also maintains no false positives or missed detections.

Table 6: Mean time to detection (MTD), false alarm count (FAC) and missed detection count (MCD) for different input with drift detector combinations on 5 seeded configurations of the Hyperplane dataset with a size of change v of 0.001 to simulate incremental drift. Mann-Whitney U test is abbreviated as MWU. A dash represents that the input with drift detector combination was unable to flag concept drift within the acceptable time window. A cross indicates that the input is not compatible with the drift detector. Bold and underlined entries represent the fastest and second-fastest combinations respectively. An asterisk stands for the fastest combination with 0 FAC and 0 MCD.

Drift Detector	MTD			FAC			MDC		
	error rate	features	$\tilde{\varepsilon}^*$-drift	error rate	features	$\tilde{\varepsilon}^*$-drift	error-rate	features	$\tilde{\varepsilon}^*$-drift
DDM	382±238	-	357±130	6±5	-	4±4	0	-	0
EDDM	<u>309±173</u>	x	x	1±1	x	x	0	x	x
ADWIN	720±159	-	-	0	-	-	0	-	-
KSWIN	-	**222±127**	-	-	11±2	-	-	1	-
MWU	387±112	367±159	374±94*	0	110±84	0	0	1	0
threshold	x	x	209±43	x	x	0	x	x	0

Table 6 depicts the results on the Hyperplane dataset with a size of change of 0.001. Based on this table, the fastest drift detector with error rate is EDDM, with an MTD of 309. When combined with features, KSWIN is the fastest, with an MTD of 222. For the combination with $\tilde{\varepsilon}^*$-drift DDM is again the fastest with an MTD of 357. Hence, features in combination with KSWIN is the fastest combination. However, this combination yields a higher FAC and contains one missed detection compared to the second fastest combination consisting of error rate with EDDM. The combination of $\tilde{\varepsilon}^*$-drift with Mann-Whitney U test, with an MTD of 374, is the fastest that has no false positives or missed detections. Notably, the combination of $\tilde{\varepsilon}^*$-drift with the threshold drift detector achieves the fastest combination with an MTD of 209 and also maintains no false positives or missed detections.

5.4.2 Running Times on Synthetic Datasets

Another interesting aspect is the running time required to obtain the three inputs. Among these, features demand no computational cost since they are immediately available with the arrival of a new instance. The error rate requires a single forward pass through a neural network, followed by a verification step to determine if the predicted target matches the ground truth. Computing the $\tilde{\varepsilon}^*$-drift value for a misclassified instance involves performing a binary search via α,β-CROWN, which is computationally expensive due to the NP-hard nature of the problem.

Specifically, we analysed the wallclock time in seconds for both the error rate and $\tilde{\varepsilon}^*$-drift values for misclassified instances across all SEA and Hyperplane datasets. For the error rate, the wallclock time reflects the time it takes to predict an instance. For the $\tilde{\varepsilon}^*$-drift value, the wallclock time describes the time it takes to compute the $\tilde{\varepsilon}^*$-drift value for a misclassified instance. Given that the six SEA dataset configurations utilise the same neural network architecture, which determines the complexity, we decided to combine the wallclock time results. The minimum, maximum and mean wallclock times are presented in Table 7.

Table 7: Minimum, maximum and mean wallclock times in milliseconds (ms) to compute a single error rate value and in seconds (s) to compute a single $\tilde{\varepsilon}^*$-drift value. SEA represents all measured wallclock times over the six different SEA configurations.

Dataset	error rate			$\tilde{\varepsilon}^*$-drift		
	Min [ms]	Max [ms]	Mean [ms]	Min [s]	Max [s]	Mean [s]
SEA	0.05	15.67	0.10	48.20	93.79	60.50
Hyperplane	0.05	15.78	0.14	52.54	2021.86	166.85

As becomes clear from Table 7, computing the error rate is on average 100,000 times cheaper than computing $\tilde{\varepsilon}^*$-drift values. Furthermore, for both the error rate and $\tilde{\varepsilon}^*$-drift values, computing values is more expensive for the Hyperplane dataset compared to the SEA datasets.

5.4.3 Discussion

To summarise, $\tilde{\varepsilon}^*$-drift with DDM achieved the fastest drift detection performance in four out of six cases for the SEA datasets simulating sudden drift. The second fastest combination was error rate with DDM. Additionally, $\tilde{\varepsilon}^*$-drift with Mann-Whitney U test was the fastest combination with no false positives and no missed detections in four out of six cases. As for the detection capabilities of $\tilde{\varepsilon}^*$-drift with DDM and Mann-Whitney U test, both combinations obtain a similar MDC as error rate combined with these two drift detection methods.

Regarding the incremental drift mimicked through the Hyperplane datasets, the fastest combination was features with KSWIN. However, the second fastest combination consisting of error rate with EDDM yielded a lower FAC and no missed detections. This reasonably fast MTD with low FAC and no missed detections aligns with expectations, as EDDM is explicitly designed to handle incremental drift. It would be interesting to adapt EDDM to handle $\tilde{\varepsilon}^*$-drift values as input, which we leave open for future work. Similarly to sudden drift, $\tilde{\varepsilon}^*$-drift in combination with the Mann-Whitney U test is slower in terms of drift detection, but resulted in no false positives or missed detections. As for the detection capabilities of $\tilde{\varepsilon}^*$-drift with DDM and Mann-Whitney U test, both combinations obtain a similar MDC as error rate combined with these two drift detection methods.

These results indicate that $\tilde{\varepsilon}^*$-drift values effectively detect concept drift. A notable disadvantage of $\tilde{\varepsilon}^*$-drift values compared to the error rate is the higher computational cost. Specifically, $\tilde{\varepsilon}^*$-drift values are on average 100,000 times more computationally expensive than the error rate in terms of wallclock time. However, this could be reduced by improving the computational efficiency of α,β-CROWN. In settings with a low frequency of incoming instances such as weather stations or the electricity market, this difference in computational cost may be negligible.

An additional advantage of $\tilde{\varepsilon}^*$-drift with DDM over error rate with DDM is the reduced number of false positives. In the Hyperplane datasets and all six cases of the SEA datasets, $\tilde{\varepsilon}^*$-drift with DDM achieved a lower average number of false positives. Additionally, $\tilde{\varepsilon}^*$-drift with the Mann-Whitney U test was the fastest combination with no false positives and no missed detections for the Hyperplane datasets and four out of six cases of the SEA datasets

Interestingly, for both sudden drift and incremental drift, $\tilde{\varepsilon}^*$-drift with the threshold drift detection achieved a faster MTD compared to the other combinations except for SEA with variants 1 (9) and 3 (9.5). This might be because this specific configuration of the SEA generator contains less severe drift compared to the other possible combinations. Namely, the shift from 9 to 9.5 encompasses a small region in which misclassifications occur. Therefore, the chance to observe misclassifications larger than the predefined threshold becomes smaller or might even be impossible. while these findings are promising, further assessment of the threshold method is needed to provide

conclusive statements.

5.4.4 Limitations

The conducted experiments demonstrated that the detection of concept drift through $\tilde{\varepsilon}^*$-drift values is effective and can result in a lower average number of false positives compared to the regular error rate. However, several limitations are associated with using $\tilde{\varepsilon}^*$-drift values.

First, this method relies on the complete neural network verifier α,β-CROWN. Given that α,β-CROWN is unable to handle regression tasks, the current method is restricted to classification tasks. Furthermore, while α,β-CROWN guarantees an exact solution given sufficient time and no out-of-memory errors, verifying such properties is considered an NP-hard problem. In practice, this makes computing $\tilde{\varepsilon}^*$-drift values computationally expensive compared to using the error rate or features. Since certain data streams might require real-time processing of incoming data [10], the current latency of α,β-CROWN could limit its application in such settings.

Second, error rate-based methods such as the $\tilde{\varepsilon}^*$-drift values generally require access to the ground truth. However, immediate access to labels is unrealistic in real-world data stream scenarios due to delays and expenses involved in acquiring true labels [47]. Without access to the ground truth, it is impossible to compute $\tilde{\varepsilon}^*$-drift values.

Third, Raab et al. [1] and Baier et al. [20] introduced a certain percentage of noise in the synthetic data generators. Specifically, the noise involved in the data generators of the River framework randomly swaps class labels. This type of noise causes our $\tilde{\varepsilon}^*$-drift detector to malfunction. Suppose that a correctly classified instance with a considerable distance from the decision boundary undergoes a class swap due to noise. As a consequence, it will be misclassified by the neural network and receive a large $\tilde{\varepsilon}^*$-drift value. In such a case, the drift detector might accidentally identify concept drift whereas there is no actual concept drift. In other words, this form of noise introduces a potential increase in false positives.

Finally, for our method to work, we assumed that misclassifications only happen to instances close to the decision boundary. In real-world scenarios, this is an oversimplistic assumption. Namely, it might be the case that the coverage of the training instances does not capture the entire space. As a result, the neural network learns a suboptimal decision boundary and behaves uncertainly in specific regions. In practice, instances from underrepresented regions may appear during model deployment potentially causing increased $\tilde{\varepsilon}^*$-drift values. Again, the observation of increased $\tilde{\varepsilon}^*$-drift values might trigger the drift detector to incorrectly identify concept drift, increasing the risk of false positives.

6 Conclusion and Future Work

In this thesis, we introduced the $\tilde{\varepsilon}^*$-drift value. The $\tilde{\varepsilon}^*$-drift value represents an approximation of the distance towards the decision boundary for misclassified instances. Through a hypothetical scenario based on a simplification of the SEA generator simulating sudden drift, we observed that these distances increase in the presence of concept drift. Therefore, we decided to utilise $\tilde{\varepsilon}^*$-drift values as input to various drift detectors to detect concept drift. Essentially, $\tilde{\varepsilon}^*$-drift values are an extension of regular error rate-based methods (i.e., binary output) by providing additional information in the form of the distance to the decision boundary. We computed the $\tilde{\varepsilon}^*$-drift values values for misclassified instances via the complete neural network verifier α,β-CROWN.

Regarding the conducted experiments, we benchmarked our proposed $\tilde{\varepsilon}^*$-drift values against two other input sources: error rate and features. In our experiments, we utilised four classical drift detectors: DDM, EDDM, ADWIN and KSWIN and included the non-parametric Mann-Whitney U test. In addition, we decided to test a preliminary threshold method exclusively accepting $\tilde{\varepsilon}^*$-drift values as input. This method flags concept drift as soon as a $\tilde{\varepsilon}^*$-drift value larger than the threshold is observed. As for the datasets, we used two synthetic data generators: SEA simulating sudden drift and Hyperplane simulating incremental drift. We assessed the input with drift detector combinations on three metrics: the mean time to detection (MTD) representing the delay in detecting drift in terms of the number of instances, the false alarm count (FAC) denoting the number of times drift is detected before the actual starting point of the concept drift and the missed detection count (MDC) indicating the number of times the detection mechanism fails to detect concept drift. Through the conducted experiments, we aim to answer the three research questions defined in Section 1.

1. Is it possible to utilise $\tilde{\varepsilon}^*$-drift values for the identification of concept drift?
The results obtained through the conducted experiments in this thesis indicate that $\tilde{\varepsilon}^*$-drift values are capable of detecting concept drift. In more detail, $\tilde{\varepsilon}^*$-drift with DDM and Mann-Whitney U test obtained similar MDC scores compared to error rate in combination with these two drift detectors. This means that $\tilde{\varepsilon}^*$-drift values with DDM and Mann-Whitney U test did not miss the occurrence of concept drift more than error rate with these two drift detectors on the datasets in the context of this thesis. As for the combination of $\tilde{\varepsilon}^*$-drift with threshold method, it correctly identified concept drift across all datasets. However, it is important to mention that currently $\tilde{\varepsilon}^*$-drift values are on average 100,000 times more expensive to compute in terms of wallclock time than the error rate. Depending on the setting and by making α,β-CROWN more computationally efficient, this difference in wallclock time can be minimised. Another interesting aspect to assess is the MTD metric, bringing us to the second research question.

2. Do $\tilde{\varepsilon}^*$-drift values in combination with a drift detector allow for detecting concept drift in an earlier stage than regular error rate-based methods?
Regarding the obtained results on the SEA configurations, $\tilde{\varepsilon}^*$-drift with DMM achieved the fastest drift detection performance in 4 out of 6 cases. The second fastest combination was error rate with DDM. This also applies to the error rate and $\tilde{\varepsilon}^*$-drift values combined with Mann-Whitney U test. While both combinations produced no false positives and no missed detections on four out of six SEA configurations, $\tilde{\varepsilon}^*$-drift was equally fast for one SEA configuration and faster on the remaining SEA configurations compared to error rate. $\tilde{\varepsilon}^*$-drift with threshold method was the fastest combination in five out of six SEA configurations yielding no false positives or missed detections compared to all other combinations.

As for the outcomes from the Hyperplane datasets, the fastest combination is features with KSWIN. However, this comes at the cost of a missed detection and a higher FAC compared to the

second fastest option: error rate with EDDM. The reasonably fast average MTD with a low FAC and no missed detections aligns with expectations since EDDM is explicitly designed to handle incremental drift. Given that the current implementation of EDDM accepts binary input making it incompatible with $\tilde{\varepsilon}^*$-drift values, an interesting future direction could be to adapt EDDM to accept $\tilde{\varepsilon}^*$-drift values and assess if $\tilde{\varepsilon}^*$-drift with EDDM is faster compared to error rate with EDDM. Again, $\tilde{\varepsilon}^*$-drift with Mann-Whitney U test was faster in detecting concept drift than error rate with Mann-Whitney U test. $\tilde{\varepsilon}^*$-drift with threshold method yielded no false positives and no missed detections and was the fastest combination compared to all other combinations.

Based on these results, we can conclude that $\tilde{\varepsilon}^*$-drift values are faster in the majority of the datasets but not consistently faster than the regular error rate.

3. Do $\tilde{\varepsilon}^*$-drift values in combination with a drift detector result in a lower number of false positives compared to regular error rate-based methods?
Similarly to error rate with Mann-Whitney U test, $\tilde{\varepsilon}^*$-drift with Mann-Whitney U test obtained no false positives on the Hyperplane and SEA datasets. $\tilde{\varepsilon}^*$-drift in combination with DDM achieved a lower average FAC than error rate with DDM across the Hyperplane datasets and all six SEA configurations. Furthermore, $\tilde{\varepsilon}^*$-drift with threshold method obtained no false positives for all Hyperplane and SEA datasets. These results reflect that $\tilde{\varepsilon}^*$-drift values obtain a similar number or lower number of false positives compared to the error rate.

As for future work, we recommend improving the computational efficiency of α,β-CROWN to decrease the wallclock time needed to compute $\tilde{\varepsilon}^*$-drift values. Furthermore, we restricted the experiments to synthetic data generators for binary classification tasks. We encourage investigating the usage of $\tilde{\varepsilon}^*$-drift values for concept drift detection in multiclass classification as well as regression settings. Error rate in combination with EDDM was the second fastest combination on the hyperplane datasets simulating incremental drift. It would be interesting to adapt EDDM to handle $\tilde{\varepsilon}^*$-drift values as input and compare the performance of $\tilde{\varepsilon}^*$-drift with EDDM against error rate with EDDM on incremental drift. We did not inject noise in the form of class swapping in our synthetic data generators since this could negatively impact the number of false positives. In future work, it could be interesting to investigate this relationship between noise and the potential increase in false positives.

Within the data stream community, there is a distinction between synthetically generated datasets and real-world datasets. Synthetically generated datasets provide annotations such as the exact point in time at which concept drift occurs and the type of concept drift. Currently, this information is often not available for real-world datasets. However, disadvantages of synthetically generated datasets include the lower complexity of the datasets and the risk of committing data bias [9].

Souza et al. [9] published the Insects dataset being a semi-real-world dataset. The data was collected from optical sensors under controlled conditions to identify six types of flying insects. Importantly, the wing-beat frequency of a flying insect allows for distinguishing between different insect species. Changes in the temperature impact the wing-beat frequency of the flying insects. By changing the temperature under controlled settings, the researchers introduced simulated different types of concept drift such as sudden and incremental drift. We encourage the data stream community to provide more open-source real-world datasets such as the Insects dataset from [9] with annotations about concept drift.

In Section 4.1, we explored the usage of approximating distances to the decision boundary for correctly classified instances through $\tilde{\varepsilon}^*$ values. One of the proposed methods involved splitting the distribution of $\tilde{\varepsilon}^*$ values based on the class label. As depicted in Figures 7a and 8a, there is a clear

difference in terms of observed $\tilde{\varepsilon}^*$ values before and after concept drift. Due to concept drift, the distribution of $\tilde{\varepsilon}^*$ values for the orange class after concept drift is missing smaller $\tilde{\varepsilon}^*$ values. This shift is demonstrated in Figure 11a.

Based on initial findings obtained in an early experimental phase, we observed that splitting the distribution of $\tilde{\varepsilon}^*$ values based on the class label is slower and yields a higher number of false positives compared to the error rate. Nevertheless, this approach indicates a distinct shift in the distribution of $\tilde{\varepsilon}^*$ values before and after concept drift. Therefore, it is worth further investigating this shift for the detection of concept drift.

Finally, we foresee that $\tilde{\varepsilon}^*$-drift values could be used for applications beyond concept drift detection. For instance, $\tilde{\varepsilon}^*$-drift values could be used in a self-monitoring AI model to enhance performance. Such a method could aim for a low range of $\tilde{\varepsilon}^*$-drift values during the training phase of the model (i.e., similarly to the threshold method) to ensure robust and reliable model performance. By observing a sufficient number of $\tilde{\varepsilon}^*$-drift values during training, we could compute the probability of encountering a specific $\tilde{\varepsilon}^*$-drift value. A probability smaller than 0.05 could be considered an indication of concept drift. Alternatively, we could compute the mean and standard deviation of the observed $\tilde{\varepsilon}^*$-drift values during training. If a $\tilde{\varepsilon}^*$-drift value deviates by three standard deviations from the mean, it could also be considered concept drift.

Another application could be to combine covariate shift with $\tilde{\varepsilon}^*$-drift values. In this thesis, we specifically focused on concept drift. However, another important type of data drift is covariate shift (see Figures 1a and 1b). Covariate shift represents a shift in the distribution of the features, while the relationship between instances and targets does not change. As a result model performance might or might not be impacted. The latter is also known as virtual drift.

We think that $\tilde{\varepsilon}^*$-drift values in combination with covariate shift detectors could provide information about the behaviour of the model. For instance, with the detection of covariate shift, we could utilise $\tilde{\varepsilon}^*$-drift values to identify how far away misclassification are from the decision boundary. Namely, this could indicate underrepresented regions which, depending on the context, might require extra attention. Via $\tilde{\varepsilon}^*$-drift values, we could identify these regions and retrain the model accordingly to potentially improve performance in the case of covariate shift.

References

[1] C. Raab, M. Heusinger, and F. Schleif, "Reactive soft prototype computing for concept drift streams," *Neurocomputing*, vol. 416, pp. 340–351, 2020.

[2] A. L. Suárez-Cetrulo, A. Cervantes, and D. Quintana, "Incremental market behavior classification in presence of recurring concepts," *Entropy*, vol. 21, no. 1, p. 25, 2019.

[3] A. Razak MS, C. R. Nirmala, M. Aljohani, and B. R. Sreenivasa, "A novel technique for detecting sudden concept drift in healthcare data using multi-linear artificial intelligence techniques," *Frontiers in Artificial Intelligence*, vol. 5, 2022.

[4] K. Namitha and G. S. Kumar, "Concept drift detection in data stream clustering and its application on weather data," *International Journal of Agricultural and Environmental Information Systems*, vol. 11, no. 1, pp. 67–85, 2020.

[5] W. W. Ng, J. Zhang, C. S. Lai, W. Pedrycz, L. L. Lai, and X. Wang, "Cost-sensitive weighting and imbalance-reversed bagging for streaming imbalanced and concept drifting in electricity pricing classification," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 3, pp. 1588–1597, 2018.

[6] A. A. Toor, M. Usman, F. Younas, A. C. M. Fong, S. A. Khan, and S. Fong, "Mining massive e-health data streams for iomt enabled healthcare systems," *Sensors*, vol. 20, no. 7, p. 2131, 2020.

[7] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2019.

[8] R. S. M. de Barros and S. G. T. de Carvalho Santos, "An overview and comprehensive comparison of ensembles for concept drift," *Information Fusion*, vol. 52, pp. 213–244, 2019.

[9] V. M. A. de Souza, D. M. dos Reis, A. G. Maletzke, and G. E. A. P. A. Batista, "Challenges in benchmarking stream learning algorithms with real-world data," *Data Mining Knowledge Discovery*, vol. 34, no. 6, pp. 1805–1858, 2020.

[10] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. Gama, "Machine learning for streaming data: state of the art, challenges, and opportunities," *ACM SIGKDD Explorations Newsletter*, vol. 21, no. 2, pp. 6–22, 2019.

[11] J. Gama, P. Medas, G. Castillo, and P. P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence, 17th Brazilian Symposium on Artificial Intelligence, 2004, Proceedings*, vol. 3171 of *Lecture Notes in Computer Science*, pp. 286–295, Springer, 2004.

[12] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Advances in Intelligent Data Analysis VIII, 8th International Symposium on Intelligent Data Analysis*, vol. 5772 of *Lecture Notes in Computer Science*, pp. 249–260, Springer, 2009.

[13] F. Bayram, B. S. Ahmed, and A. Kassler, "From concept drift to model degradation: An overview on performance-aware drift detectors," *Knowlegde-Based Systems*, vol. 245, p. 108632, 2022.

[14] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning*, vol. 23, no. 1, pp. 69–101, 1996.

[15] I. I. F. Blanco, J. del Campo-Ávila, G. Ramos-Jiménez, R. M. Bueno, A. A. O. Díaz, and Y. C. Mota, "Online and non-parametric drift detection methods based on hoeffding's bounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 810–823, 2015.

[16] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proceedings of the Seventh SIAM International Conference on Data Mining*, pp. 443–448, SIAM, 2007.

[17] W. N. Street and Y. Kim, "A streaming ensemble algorithm (SEA) for large-scale classification," in *Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 377–382, ACM, 2001.

[18] C. Salperwyck, M. Boullé, and V. Lemaire, "Concept drift detection using supervised bivariate grids," in *2015 International Joint Conference on Neural Networks*, pp. 1–9, IEEE, 2015.

[19] D. M. dos Reis, P. A. Flach, S. Matwin, and G. E. A. P. A. Batista, "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*, pp. 1545–1554, ACM, 2016.

[20] L. Baier, T. Schlör, J. Schoeffer, and N. Kühl, "Detecting concept drift with neural network model uncertainty," in *56th Hawaii International Conference on System Sciences* (T. X. Bui, ed.), pp. 835–844, ScholarSpace, 2023.

[21] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations*, 2015.

[22] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd International Conference on Learning Representations*, 2014.

[23] R. Bunel, J. Lu, I. Turkaslan, P. H. S. Torr, P. Kohli, and M. P. Kumar, "Branch and bound for piecewise linear neural network verification," *Journal for Machine Learning Research*, vol. 21, pp. 42:1–42:39, 2020.

[24] M. König, A. W. Bosman, H. H. Hoos, and J. N. van Rijn, "Critically assessing the state of the art in neural network verification," *Journal of Machine Learning Research*, vol. 25, no. 12, pp. 1–53, 2024.

[25] A. W. Bosman, H. H. Hoos, and J. N. van Rijn, "A preliminary study of critical robustness distributions in neural network verification," in *Proceedings of the 6th workshop on formal methods for ML-enabled autonomous systems*, 2023.

[26] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Computer Aided Verification - 29th International Conference*, vol. 10426 of *Lecture Notes in Computer Science*, pp. 97–117, Springer, 2017.

[27] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C. Hsieh, and J. Z. Kolter, "Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021*, pp. 29909–29921, 2021.

[28] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Survey*, vol. 46, no. 4, pp. 44:1–44:37, 2014.

[29] T. R. Hoens, R. Polikar, and N. V. Chawla, "Learning from streaming data with concept drift and imbalance: an overview," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 89–101, 2012.

[30] J. G. Moreno-Torres, T. Raeder, R. Alaíz-Rodríguez, N. V. Chawla, and F. Herrera, "A unifying view on dataset shift in classification," *Pattern Recognition*, vol. 45, no. 1, pp. 521–530, 2012.

[31] I. Khamassi, M. Sayed Mouchaweh, M. Hammami, and K. Ghédira, "Discussion and review on evolving data streams and concept drift adapting," *Evoloving Systems*, vol. 9, no. 1, pp. 1–23, 2018.

[32] R. Sebastiao and J. Gama, "A study on change detection methods," in *Progress in artificial intelligence, 14th Portuguese Conference on Artificial Intelligence*, pp. 12–15, 2009.

[33] E. A. Rudnitskaya and M. A. Poltavtseva, "Adversarial machine learning protection using the example of evasion attacks on medical images," *Automatic Control and Computer Sciences*, vol. 56, no. 8, pp. 934–941, 2022.

[34] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Computer Science Review*, vol. 37, p. 100270, 2020.

[35] L. Li, T. Xie, and B. Li, "Sok: Certified robustness for deep neural networks," in *44th IEEE Symposium on Security and Privacy*, pp. 1289–1310, IEEE, 2023.

[36] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C. Hsieh, "Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers," in *9th International Conference on Learning Representations*, OpenReview.net, 2021.

[37] A. D. Palma, R. Bunel, A. Desmaison, K. Dvijotham, P. Kohli, P. H. S. Torr, and M. P. Kumar, "Improved branch and bound for neural network verification via lagrangian decomposition," *Computing Resource Repository*, vol. abs/2104.06718, 2021.

[38] M. N. Müller, C. Brix, S. Bak, C. Liu, and T. T. Johnson, "The third international verification of neural networks competition (VNN-COMP 2022): Summary and results," *ArXiv*, 2022.

[39] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *Fourth international workshop on knowledge discovery from data streams*, vol. 6, pp. 77–86, Citeseer, 2006.

[40] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[41] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem, and A. Bifet, "River: machine learning for streaming data in python," *Journal of Machine Learning Research*, vol. 22, pp. 110:1–110:8, 2021.

[42] G. Hulten, L. Spencer, and P. M. Domingos, "Mining time-changing data streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 97–106, ACM, 2001.

[43] A. Bifet, J. Read, B. Pfahringer, G. Holmes, and I. Zliobaite, "CD-MOA: change detection framework for massive online analysis," in *Advances in Intelligent Data Analysis XII - 12th International Symposium*, vol. 8207 of *Lecture Notes in Computer Science*, pp. 92–103, Springer, 2013.

[44] H. Chen, H. Zhang, S. Si, Y. Li, D. S. Boning, and C. Hsieh, "Robustness verification of tree-based models," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, pp. 12317–12328, 2019.

[45] M. Andriushchenko and M. Hein, "Provably robust boosted decision stumps and trees against adversarial attacks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, pp. 12997–13008, 2019.

[46] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC genomics*, vol. 21, pp. 1–13, 2020.

[47] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Information Fusion*, vol. 37, pp. 132–156, 2017.

A SEA Results

Table 8: Mean time to detection (MTD), false alarm count (FAC) and missed detection count (MCD) for different input with drift detector combinations on 5 seeded configurations of the SEA dataset with variants 0 (8) and 2 (7). Mann-Whitney U test is abbreviated as MWU. A dash represents that the input with drift detector combination was unable to flag concept drift within the acceptable delay window. A cross indicates that the input is not compatible with the drift detector. Bold and underlined entries represent the fastest and second-fastest combinations respectively. An asterisk stands for the fastest combination with 0 FAC and 0 MCD.

Drift Detector	MTD			FAC			MDC		
	error rate	features	$\tilde{\varepsilon}^*$-drift	error rate	features	$\tilde{\varepsilon}^*$-drift	error-rate	features	$\tilde{\varepsilon}^*$-drift
DDM	**69±62**	-	<u>92±168</u>	3±4	-	2±2	0	-	0
EDDM	303±88	x	x	1±1	x	x	0	x	x
ADWIN	539±127	-	-	0	-	-	1	-	-
KSWIN	-	445±295	-	-	3±3	-	-	1	-
MWU	456±243	511±82	251±129	0	22±16	0	1	2	1
threshold	x	x	29±35	x	x	0	x	x	0

Table 9: Mean time to detection (MTD), false alarm count (FAC) and missed detection count (MCD) for different input with drift detector combinations on 5 seeded configurations of the SEA dataset with variants 0 (8) and 3 (9.5). Mann-Whitney U test is abbreviated as MWU. A dash represents that the input with drift detector combination was unable to flag concept drift within the acceptable delay window. A cross indicates that the input is not compatible with the drift detector. Bold and underlined entries represent the fastest and second-fastest combinations respectively. An asterisk stands for the fastest combination with 0 FAC and 0 MCD.

Drift Detector	MTD			FAC			MDC		
	error rate	features	$\tilde{\varepsilon}^*$-drift	error rate	features	$\tilde{\varepsilon}^*$-drift	error-rate	features	$\tilde{\varepsilon}^*$-drift
DDM	<u>28±24</u>	-	**22±34**	3±4	-	2±2	0	-	0
EDDM	167±30	x	x	1±1	x	x	0	x	x
ADWIN	272±72	-	-	0	-	-	0	-	-
KSWIN	-	445±295	-	-	3±3	-	-	1	-
MWU	89±12	511±82	81±11*	0	22±16	0	0	2	0
threshold	x	x	5±4	x	x	0	x	x	0

Table 10: Mean time to detection (MTD), false alarm count (FAC) and missed detection count (MCD) for different input with drift detector combinations on 5 seeded configurations of the SEA dataset with variants 1 (9) and 2 (7). Mann-Whitney U test is abbreviated as MWU. A dash represents that the input with drift detector combination was unable to flag concept drift within the acceptable delay window. A cross indicates that the input is not compatible with the drift detector. Bold and underlined entries represent the fastest and second-fastest combinations respectively. An asterisk stands for the fastest combination with 0 FAC and 0 MCD.

Drift Detector	MTD			FAC			MDC		
	error rate	features	$\tilde{\varepsilon}^*$-drift	error rate	features	$\tilde{\varepsilon}^*$-drift	error-rate	features	$\tilde{\varepsilon}^*$-drift
DDM	<u>11±7</u>	-	**5±3**	7±3	-	5±3	1	-	1
EDDM	117±46	x	x	0	x	x	0	x	x
ADWIN	221±90	-	-	0	-	-	0	-	-
KSWIN	-	445±295	-	-	3±3	-	-	1	-
MWU	55±15	511±82	47±11*	0	22±16	0	0	2	0
threshold	x	x	4±3	x	x	0	x	x	0

B Dataset Configurations

Dataset	SEA
Configuration Name	SEA_8_9
#instances context 1	5000
#instances context 2	5000
#total instances	10.000
drift postition	5000
parameters context 1	variant = 0 noise = 0 seeds = 100-104
parameters context 2	variant = 1 noise = 0 seeds = 105-109

Dataset	SEA
Configuration Name	SEA_8_7
#instances context 1	5000
#instances context 2	5000
#total instances	10.000
drift postition	5000
parameters context 1	variant = 0 noise = 0 seeds = 100-104
parameters context 2	variant = 2 noise = 0 seeds = 105-109

Dataset	SEA
Configuration Name	SEA_8_9.5
#instances context 1	5000
#instances context 2	5000
#total instances	10.000
drift postition	5000
parameters context 1	variant = 0 noise = 0 seeds = 100-104
parameters context 2	variant = 3 noise = 0 seeds = 105-109

Dataset	SEA
Configuration Name	SEA_9_7
#instances context 1	5000
#instances context 2	5000
#total instances	10.000
drift postition	5000
parameters context 1	variant = 1 noise = 0 seeds = 100-104
parameters context 2	variant = 2 noise = 0 seeds = 105-109

Dataset	SEA
Configuration Name	SEA_9_9.5
#instances context 1	5000
#instances context 2	5000
#total instances	10.000
drift postition	5000
parameters context 1	variant = 1 noise = 0 seeds = 100-104
parameters context 2	variant = 3 noise = 0 seeds = 105-109

Dataset	SEA
Configuration Name	SEA_7_9.5
#instances context 1	5000
#instances context 2	5000
#total instances	10.000
drift postition	5000
parameters context 1	variant = 2 noise = 0 seeds = 100-104
parameters context 2	variant = 3 noise = 0 seeds = 105-109

Dataset	Hyperplane
Configuration Name	HYP_m001
#instances context 1	5000
#instances context 2	5000
#total instances	10.000
drift position	5000
parameters context 1	n_features = 10 n_drift_features = 2 mag_change = 0.001 sigma = 0 noise = 0 seeds = 105-109
parameters context 2	n_features = 10 n_drift_features = 2 mag_change = 0.001 sigma = 0 noise = 0 seeds = 105-109

C Neural Network Configurations

Dataset	Seeds	Architecture (units)	Batch Size	Epochs	Optimizer	Learning Rate
SEA_8_9	100;105	3-2-2	16	100	Adam	0.005
SEA_8_9	101;106	3-2-2	16	100	Adam	0.005
SEA_8_9	102;107	3-2-2	64	100	RMSprop	0.005
SEA_8_9	103;108	3-2-2	32	100	RMSprop	0.01
SEA_8_9	104;109	3-2-2	128	100	Adam	0.01
SEA_8_7	100;105	3-2-2	16	100	Adam	0.005
SEA_8_7	101;106	3-2-2	16	100	Adam	0.005
SEA_8_7	102;107	3-2-2	64	100	RMSprop	0.005
SEA_8_7	103;108	3-2-2	32	100	RMSprop	0.01
SEA_8_7	104;109	3-2-2	128	100	Adam	0.01
SEA_8_9.5	100;105	3-2-2	16	100	Adam	0.005
SEA_8_9.5	101;106	3-2-2	16	100	Adam	0.005
SEA_8_9.5	102;107	3-2-2	64	100	RMSprop	0.005
SEA_8_9.5	103;108	3-2-2	32	100	RMSprop	0.01
SEA_8_9.5	104;109	3-2-2	128	100	Adam	0.01
SEA_9_7	100;105	3-2-2	128	100	Adam	0.01
SEA_9_7	101;106	3-2-2	32	100	Adam	0.005
SEA_9_7	102;107	3-2-2	64	50	Adam	0.01
SEA_9_7	103;108	3-2-2	16	100	Adam	0.005
SEA_9_7	104;109	3-2-2	64	100	Adam	0.005
SEA_9_9.5	100;105	3-2-2	128	100	Adam	0.01
SEA_9_9.5	101;106	3-2-2	32	100	Adam	0.005
SEA_9_9.5	102;107	3-2-2	64	50	Adam	0.01
SEA_9_9.5	103;108	3-2-2	16	100	Adam	0.005
SEA_9_9.5	104;109	3-2-2	64	100	Adam	0.005
SEA_7_9.5	100;105	3-2-2	32	100	RMSprop	0.005
SEA_7_9.5	101;106	3-2-2	64	100	RMSprop	0.01
SEA_7_9.5	102;107	3-2-2	64	100	Adam	0.01
SEA_7_9.5	103;108	3-2-2	128	100	RMSprop	0.01
SEA_7_9.5	104;109	3-2-2	32	100	Adam	0.005
HYP_m001	105;105	10-8-4-2	16	100	SGD	0.01
HYP_m001	106;106	10-8-4-2	16	100	SGD	0.01
HYP_m001	107;107	10-8-4-2	16	100	Adam	0.001
HYP_m001	108;108	10-8-4-2	16	100	SGD	0.01
HYP_m001	109;109	10-8-4-2	16	100	SGD	0.001

D α,β-CROWN YAML configuration file

```yaml
specification:
  norm: .inf
  epsilon: 0.001 (example)
solver:
  batch_size: 2048
  beta-crown:
    iteration: 20
    lr_beta: 0.03
  mip:
    parallel_solvers: 8
    solver_threads: 4
    refine_neuron_time_percentage: 0.8
bab:
  branching:
    candidates: 5
    reduceop: max
  timeout: 600
```

E Misclassifications

Dataset	Seeds	Misclassifications Training Phase	First Misclassification After Concept Drift	Timeouts	out-of-memory errors
SEA_8_9	100;105	6	5003	0	0
SEA_8_9	101;106	5	5036	0	0
SEA_8_9	102;107	9	5000	0	0
SEA_8_9	103;108	30	5008	0	0
SEA_8_9	104;109	7	5009	0	0
SEA_8_7	100;105	6	5008	0	0
SEA_8_7	101;106	5	5005	0	0
SEA_8_7	102;107	9	5005	0	0
SEA_8_7	103;108	30	5007	0	0
SEA_8_7	104;109	7	5000	0	0
SEA_8_9.5	100;105	6	5003	0	0
SEA_8_9.5	101;106	5	5011	0	0
SEA_8_9.5	102;107	9	5000	0	0
SEA_8_9.5	103;108	30	5000	0	0
SEA_8_9.5	104;109	7	5006	0	0
SEA_9_7	100;105	9	5005	0	0
SEA_9_7	101;106	5	5005	0	0
SEA_9_7	102;107	11	5000	0	0
SEA_9_7	103;108	10	5007	0	0
SEA_9_7	104;109	6	5000	0	0
SEA_9_9.5	100;105	9	5003	0	0
SEA_9_9.5	101;106	5	5011	0	0
SEA_9_9.5	102;107	11	5041	0	0
SEA_9_9.5	103;108	10	5000	0	0
SEA_9_9.5	104;109	6	5006	0	0
SEA_7_9.5	100;105	6	5003	0	0
SEA_7_9.5	101;106	19	5005	0	0
SEA_7_9.5	102;107	8	5000	0	0
SEA_7_9.5	103;108	6	5000	0	0
SEA_7_9.5	104;109	9	5000	0	0
HYP_m001	105;105	10	5115	0	0
HYP_m001	106;106	13	5020	0	0
HYP_m001	107;107	12	5122	0	0
HYP_m001	108;108	5	5091	0	0
HYP_m001	109;109	32	5004	0	0