Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Concept Drift Detection and adaptation for machine learning

Pratyusha Bhattacharya

Course of Study:	M.Sc Computer Science
Examiner:	Prof. Dr. Stefan Wagner
Supervisor:	Markus Haug, M.Sc.
Commenced:	November 9, 2021
Completed:	May 23, 2022

Abstract

Machine learning models encounter plethora of challenges due to the changing data over time. This phenomenon is known as concept drift. Existing techniques for concept drift detection have shown promising results, but they require true labels as a precondition for drift detection to be successful. True labels are limited and expensive, especially in real-world application settings. To deal with this problem, this thesis proposes an AutoEncoder based Drift Detector (AEDD) technique for drift detection, that can detect drifts without access to true labels. This study combines two different techniques to achieve this. First, reconstruction error is measured by an autoencoder and then from the measured reconstruction error, using the ADaptive sliding WINdow (ADWIN) technique identifies structural changes over time. The observed drifts are utilized to retrain the prediction model. The thesis demonstrates the superiority of the technique by showing that the AEDD outperforms alternative state-of-the-art algorithms for classification tasks on real world datasets with artificially induced drift.

Acknowledgement

I want to express my sincere gratitude to my Professor Dr. Stefan Wagner and my supervisor Mr. Markus Haug. Without their guidance throughout the project, this thesis would not have been possible. Their expertise, insights, and editing shaped this thesis.

Last but not least, I am eternally grateful for my parents' and brother's unwavering support and encouragement.

Contents

List of Figures

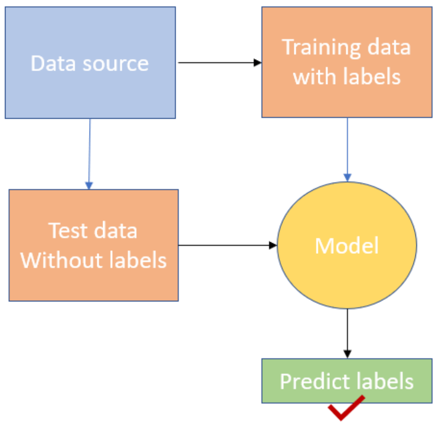
List of Tables

Acronyms

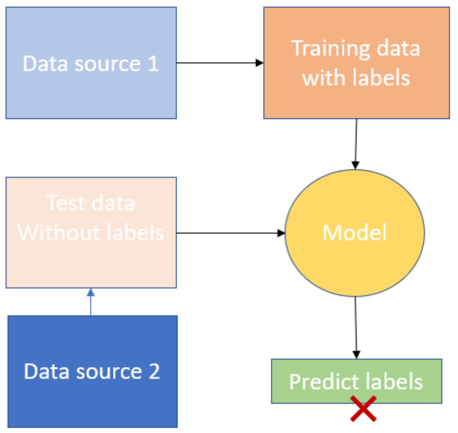
ADWIN	ADaptive sliding WINdow. 3
AEDD	AutoEncoder based Drift Detector. 3
ARF	Adaptive Random Forest. 37
CUSUM	CUmulative SUM. 32
D3	Discriminative Drift Detector. 33
DDM	Drift Detection Method. 33
DWM	Dynamic Weighted Majority. 37
EDDM	Early Drift Detection Method. 33
EFDT	Extremely Fast Decision Tree classifier. 35
FHDDM	Fast Hoeffding Drift Detection Method. 32
HDDDM	Hellinger distance-based drift detection. 33
HT	Hoeffding Tree. 35
IKS	Incremental Kolmogorov-Smirnov. 33
MD3	Margin Density Drift Detection. 34
MDR	Missed Detection Ratio. 57
ML	Machine Learning. 15
MTD	Mean Time to Detection. 57
MTFA	Mean Time Between False Alarms. 57
ND	Number of Detections. 58
STUDD	Student–Teacher Unsupervised Concept Drift Detection. 40
SVM	Support Vector Machine. 34
UDD	Uncertainty Drift Detection. 39

1 Introduction

Nowadays, Machine Learning (ML) models are being used in almost every industry to make the ever-increasing volumes of data accessible. Most ML experts develop models with the presumption that future incoming data streams will be steady, meaning that the data generation process will not vary over time while implementing models. However, in most real-world applications, this assumption is incorrect [APHW03]. The model's predictive performance degrades due to distributional changes in training and test data over time (shown in Figure 1.1). This phenomenon is known as concept drift or dataset shift in the ML world.



(a)



(b)

Figure 1.1: illustrate (a) static and (b) dynamic environment

ML practitioners have created several learning algorithms that can adapt incrementally [Sha+12] under concept drift or detect concept drift explicitly and initiate a model retraining [BG07]-[GMCR04]. These strategies demand full and rapid access to the true label labels, which is an unreasonable assumption in most real-world applications. To explain this an example with a production line with manual quality control at the end of the line is considered. A predictive model can replace manual quality control and reduce repetitive and expensive human labor by gathering sensor data from all manufacturing stations and integrating it with previously collected quality judgments (labels) of human experts. However, due to changes in raw materials, machine wear, aging sensors, or changing indoor temperatures due to seasonal variations, this prediction model is likely to experience concept drift.This issue creates an adverse cost implication and quality control problems for the company. Traditional concept drift detection algorithms are not appropriate in this situation since a constant stream of true label labels for concept drift identification is not provided.

The thesis presents an AEDD that can detect drift without access to true label labels. This concept drift detection algorithm is based on the estimated reconstruction error of a neural network at inference time. It is assumed that true labels are available upon request (e.g., provided by domain experts) for retraining the prediction model in a detected drift scenario. The thesis compares AEDD to other state-of-the-art drift detection methods on real-world datasets where drift has been explicitly induced.

1.1 Motivation

Concept drift detection and adaption are typically accomplished by combining prediction models with a change detection method [GRB+19]. When any modification in input data distribution is detected, the detector raises the alarm in the monitoring system. The prediction model's performance depends on the error rate used as input data for most of the standard existing drift detection techniques. If the performance of the prediction methods deteriorates dramatically, drift is detected in the traditional system. Nevertheless, labels are not readily available to quantify model performance in several real-world settings. Due to the high cost of labeling, some labels may arrive late or never arrive. This presents significant difficulty for learning algorithms that rely on concept drift detection [Žli10a]. An unsupervised techniques for concept drift detection are gaining popularity in these settings. These techniques presume that no additional labels are available during the deployment of the model in a test set after an initial fit. As the prevalent unsupervised drift detectors have not satisfactory results, in this thesis we have focused on improving existing drift detection problems with a novel deep learning approach.

1.2 Concept drift in data stream mining

Due to the rising volume of data generated by diverse sources such as social networks, online enterprises, and military-financial applications, data stream mining has become a key area of study [FB13]. Due to computational and memory constraints, most of this data is unused and discarded. Due to the dynamic nature of the streaming environment, they must be processed right away, or they would be lost [KŽB+14]-[GBBC19]. As we know that the data streams are generated in a non-stationary environment, learning from them is a continual process. If it is envisaged that the environment may change throughout the deployment phase, the practitioner must devise a plan to detect changes and adapt accordingly [CGB20].

1.3 Problem statement

As explained in earlier sections, the natural concept drift in a data stream is a significant concern for production models since it substantially impacts model performance. The most efficient way to address this issue is to identify when the learned relation between dependent and independent variables is no longer applicable for the incoming data. Afterward, a new model is trained to learn the novel concepts.

We must first create a viable method for detecting drifts based on the criteria listed above to achieve this goal. Following that, we must complete an application that can view the data together with the detection findings and provide some strategies to adapt with the non-static data. This thesis aims to create and deploy a practical methodology to investigate concept drift on high-dimensional streams and make it easier to analyze real-world datasets.

1.4 Contribution of this thesis

The contribution of this thesis are the followings:

1) An autoencoder based unsupervised drift detector is developed to identify model drift where true labels are inaccessible or might be available later.

2)The encoder and decoder paradigm is used to detect the model drift by monitoring the reconstruction error [HZ93] with ADWIN [BG07]. This approach is also compared with the existing baseline models for detecting concept drift.

3) An incremental adaptive model is then used as an inner learner for classification tasks. The drift detector updates the model approximation with the new data sets when it indicates drift resulting automatic handling of model drift.

4) The performance analysis between the state of art adaptive algorithms and the proposed AEDD algorithm is compared and results are analysed.

1.5 Thesis outline

This thesis is organized in the seven chapters.

Chapter 2 (Background): elaborates on the data streams, and explains concept drift and possible variations. It also provides possible reasons for concept drift occurring in real-world data sets, the core idea behind the concept drift learners, and possible solutions to detect and avoid concept drift.

Chapter 3 (Literature Review and Related work): provides an overview of the existing drift detection methods and identifies common trends. It also reveals the current research gap among the state of art algorithms. This chapter also summarizes the working mechanism for different concept drift detection and adaptation strategies.

Chapter 4 (Study design): outlines the research questions and our approach to answering with the proposed methodology.

Chapter 5 (Proposed Methodology): proposes the novel concept of a AEDD, which is generative and unsupervised. It tries to catch the change for the essential features more relevant to prediction. It illustrates the theoretical details of the proposed .

Chapter 6 (Experimental results): explains data generation, data sets, and parameter setups. It also contains different experiments to approach the research questions abbreviated later and demonstrates the impact of these experiments. After combining drift detection and drift adaptation strategies, other existing drift detectors' performance and accuracy are compared with the proposed method.

Chapter 7 (Conclusion and Future work): concludes this thesis and highlights possible areas for future work.

2 Background

This chapter provides a brief overview of the technological underpinnings of data stream, concept drift and their different forms. It also discusses drift detection frameworks and drift adaptation mechanisms.

2.1 Data stream

Before we delve further, a few terminologies are clarified in this section. Dynamic data environments are often considered to elaborate concept drift in production. While discussing the concept drift detection and adaptation, it is always referenced to the data streams, in which data instances occur continually and sequentially throughout the period. Because the streaming data is frequently produced on the go and at a rapid and variable rate with a range up to infinity - it is an ideal option to take it as a reference for expanding dynamic data distributions.

Nevertheless, these streams may not be the only source of concept drift. Concept drift detection approaches frequently use sliding windows or groupings of progressively ordered data [GCGD20] to frame a conversation incorporating both stream and batch settings. One window, for example, may contain cases from the most recent known concept that were used to train or update the deployed model, whereas the other may have instances that have experienced concept drift. In this example, The first window will be the reference window and the second window is the detection window. This concept is shown in Figure 2.1.

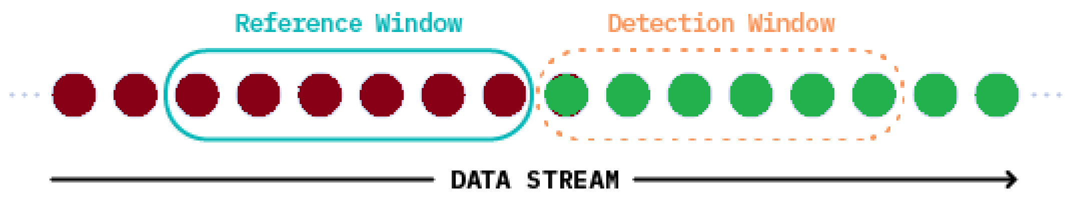


Figure 2.1: Data stream divided into reference window and the detection window

2.2 Concept drift definition

Concept drift occurs when the statistical features of a target domain vary unpredictably over time [LZL14]. Literature in [SG86] was the first to suggest that similar occurrences could be classified as noise information or non-noise information at various time instances. Modifications in hidden factors that cannot be observed directly may be the reason for occurrence of these drifts [LSZL17].

2.2.1 Mathematical notation

The ML model's challenge is approximating a mapping function F learned from the seen training data, which is generally speaking, static. Here, the input data X is given to forecasting an output value Y, known as predictive modeling. We define a function as such:

$$F : X \rightarrow Y$$

The definition of a joint distribution can be denoted as $P(X, Y)$. It is assumed a sample from the data-stream can be represented as S and D:

$$D_{[0,t_1]} = \{S_0, S_1, S_2, ..., S_{t_1}\}$$

Where, The input feature vector is described as X and label vector denoted as Y and $S_i = (X_i, Y_i)$.

Regardless of whether or not there is concept drift, the classification problem can be characterized as follows:

The joint distribution can be written as:

$$P(X, Y) = P(Y|X)P(X) = P(X|Y)P(Y)$$

$$\mathrm{P}(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Where,

- $P(Y|X)$ is the posterior probability of class distribution for the given data
- $P(X|Y)$ is the class conditional probability
- $P(X)$ is known as the evidence
- $P(Y)$ is the prior probability of classes

2.2.2 Aspect of a concept drift

The phenomena of changing data distributions w.r.t. to training data over time, impacts machine learning model performance. This can be described in diverse ways by the ML and data mining communities [MRA+12]. Several authors have defined concept drift under different titles, like dataset shift [Sto09] or concept shift [WK96]. In [MRA+12], the researchers claim that concept drift or shift is merely one subtype of dataset shift, encompassing covariate shift, prior probability shift, and concept shift. In other words, the concept drift is a subset of dataset shift. Nevertheless, because concept drift is frequently linked to covariate shift and prior probability shift, many publications [WK96] refer to the problem as "concept drift". During training ($\mathrm{t_r}$) and test time ($\mathrm{t_{st}}$), dataset shift is associated with changes in the joint probability distribution of input data x and corresponding label y:

$$P_{\mathrm{tr}}(x,y) \neq P_{\mathrm{tst}}(x,y) \ \& \ P_{\mathrm{t_0}}(x,y) = P_{\mathrm{t_1}}(x,y),$$

Where, t_0 and t_1 are two separate specific times with $t_1 > t_0$, is indeed a typical explanation of concept drift. Dataset shift is concerned with the differences between the historical training and testing data environments, while concept drift is concerned with the conceptual structure of the data over time.

At a particular time t_1, the concept can be defined as $P_{t_1}(X,Y)$ and during a specific time period $[0,t_1]$, the distribution can be expressed as $P_{[0,t_1]}(X,Y)$. There are three ways that model drift might happen:

1. $P(Y)$ The class priors may alter over time

2. The class conditional probability of one or more classes $p(X|Y)$ might change

3. The posterior distributions of class memberships $p(Y|X)$ might change over time

As a result, drift can happen in three ways:

1. Virtual drift:

Virtual drift occurs when input data distribution changes with time but the posterior probability remains the same (as shown in Figure 2.2). This is also known as co-variate shift or input drift which can lead to model performance degradation, where,

$$P_t(X) \neq P_{t+1}(X) \ \& \ P_t(Y|X) = P_{t+1}(Y|X)$$

2. Prior drift:

Prior drift can happen when the prior probability of classes change with time but the posterior probability of the given data with respect to the classes remains the same. This kind of drift can change a model's decision boundary which will degrade the model performance where,

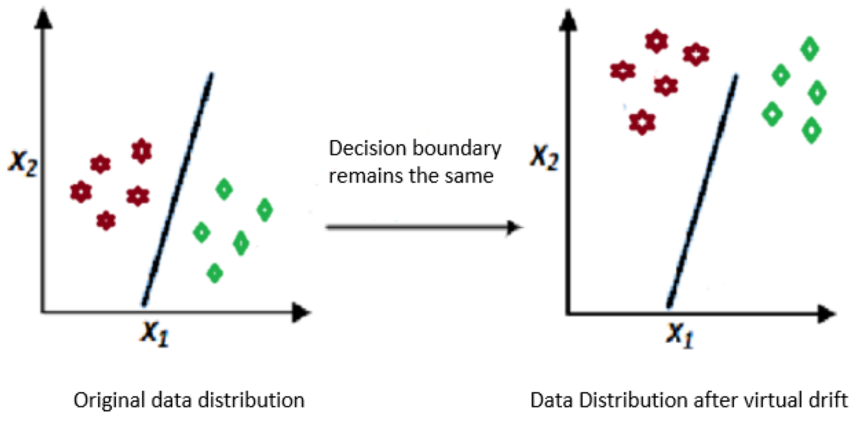

X2
X1
Decision boundary remains the same
Original data distribution
Data Distribution after virtual drift

Figure 2.2: Example of Virtual drift

$$P_t(Y) \neq P_{t+1}(Y) \;\&\; P_t(Y|X) = P_{t+1}(Y|X)$$

3. Concept drift:

This kind of drift happens when the posterior probability changes with time (shown in Figure 2.3). This might lead to obsoleting the model in this case as the underlying approximating function changes and as a result, the model won't be able to predict reliably when,

$$P_t(Y|X) \neq P_{t+1}(Y|X)$$

If a concept drift occurs between two point of time, it can be denoted as:

$$P_0(X,Y) \neq P_{t_1}(X,Y)$$

and if the concept between two time periods can change i.e $[0, t_1]\&[t_1 + 1, t_2]$, It can be expressed as:

$$P_{[0,t_1]}(X,Y) \neq P_{[t_1+1,t_2]}(X,Y)$$

2.3 Reasoning for drift in real world scenario:

There are many reasons which can cause the drifting model, but we will discuss a few example below. An example can be seen with a technical problem in the context of the upstream data frame. Due to a faulty pipeline, a modifications in one of the feature's values can occur. This results an unplanned unreported format change of feature's. Even a difference in the default value can

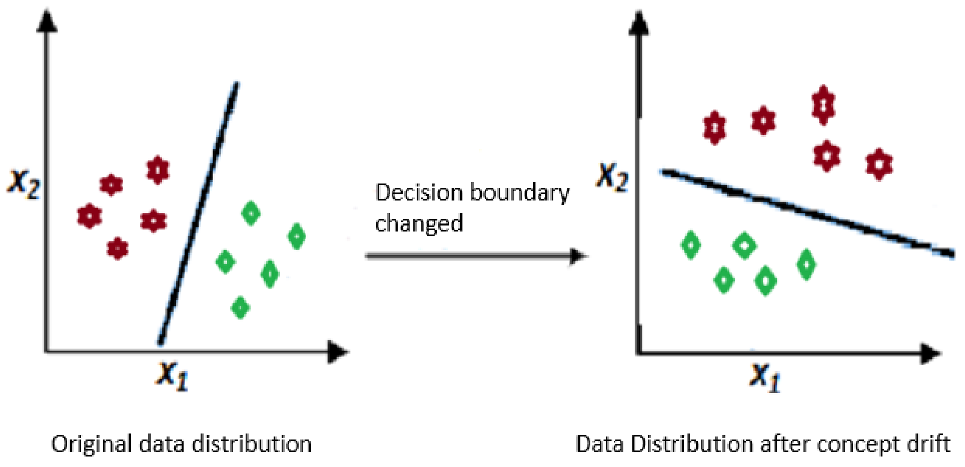

X2
X1
Decision boundary changed
X2
X1
Original data distribution
Data Distribution after concept drift

Figure 2.3: Example of Concept drift

cause a new pattern in the data stream. Another common examples are: phishing email filtering, IOT intrusion detection, or financial fraud detection. Here the attackers change their method's characteristics to defeat the model to carry out their malicious activity. This is commonly known as adversarial classification problems and has a significant contribution toward the change of the data distribution. In another example, deliberate business actions can change the distribution of data sets. This entails for example, starting an advertising campaign that draws new types of users or making changes to a website that may (or may not) alter user behavior. Finally, the training data might be accumulated by flawed joining or through a bias which is known as sample selection bias and if that occurs, the model can't work as expected in the deployed environment.

2.4 Principle of concept drift learner

In [Žli10b] authors propose the following paradigm for thinking about concept drift and the judgments that machine learning practitioners must make:

1. Future assumption for Learner's perspective
2. Change of Pattern
3. Adaptivity of learner
4. Model selection

This paradigm can assist with this thesis in considering the options accessible to us when dealing with concept drift in predictive modeling situations. This concept is illustrated in Figure 2.4. The underlying concept is further described in later sections.

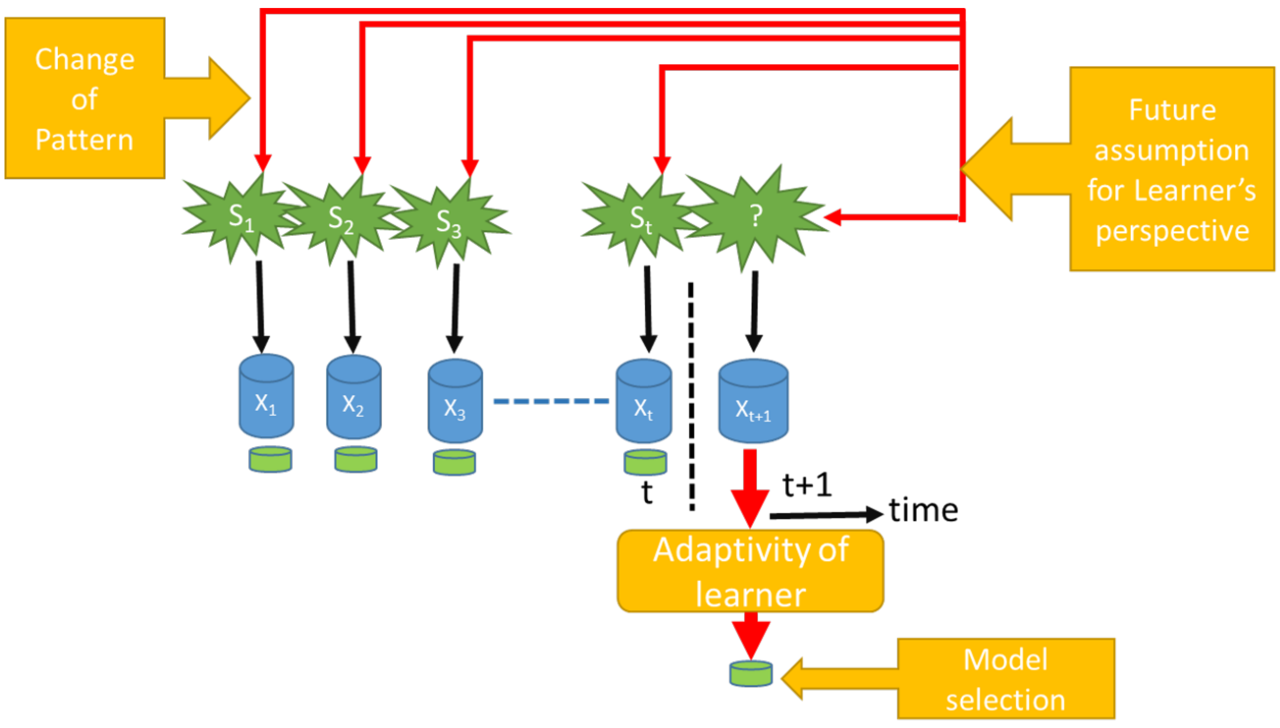


Figure 2.4: The design of concept drift learner

2.4.1 Future assumption for learner's perspective

With future data streams, the learner should deliver the best accurate generalization for the data.The developer must make assumptions about the source from which new data will be created in the future. So far, machine learning practitioners have discovered three distinct types of choices [Žli10b].

First, most people believe that the source's characteristics will not change over time. As a result, the ML model will anticipate data from distributions similar to previous data.
Second, practitioners attempt to estimate the source by statistically assessing the distance between new and past incoming data.
Third, there is a frequently occurring concept drift problems that the future data source is not defined clearly. We can incorporate the trainable prediction rules to predict the future change earlier and use the estimation in incremental learning.

These are a framework and assumptions regarding the data source for determining model drift and the machine learning practitioner's judgments.

2.4.2 Change of pattern:

The data could be altered in any way. It's easy to think about the cases where the transformation has some temporal coherence. Data obtained over a specific period reveal the same association, which changes gradually over time. The fact that the configuration pattern of the data streams changes over time is referred to as change type.

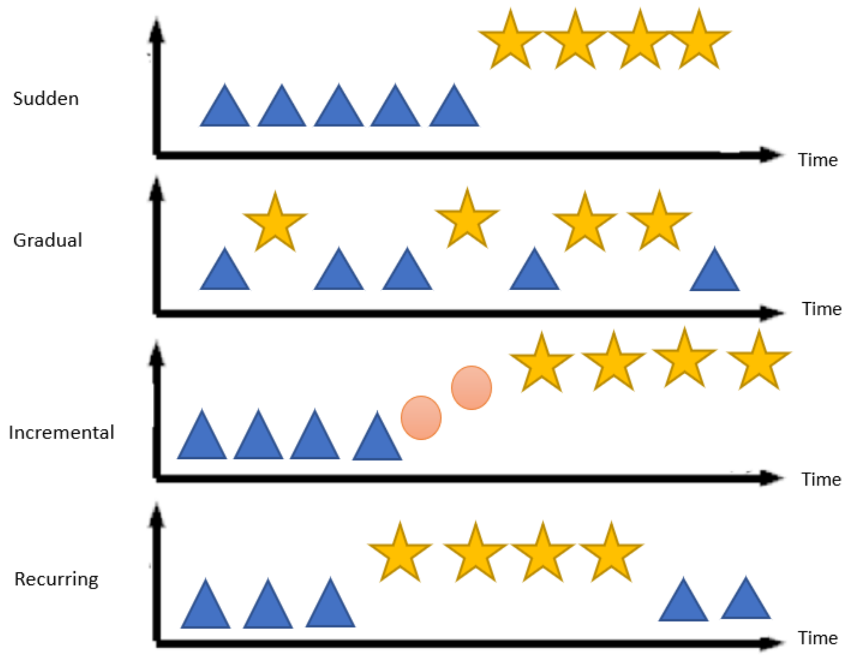


Figure 2.5: Type of changing patterns with time

Many scientists prefers classifying drifts according to how they occur, the cause of the drift, and effect of the changes. As shown in Figure 2.5, concept drifts can be divided into four types: sudden, incremental, gradual, and recurrent. A sudden drift occurs when the rate of change of a feature is exceptionally high, for instance when a new policy is implemented. Similarly, gradual drift refers to a drift with a slow pace of change, such as a piece of production equipment that is slowly wearing out and causing a continuous drop in output part quality. Many fields are likely to repeat hidden contexts. Seasonal phenomena, such as changing the seasons, can also create recurring contexts. Figure 2.5 illustrates the main pattern-changing drift types. It is assumed that data is one-dimensional (1-D). Only the data from one class is depicted.

2.4.3 Adaptivity of learner

There are four primary areas of adaptability that has been identified [Žli10b]:

Base learners Regularly updating the basic model is a helpful first-level intervention. Rather than entirely discarding the static model, the current state serves as the starting point for a fit technique that adjusts the model fit by employing a sample of the most recent historical data.

Learner parametrization Practitioners can consider weighing the importance of input data with age. They apply a weighting that is inversely proportionate to the age of the data in the case of concept drift; this is known as learner parametrization. The most recent data (with a more significant weight) receives more attention, while minor current data receives less attention (smaller weight).

Training set formation (e.g. training windows, instance selection) The development of a training set can be broken down into three parts: training set selection, training set manipulation, feature set manipulation.

Fusion rules of the ensembles The static model can be left alone when using an ensemble technique. A new model is trained to correct the static model's predictions using more recent data associations. This is a boosting type ensemble, with future models revising the forecasts of previous models.

2.4.4 Model selection

According to this study [Žli10b], the generalization error can be used as a significant metric of concept drift learner performance. As a result, the process for predicting the expected generalization error for the target instance at each time step must be established for model selection (training) purposes. The two primary options are the hypothetical evaluation of the generalization error and employing k-fold cross-validation to estimate the generalization error. The choice of error estimation is strongly linked to the future assumption because it is based on the anticipation of future data sources. As a result, the change type and model chosen have a significant impact on learner's adaptability. The speculation about the type of change present in the data must be considered while considering the learner adaptive. Moreover, assumptions about the future data source must be accounted for selecting the model and evaluate them.

2.5 Drift detection

The concept drift detection framework consists of four stages as described below:

- Information extraction: Data streams are divided into segments for processing.
- Modeling: Important features are extracted from the segments for further analysis.
- Statistical analysis: The key features extracted from the previous step are used for statistical analysis to understand the similarity and dissimilarities among the segments.
- Hypothesis testing: To understand the significance of statistical analysis, p values are compared among consecutive segments to make the final decision. This process framework is demonstrated in Figure 2.6

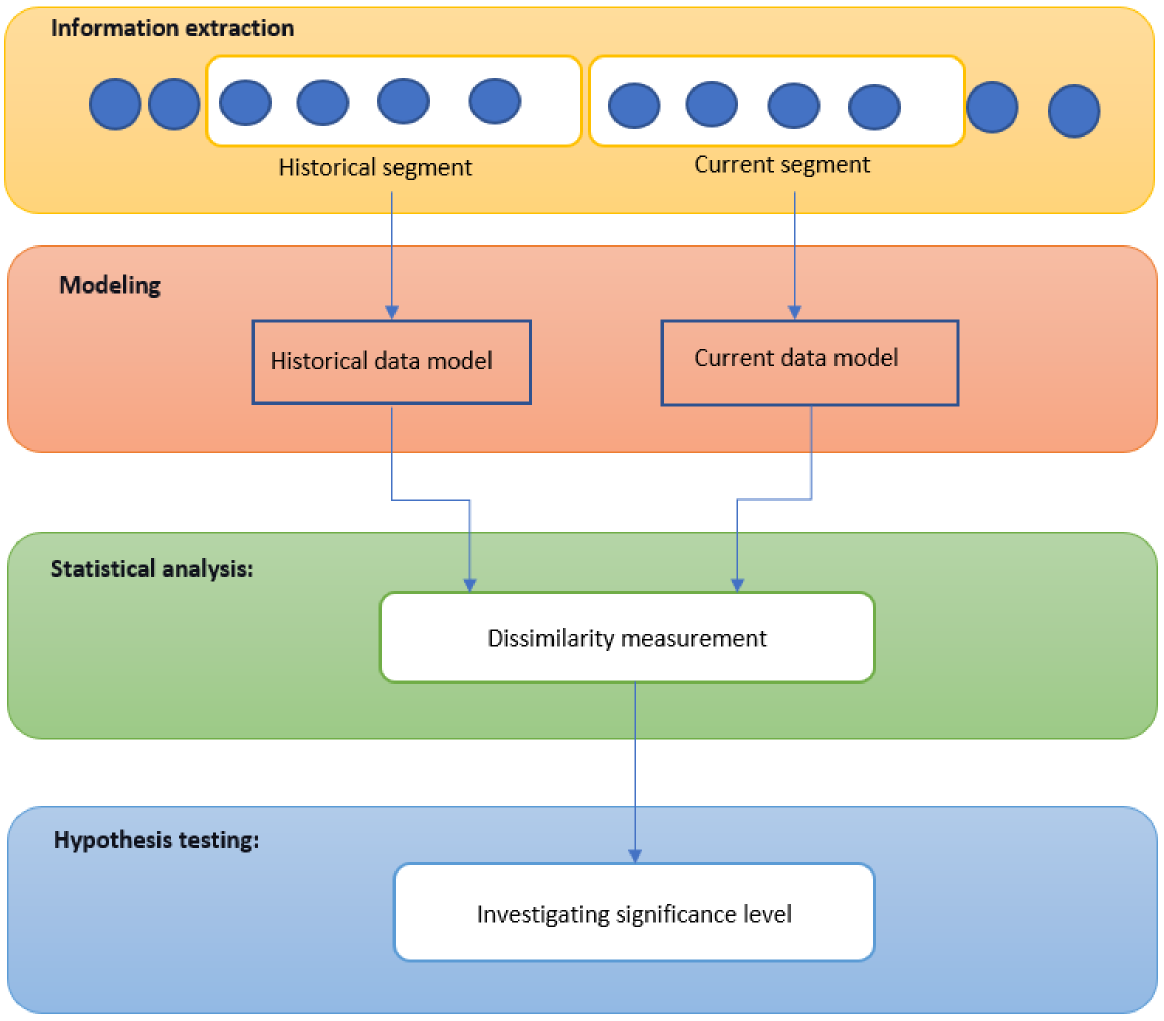


Figure 2.6: Drift detection framework

Sequential analysis [Pag54], statistical process control [Pag54], and distribution monitoring [GMCR04] are standard concept drift techniques. The predictive model evolves by adding new facts to its knowledge base whenever a variation is identified.

Discarding the current model and training a new one from scratch is a simple example of an adaptation process. This process overview is shown in Figure 2.7. Here, the first reference data stream is trained on the reference window. Afterward, the performance metrics are calculated based on the trained model. Next, the new incoming data stream is fed into the trained model, and accuracy is evaluated based on the true ground level and predicted level. If the model's performance decreases significantly, the system considers it as drift and retrains the model. This iterative process goes on till the end of the data stream.

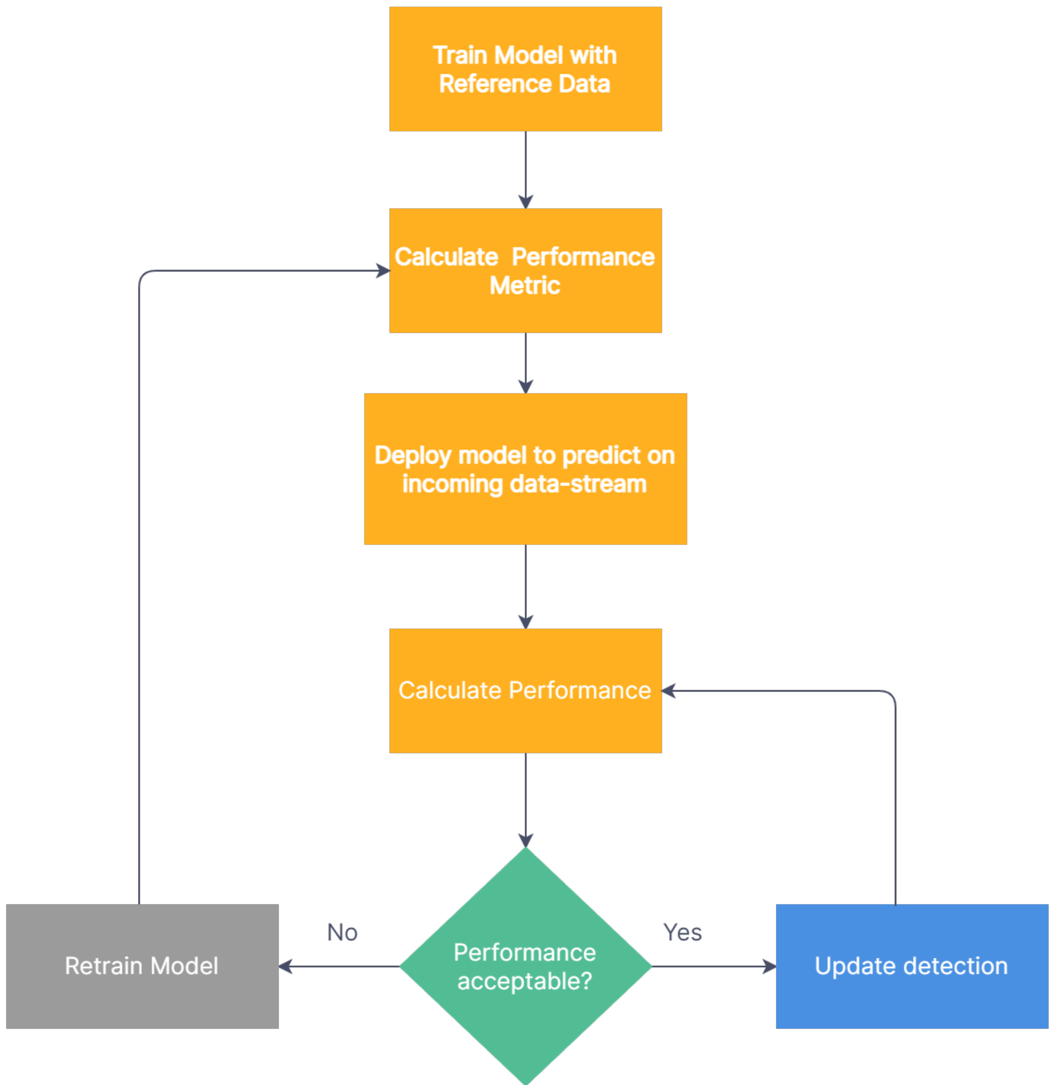

Train Model with Reference Data
Calculate Performance Metric
Deploy model to predict on incoming data-stream
Calculate Performance
Performance acceptable?
No
Yes
Retrain Model
Update detection

Figure 2.7: Traditional drift detection and adaptation process

2.6 Data stream processing with concept drift

There are various types of generic solutions for categorization challenges with concept drift that have been described in the literature. To better understand the tactics utilized to cope with concept drift, many factors including the amount of data input, classifiers, incremental vs. non-incremental learning, and active vs. blind procedures must be taken into consideration.

2.6.1 Ensemble learning

One of the approaches to categorizing resilient solutions for concept drift is the number of classifiers employed to determine whether an ensemble of classifiers or a single classifier is utilized. Concept drifts are generally difficult to manage, and single classifiers are ineffective. One reason is that unless a classifier is retrained after training, its knowledge will not adapt to the changes. The second concern is that retraining the classifier frequently will impact the classifier to lose previously learned concepts, leading to profound performance loss, mainly when the environment changes often. As a result, traditional single classifiers can only be used when the data source remains static, which is inapplicable for data stream processing. Data stream processing challenges have been effectively solved using ensembles of the classifiers. An ensemble of classifiers can be employed to boost the system's decision capability rather than building a new robust and well-adapted classification. An ensemble of classifiers can be utilized to upgrade the software's reasoning capability instead of constructing a new resilient and well-adapted classifier. Multiple papers [Alt07; ZZS08] demonstrate that ensembles of ML classifiers outperform individual ML models' performance in concept drift scenarios.

2.6.2 Incremental Learning vs. non-incremental learning

This categorization takes the data re-utilization into account. Non-incremental learning occurs when a sample is utilized more than once. Whereas incremental learning only uses one instance at a time and cannot be reused.

This method can also be dubbed as online learning. It is based on data stream processing with runtime and memory capacity constraints to enhance computational systems. This method is primarily employed in processing stream, or batch sequence data [DP12].

The advantage of online learning is that each training sample is processed only once by model, eliminating the need for storage or reprocessing. [MY11] argue that online learning is most common in those applications that deal with data streams. In another research, [Kun04] also explains that incremental learning is excellent for stream processing as runtime happens within time constraints and data is not stored anywhere. This helps in dealing with memory capacity restrictions and enhances computational systems. As a result, online learning can be thought of as incremental learning. In addition, the latter refers to learning machines, which are also used to simulate continuous processes. It also handles incoming data in pieces rather than processing each training example individually. One-pass learning is another example of incremental learning, but learning methods requiring access to historical information which cannot be termed incremental.

2.6.3 Active Strategy vs. Blind Strategy

Finally, based on whether or not a drift detection mechanism is included as part of the solution, this discussion splits concept drift solutions into blind and active strategies. Blind techniques don't have any strategies to monitor the input data or performance of the model. This technique just updates the model without observing changes. It claims that the method's detection mechanism is built-in. Blind techniques do not include a drift detection mechanism as part of the detection process. Previous researches on ensemble classifiers use dynamic combination rules to always keep the system updated. For example weights can be assigned to every classifier member based on past performances (i.e. the ensemble classifier members with the highest classification performances will be assigned the highest weights during the final decision). The primary drawback of blind strategies is the computational cost. As the system needs to maintain a constant update-process even if no changes occur, this may result in an increase in processing time by updating the system unnecessarily.

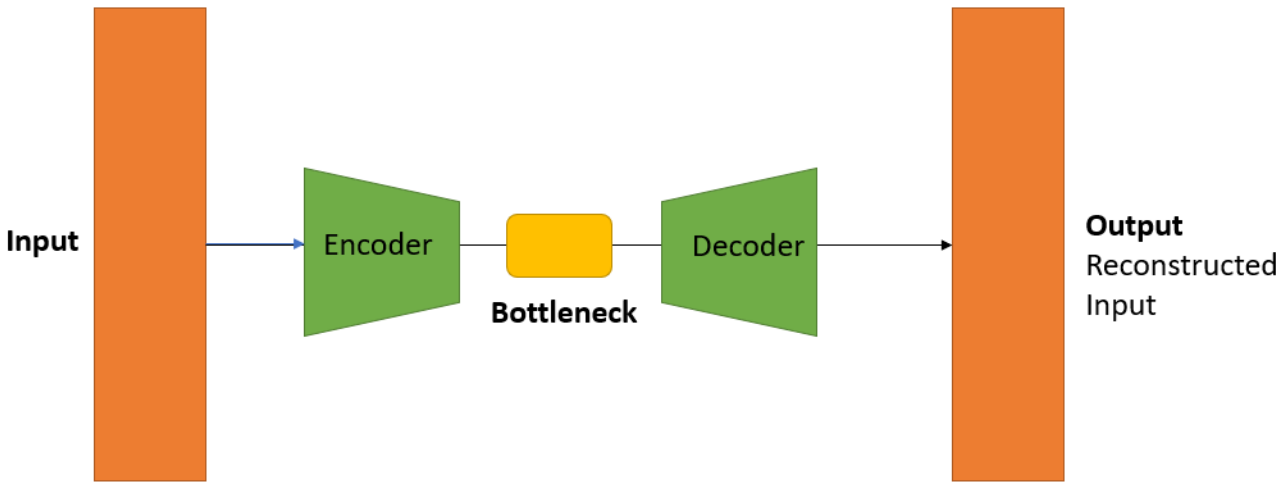


Figure 2.8: Autoencoder Architecture

The alternative to blind strategies is to employ active strategies. The active strategies include an explicit drift detection mechanism. Therefore, only when it detects changed environmental factors, the system modify its knowledge to new inputs saving valuable computational memory.

2.7 Autoencoders

Autoencoder is a generative unsupervised deep learning system [ZP17]. The encoder and the decoder are the two main components of an autoencoder. The encoder's goal is to change incoming data non linearly while capturing the essential information. The decoder is trained to reconstruct the encoder's output as closely as possible to the input data. This neural network uses a tiny bottleneck layer in the middle of the encoder and decoder that includes the latent representation of the input data to reconstruct high-dimensional input data. The number of layers in encoding and decoding is usually, but not always, symmetrical. Figure 2.8 illustrates the autoencoder architecture in details. The autoencoder's latent space is a crucial feature and it is responsible of memorizing the input layer information. An ideal autoencoder should be sensitive enough to decode a latent space accurately while avoiding over-fitting the input data.

When an autoencoder is implemented, we anticipate it to learn the distribution of the input data, and the information will be compared with the reconstructed output. If the input data distribution varies over time, the input and reconstructed output will significantly differ. Due to drift, the input and reconstructed output data encounters data loss. The loss of input and reconstructed output is known as the reconstruction error. The autoencoder will trained to minimize the reconstruction error so that it can learn the new data distributions.

The application of autoencoders are mostly found in data compression and anomaly detection.

3 Literature Review

This chapter conducted a rapid survey on the existing drift detection and adaptation methods on concept drift. Several literature and possible methodologies are studied in details. Based on the literature review, we have divided the concept of drift detectors and adaption mechanisms concerning their characteristics. Under these characterizations most commonly used techniques are highlighted.

3.1 Concept drift detection strategies

Concept drift detection involves sophisticated strategies and there are several methods described in scientific literature. This section presents a literature review of most apropos studies focused on concept drift detection strategies. We summarize several methods to catch concept drift and divided these studies into supervised, unsupervised, and semi-supervised methods.

3.1.1 Supervised concept drift detection methods

These strategies typically need access to the true labels of the whole dataset. Drift detectors, whose strategy is usually focused on error monitoring, are classified according to whether or not drifts are expressly or implicitly recognized.

The biggest group of techniques is PLearner error rate-based drift detection techniques. The goal of these algorithms is to keep track of the progress in the online error rate of base classifiers. An upgrading procedure (drift alert) will be initiated if a statistically significant rise or decrease in the error rate is discovered.

Concept drift detection methods are a type of detector that can detect changes in data stream distributions depending on the performance of a learning algorithm or the statistics of the input data. The authors of [GŽB+14] divided drift detection techniques into three categories:

Methods for tracking two different time frames' distributions

Determine whether two fixed-length sequences are from the same distribution using a confidence level. A fixed reference window, which represents a summary of the previous concept, and a sliding detection window, which contains the cases to be tested on, are typically used in these methods. The idea behind based on null hypthosis. The null hypothesis assumes that the two distributions from two subsequent windows are equal. Based on that, it will perform specific statistical tests on the two windows to check if they are similar or not. If there is a statistical difference between the two windows and there is a significant change, then the null hypothesis is rejected, and the test window will be incorporated to update the change. The most well-known representative of approaches from the window based approach the is ADWIN [BG07] method. It takes a potentially endless sequence of real numbers $x_1, \ldots, x_t, ...$, a confidence parameter $\delta(0,1)$, a confidence parameter (0, 1), and a minimum number of items n to begin searching for changes. ADWIN maintains a variable-length window W of recently observed objects, with the most extended length statistically consistent with the premise "no change in the average value inside the window". When the number of items in the window reaches n, it loops over all of the partitions of W = $W_0.W_1$. It raises alarm and indicates drift, if the average value in one sub-window differs significantly from the other with the confidence level δ.

Another window based approach is SeqDrift2 [BG07]. This method stores samples in the detection window using a random sampling technique. SeqDrift2 uses left and proper repositories to store entries. The reservoir sampling approach ensures that the left repository contains a mix of older and new entries as entries are processed over time. The latest entries are collected in the appropriate repository. When the mean of the two windows differs significantly from each other, an alarm is triggered.

Finally, The Fast Hoeffding Drift Detection Method (FHDDM) [PV16] is also part of sliding window-based methods to find drift spots and use Hoeffding's inequality to build a robust drift detector. This method requires the most recent probability for accurate predictions. This method utilizes an interesting strategy by moving a long and a short window layered on top of each other, stacking FHDDM [PVP18] together. This is an improved version of FHDDM. The longer window lowers false negatives, whereas the narrower window identifies drifts more quickly.

Sequential analysis-based detectors

Change detection techniques in this category are based on sequential probability ratio tests like the Wald test [Wal73]. In the following way these approaches identify changes: Let $x_1, x_2, x_3, ..., x_n$ represent a series of instances. Assumed that the subset of cases $x_1, x_2, x_3, ..., x_w$ comes from an unknown distribution P_0, and $x_w, ..., x_t$ comes from an distribution P_1. The drift is triggered at the time instance w. If the chance of witnessing sub sequences under P_1 is significantly higher (over some threshold) than P_0, the sequential analysis-based detector indicates drift.

CUmulative SUM (CUSUM) [Pag54] is a popular sequential analytic method. If the average of the incoming data differs considerably from 0, it triggers an alarm. It takes a sequence of real values as input, such as $x_1, x_2, x_3, ..., x_t$, and defines $S_{(n+1)} = max\{0, S_n + x_{n-\delta}\}$ at time $t = n + 1$. When $S_n > \gamma$, (the predetermined threshold value) a drift is triggered. Page-Hinkley is a CUSUM variation that considers the difference between observed and average classifier error.

Statistical Process Control-based detectors

Statistical Process Control uses the learning process as main tool, tracks progress of the learning evolution. The model's prediction outcome for each example from the data stream can be correct or incorrect. For a collection of samples, the error of a random variable is derived from Bernoulli's distributions. The Binomial distribution is a generalized form of probability for a random variable that reflects the number of failures in a collection of n occurrences. The standard deviation $\delta_i = \sqrt{(p_i - 1)/i}$ is the probability of observing false p_i for each point i in the series. During model operation, the drift detector handles two registers: p_{min} and σ_{min} are introduced. if $p_i + \sigma_i$ is less than $p_{min} + \sigma_{min}$, at the time determination instant i, the parameters are updated as, $p_{min} = p_i$ and $\sigma_{min} = \sigma_i$.

The Normal distribution can also be used to approximate the Binomial distribution. The requirement for this the observations must be sufficiently large and should have similar mean and variance. The δ-1/2 confidence interval for p_i for n > 30 cases is about p_i , given that the probability distribution is unchanged unless a concept drift occurs. The threshold $p_i + \sigma_i \geq$ $\mathrm{p_{min}}$ +2 δ_{min} has a 95% confidence level for warning, and the threshold $p_i + \sigma_i \geq p_{min} + 3\ \sigma_{min}$ have a 99% confidence level for drift and they can be used as confidence level(α) for triggering the warning due to drift.

The most representative detector under this category is the Drift Detection Method (DDM) [GMCR04]. It is assumed that examples arrive one at a time. As more training instances are observed, the error rate of the learning algorithm p_i and its standard deviation $s_i = \sqrt{p_i(1 - p_i)/i}$ should decrease. Drift occurs when the categorization error has greatly risen. The warning level is attained if $p_i + s_i \geq p_{min} + 2s_{min}$. The drift level is attained when $p_i + s_i \geq p_{min} + 3s_{min}$. If the model hits the alert level at instance x_w, a new context is assumed to begin at x_w.

[BCF+06] devised the Early Drift Detection Method (EDDM) to address the issues where the drift detection was identified with a delay with the DDM [GMCR04] . This method also maintains strong performance with short concept drifts. The approach is based on the fact that the risk of drift is higher when the distance between errors is smaller. As a result, it computes the average distance between two recent errors and detects a drift if a predetermined threshold is exceeded. Because this is more sensitive than the prior method, EDDM achieves early detection in the presence of incremental changes, even when the difference is sluggish. However, when the data contains a lot of noise, its sensitivity can be a big disadvantage.

3.1.2 Unsupervised concept drift detection methods

Unlike the previous solutions, this group of methods aims to deal with concept drift even when just unlabeled data is provided. Several data stream difficulties arise in this situation, with just unlabeled data.

Model-independent approaches

The goal of unsupervised detection is to find drift that isn't labeled. Multivariate statistical testing is a significant methodology used to detect changes in features(X), and P-values are compared to model-independent ways of unsupervised concept drift detection methods. The Hellinger distance measures Hellinger distance-based drift detection (HDDDM) [DP11], which uses an adaptive threshold to infer if drift exists between two chunks of training data. The Discriminative Drift Detector (D3) [GBBC19] employs a discriminative classifier that may be used online without a built-in drift detector. A simple classifier is trained to distinguish between two data chunks and discover drifts in the classifier's performance. The Incremental Kolmogorov-Smirnov (IKS) algorithm [RFMB16] is faster for recalculating the KS statistic for samples that change over time. It enables the KS hypothesis test to be done in real-time with two samples that change over time. The request and reverse strategy [YWP18] was designed as a two-layered hierarchical hypothesis testing framework to detect idea drifts by requesting labels only when necessary, combining supervised and unsupervised approaches. The present unsupervised model-independent techniques have a crucial flaw in that they do not assess the performance of the currently deployed model. Detecting changes in the distribution of input features that aren't important to classification can result in many false positives.

Model-dependent approaches

Recent unsupervised drift detection methods in model-dependent approaches attempt to monitor the classifier output rather than drifts through $P(X)$ with a statistical test. The Margin Density Drift Detection (MD3) technique [SK17] measures the fraction of samples within a margin. It is based on simple margin-based algorithms such as Support Vector Machine (SVM) (margin-based approach). These strategies are only relevant to stated models or ones that can generate probability scores, dependent on the model design [GCGD20].

3.1.3 Semi-supervised concept drift detection methods

Even though unsupervised drift detectors have been thought to be the best option for attempting to deal with entirely unlabeled data, these techniques must guess some configuration inside the underlying data distribution and forced to store clusters in short-term memory. Their actions are based on assessments of resemblance and incongruence. When used in practical situations, these factors could jeopardize machine performance. Semi-supervised approaches, which are often better equipped with a small quantity of labeled data and higher quantity of unlabeled data, may be an attractive alternative for avoiding storing examples and providing more confident decision models. Few methods are worth mentioning. The SUN [WLH12] method splits the streaming data into training and testing datasets. In the beginning, the process trains the model by construing the labeled data. The method incrementally creates a developing decision tree and generates concept clusters. The unlabeled data are labeled according to the majority class of their nearest group. In another way [KRW10], the concept of usage of the ensemble of the classifier is introduced. This method calculates the similarity and dissimilarity between normal and suspicious samples to create a new concept. The suspicious samples are labeled afterward and put together to form new regions. Following a labeling procedure, the ensemble creates a new member classifier and discards the old one.

3.2 Concept drift adaptation

This section focuses on drift adaptation and reaction strategies, which is updating current learning models based on the drift. Some drift adaptation approaches rely on drift detection technologies and use various retraining tactics to handle different drift types better. Others may not use a global drift detection approach but instead update models in real-time based on changes in the distribution of newly available data. There are many popular ensemble learning style, which employs a variety of learners to handle various ideas.

3.2.1 Model retraining

Retraining a new model using the most recent data to replace the old-fashioned model is perhaps the most straightforward approach to reacting to concept drift. Figure 3.1 illustrates the idea of data being updated after drift is detected. System can use external drift detector to identify the point when drift has occurred in the system (explained in Section 3.1 on drift detection). This method frequently uses a windowing strategy to save the most current data for retraining and previous data for distribution change testing. Paired Learners [BM08] uses this strategy by using two learners. One of them is a stable learner and the other is a reactive learner. This method is simple to understand and implement, and it may be used at any point in the data stream. If the stable learner frequently misclassifies examples that the reactive learner correctly classifies, the reactive learner is substituted for the stable learner. A small window better reflects the most recent data distribution, whereas a large window gives more data to train a new model. A trade-off must be made when deciding on an acceptable window size for a window-based technique. ADWIN [BG07] is a standard window scheme method that seeks to solve this problem.

Here, users don't have to guess a fixed size of the windows in advance; instead, it investigates all conceivable window cuts and calculates ideal sub-window estimates based on the rate of change between the two sub-windows. After determining the best window cut, the old data window is removed, and a new model can be trained using the most recent window data.

Many studies have incorporated the drift detection technique in the machine learning algorithm, and machine learning models must be retrained once the drift detectors detect drift. DEML [XW17] improves the standard ELM algorithm by altering the number of hidden layer nodes to manage concept drift. DEML adds more nodes to the network layers when the error rate increase. In this way, the onset of a concept drifts to strengthen its approximation capabilities. FP-ELM [LWJ16] is another ELM-extended approach that responds to drift by adding a forgetting component to the ELM model. A parallel variant of the ELM-based technique [HGL15] has also been developed for high-speed classification under idea drift. OS-ELM [SA16] is an online learning ensemble of repress models that combines ELM with an ordered aggregation (OA) technique to solve the challenge of determining the ideal ensemble size.

3.2.2 Adaptive models

Developing a model that adaptive learns from changing input is an alternative to retraining a whole learner. An adaptive model can partially update itself as the underlying data distribution changes (see Figure 3.2). This strategy may be more effective when drift occurs exclusively in local areas. An online decision tree algorithm called Hoeffding Tree (HT) [HSD01] was proposed in a foundational study [DH00], specifically optimized for high-speed data streams. The Hoeffding limitation limits the number of instances necessary for node splitting.

Another online decision tree algorithm called Extremely Fast Decision Tree classifier (EFDT) [DH00] was proposed in a foundational study, specifically optimized for high-speed data streams. The Hoeffding limitation limits the number of instances necessary for node splitting. Because of various advantages, this strategy has become highly popular: 1) It only needs to process each

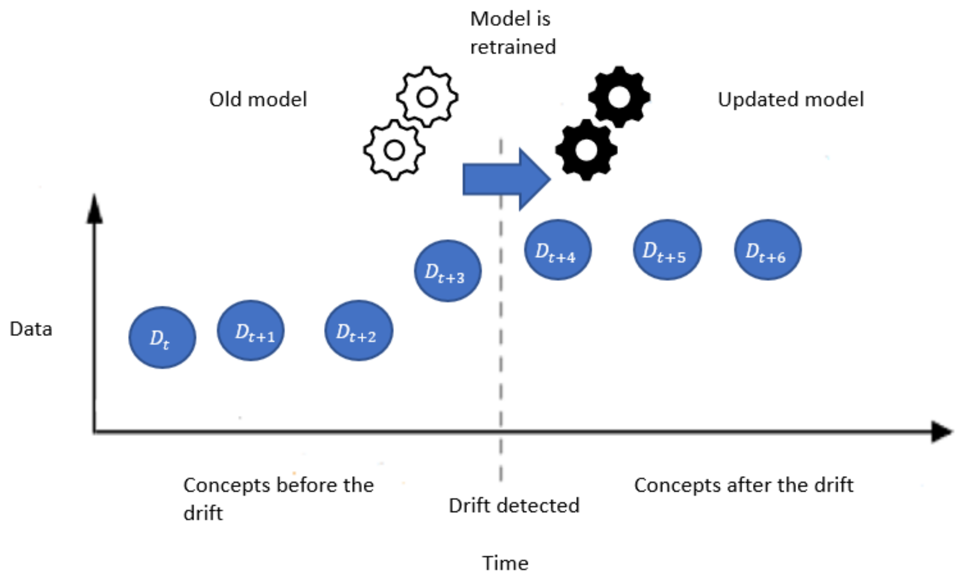


Figure 3.1: Model retraining process in presence of concept drift where new model is trained with latest data and old model is discarded

instance once and does not store instances in memory or on disk; 2) the tree itself takes up very little space and does not grow in size with the number of instances it processes unless there is new information in the data; 3) tree maintenance is very inexpensive.

Two node-level drift detection methods based on observing discrepancies between a node and its sub-nodes have been proposed. The first technique employs the classification error rate, while the second checks the distribution difference directly. When a node detects drift, it transforms into a leaf, and its descending sub-tree is eliminated. Later work [YF12] developed VFDT further by employing an adaptive leaf technique that selects the best classifier from three options: Majority Voting, Naive Bayes, and Weighted Naive Bayes.

3.2.3 Adaptive ensemble

The stream data mining research community has given much attention to ensemble approaches in recent years. Ensemble methods comprise a group of base classifiers with varying types and parameters. To anticipate the freshly arriving data, the output of each base classifier is combined using specified voting criteria. Many adaptive ensemble methods have been devised to address concept drift, either by expanding classical ensemble methods or developing specific adaptive voting rules. A new base classifier is introduced to the ensemble when concept drift occurs (Figure 3.3 demonstrate the process).

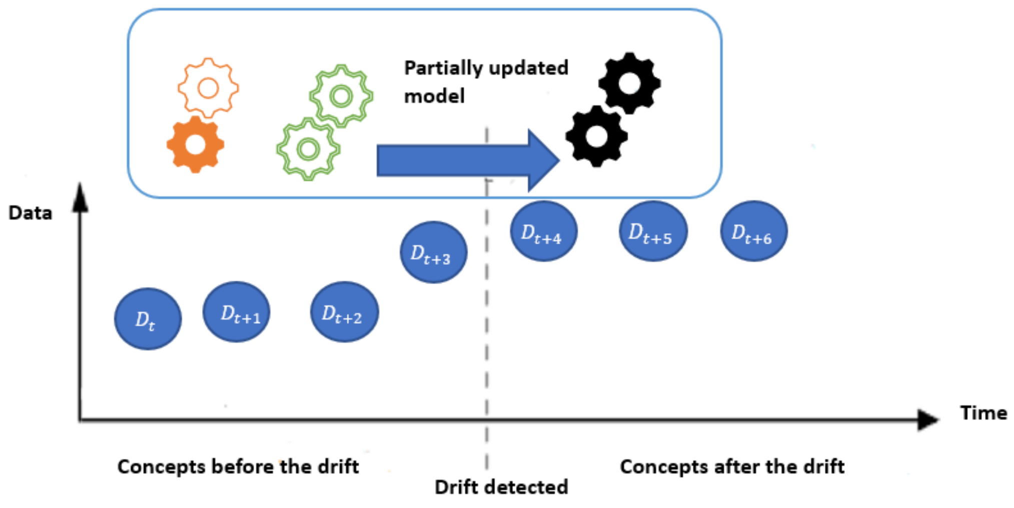


Figure 3.2: Model upgradation process in presence of concept drift where model is partially updated with new data

Bagging, Boosting, and Random Forests are traditional ensemble methods for improving single classifier performance. All of them have been enhanced to handle streaming data with concept drift. [OR01a] proposed the first online version of the bagging method, which employs each instance only once to approximate batch mode bagging. This method was integrated with the ADWIN drift detection system in a later work [BHP10] to address concept drift. The newly suggested Leveraging Bagging approach trains a new classifier on the most recent data to replace the previous classifier with the poorest performance when a concept drift is detected. Similarly, [CZ04] created an adaptive boosting technique that manages concept drift by monitoring prediction accuracy using a hypothesis test, assuming that classification errors on non-drifting data should follow a Gaussian distribution.

Ensemble learning approaches like as bagging and boosting are well-known. They use a combination of learnt base models to improve generalization performance. They have mostly been employed in batch mode so far, and no successful online versions have been offered. [OR01b] describes a simple online bagging and boosting algorithms that, outperform the offline counterparts.

The Adaptive Random Forest (ARF) algorithm was suggested in a recent paper [GBR+17], which extends the random forest tree algorithm with a concept drift detection approach, such as ADWIN, to determine when an outmoded tree should be replaced with a new one. [LWHW15] employs the Hoeffding bound to separate concept drift from noise within trees and does something similar.

Many new ensemble approaches have been created to address concept drift employing novel voting procedures [STM+16] and expanding conventional methods. With a basic set of weighted voting rules, Dynamic Weighted Majority (DWM) [KM07] is an ensemble approach capable of responding to drifts. This method controls base classifiers based on the individual classifiers and the global ensemble's performance. DWM would train a new base classifier and add it to the ensemble if the

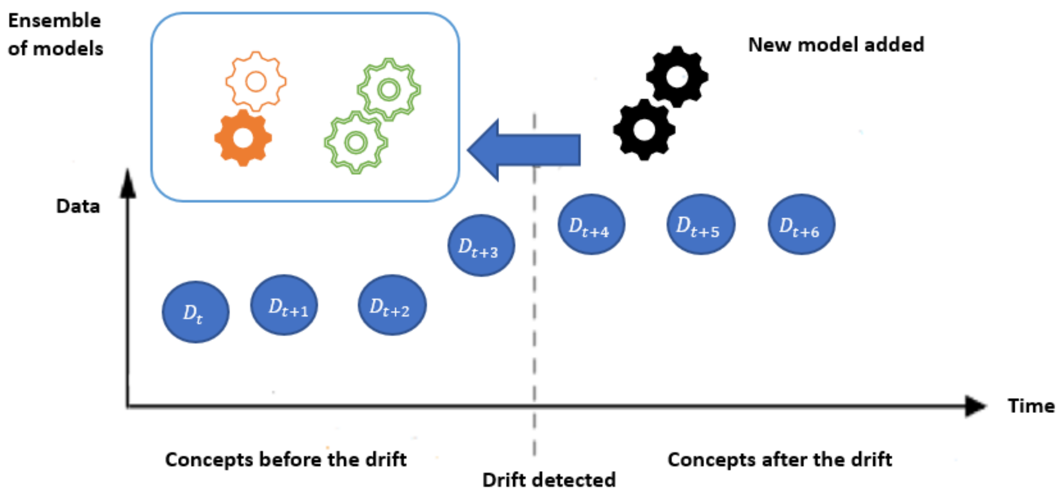


Figure 3.3: Model upgradation process in presence of concept drift where a new base classifier is added to the ensemble

ensemble misclassified an instance. DWM would decrease an instance's weight by a factor if a base classifier misclassified it. DWM removes a base classifier from the ensemble when its weight falls below a user-defined threshold.

The major disadvantage of this strategy is that the adding classifier process may be triggered too often, resulting in performance concerns in some cases, such as when progressive drift develops. Learn++NSE [EP11], a well-known ensemble approach, mitigates this problem by weighting base classifiers based on their prediction error rate on the most recent batch of data. If the youngest classifier's error rate exceeds 50%, a new classifier will be taught using the most recent data. This method has several other advantages: it can quickly adapt almost any base classifier algorithm; it only stores the most recent batch of data, which it only uses once to train a new classifier; and it can handle sudden, gradual, and recurrent drift because underperforming classifiers can be reactivated or deactivated as needed by adjusting their weights. Other voting procedures outside weighted voting have been used to deal with concept drift. Hierarchical ensemble structure [YHH15], short and long-term memory [XXYA17], and dynamic ensemble sizes are only a few examples.

Several studies have developed ensemble approaches for dealing with various types of notion drift. The Accuracy Update Ensemble (AUE2) [BS13] was proposed to be able to handle both sudden and gradual drift. It's an incremental base classifier-based batch mode weighted voting ensemble approach. The ensemble can react fast to sudden drift by re-weighting. Likewise, all classifiers are gradually trained with the most recent data, ensuring that the ensemble drifts slowly.

This thesis aims to take advantage of instance-based techniques while avoiding the computational overhead and build efficient and descriptive methods for detecting and adapting to context drift.

3.3 Related work

In [BSSK21], authors proposed a novel Uncertainty Drift Detection (UDD) algorithm to identify concept drift without true label. This methodology can be used both for classification and regression tasks. The main idea behind this approach is to identify drift by monitoring the uncertainty estimation of the deep neural network in conjunction with monte carlo dropout. The uncertainty change with respect to time is detected with ADWIN and notified to the system by ADWIN to retrain the predictive model. UDD doesn't only detect drift based on input data but also incorporate the effect of feature's on the classification model.

Explanation of Uncertainty in Neural Networks w.r.t concept drift:

Previous research shows that test error in neural networks is highly intertwined with the it's uncertainty [RCNW18]. In neural networks, authors have used this uncertainty as a proxy of error rate which is utilized as input to ADWIN to find drift. Generally, the common drift detector such as DDM or EDDM can only accept inputs originating from a Binomial distribution. Since the uncertainty has a different distribution characteristics, ADWIN is more suitable canditate for such cases. Once drift is detected by the UDD, it is required to retrain the predictive model. To get a good estimation of uncertainty, authors have incorporated Monte Carlo dropout in the neural network. The UDD algorithm is mentioned in Algorithm 3.1.

Algorithm 3.1 UDD: Uncertainty Drift Detection

```
Input: Trained model M; Data stream T ;
Training data D
Output: Predict output for a given t
repeat
Receive incoming instance $x_t$
$y_t, U_t \leftarrow M.predict(x_t)$
ADD $U_t$ to ADWIN
if ADWIN detects change then
    AE.train (D ∪ T)
    Return Drift
end if
Until T end
```

In [GBBC19], the author proposed a discriminative classifier (D3) along with ADWIN for concept drift classification . The idea of D3 is to detect Drift in an unsupervised way. D3 can be used along with any online classifier without any explicit drift detector specified. The drift identification depends on the discriminative classifiers's performance. Also, the advantage of D3 is that it can identify drift continuously without measuring any distribution.The concept of D3 is as follows:

D3 has a fixed size window of W which consists of reference and part of a new data with respect to reference data. The new samples are added in the right of the window. Before starting any analysis, the process needs to wait until the window is full. The reference and newer data are labeled with stack variables once the window is full and then they are fed into a logistic regression model for classification to check if the new incoming data pattern is changed or not. To identify drift, area under the curve(AUC) of the classifier is monitored. If there is a significant change in the AUC value and this value exceeds the programmed threshold, it is detected as drift. On the other hand,

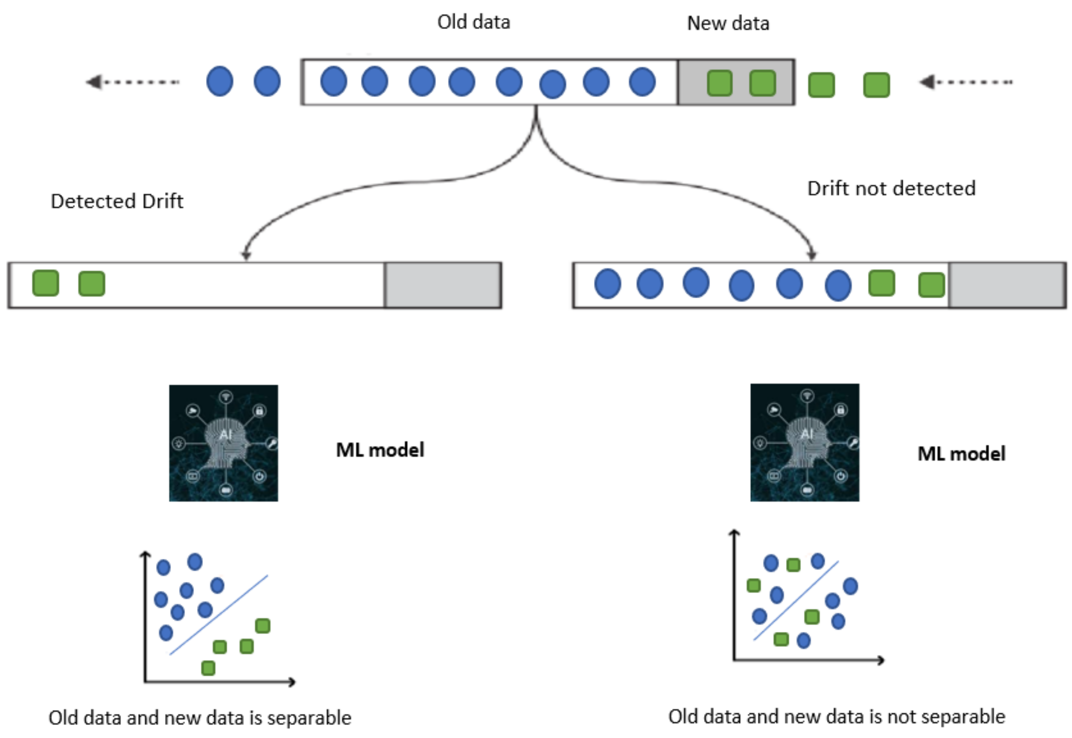
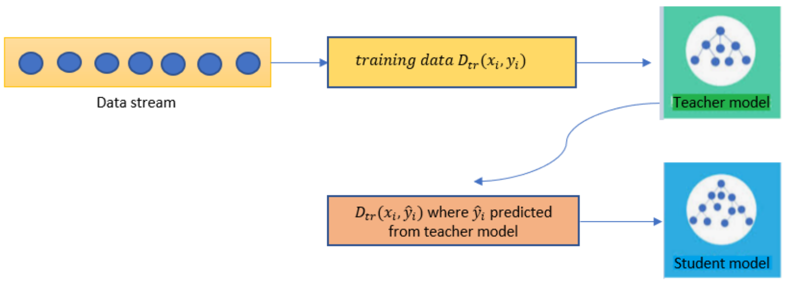

Old data
New data
Detected Drift
Drift not detected
ML model
ML model
Old data and new data is separable
Old data and new data is not separable

Figure 3.4: Illustration of workflow of the D3 algorithms

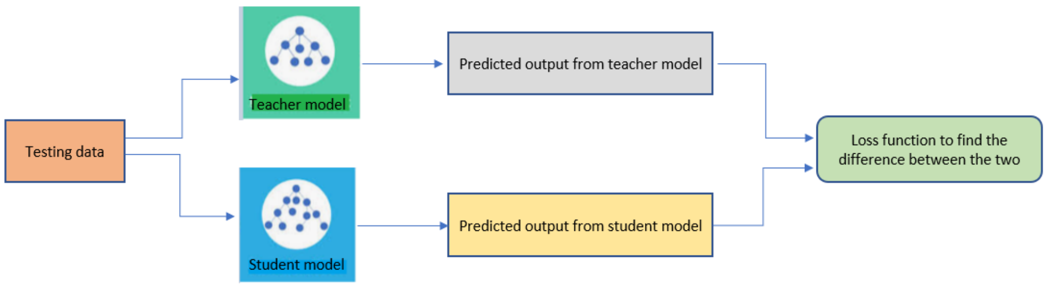
if the AUC values stays under the threshold, it is considered as no significant change within the dataset, and therefore no drift detection information in intimated. When a drift is identified, the reference data are removed from the window and new data are added. In case of no drift is detected, the window is shifted towards the left. This workflow can continue as long as the stream generates the data. The workflow of D3 method is depicted in Figure 3.4.

With a student-teacher (Student–Teacher Unsupervised Concept Drift Detection (STUDD)) learning method, the authors [CGBT21] provided another novel approach to concept drift detection. This method only requires the feature vectors X of the testing samples from the environment. The authors refer to the suggested method as unsupervised because it is not dependent on the labels of the target variable y. The main idea of this study focuses following four primary steps:
1. Developing the teacher as a secondary model.
2. Developing a student model as primary model to replicate the teacher model's behavior
3.Compress the large ensemble (the teacher) to a compact predictive model (the student).
3. Deploy the primary teacher model and detecting concept drift.

The available training instances are denoted by $D_{tr}(x, y)$ is to train the main classifier T, where $D_{tr}(X, y)$ represents an initial batch of training cases. The forecast of upcoming future instances in the stream D, is referred to as X_{new}. Due to assumed incremental nature of the model , T is updated as soon as new labels are obtained. Using Student–Teacher method, it is assumed that the corresponding labels of X_{new} are not immediately available after the prediction was made. Therefore the method that rely on tracking the T data losses becomes ineffective. A clever solution to this problem is to use student-teacher (ST) learning method. T is named as the teacher model. A second predictive model, S, the student is created that can be trained to closely represent teacher's T behavior. This can be accomplished by first obtaining the predictions $\hat{y}_i$,of T for $D_{tr}(x, y)$. Then S is trained using the same observations used to train the teacher with a new dataset S. As a result, the predictions of the teacher replaces the target variable. The ST method working principle is depicted graphically in Figure 3.5


Data stream
training data $D_{tr}(x_i, y_i)$
Teacher model
$D_{tr}(x_i, \hat{y}_i)$ where $\hat{y}_i$ predicted from teacher model
Student model

(a)


Testing data
Teacher model
Student model
Predicted output from teacher model
Predicted output from student model
Loss function to find the difference between the two

(b)

Figure 3.5: The workflow of Student-Teacher approach

4 Study Design

This chapter highlights the research question that this thesis attempts to answer. It also explains the objectives of the research and research methodology used to address these study questions.

4.1 Research question

The research questions(RQ) are framed and described in detail in this section. Solution to three main research questions are explored in the scope of this thesis.

RQ1 How can you accurately detect real concept drift while separating it from false alarm?

Arbitrary changes in distribution usually interfere with machine learning performance on streaming data. Classification boundary transformation, also known as real concept drift, is the primary source of classification performance drop. Existing drift detection methods, such as two-sample distribution tests and monitoring classification error rates, have inherent limitations, such as the inability to distinguish between false alarm and real concept drift. The drifts that do not affect the classification boundary will result in unnecessary model maintenance, computational cost, and statistical power. Therefore, we have considered detecting false alarms, which don't affect model performance degradation.

RQ2 How does the proposed drift detector's performance compare to other supervised and unsupervised drift detection algorithms?

There are a few challenges to incorporating drift detectors in machine learning algorithms. Most research counts on the error rate for their identify drift in their machine learning model in a supervised way. But in many real-world scenarios, supervised drift detectors are not along sufficient as true label availability are rare. Over the years, many unsupervised drift detection mechanisms have been proposed, but these methods have their own limitations. Therefore, we need a new strategy to evaluate those unsupervised drift detectors in a meaningful way with respect to different real-world scenarios.

RQ3 How well the proposed method adapts to the data stream in the classification task with the presence of concept drift?

Processing streaming data under non-stationary conditions is more complex than processing static data. Traditional offline machine learning (ML) models cannot cope with concept drift, necessitating the creation of online adaptive analytical models that can adapt to both predictable and unpredictable changes. Adaptive machine learning algorithms are suitable options because they can process changing data streams while responding to any notion deviations.

4.2 Objective

In this section the objectives of the research questions that are being attempted to be solved is briefly explained.

Objective 1 Create a novel create drift detection method to detect concept drift and avoid false alarms

This objective refers to the first research question. The existing supervised concept drift detection algorithms can be a good starting point for monitoring drift. They observe prediction output instantly and are computationally efficient. They can witness true concept drift. Nevertheless, they cannot adequately define the drift and require true labels. On the other hand, unsupervised drift detection methods such as two-sample tests can notice changes in distribution without employing true labels. However, these techniques have severe flaws, such as they are prone to false alarms. They can't tell the difference between concept drift i.e., the changes in the classification border, and virtual drift, i.e., changes in the distribution that don't affect classification results. In this context, this work needs to investigate a robust deep learning-based drift detection method that aims to detect true concept drift while disregarding false alarm or virtual drift.

Objective 2 Define a strategy comparing different drift detection algorithms

Real-world datasets frequently contain various drift types and lack specific indicators for start and end of drifting points, making a proper evaluation of drift detection systems difficult. Artificial drift can be used to understand better how to drift detectors work in various settings. The concept drift detectors need to be tested without employing classifiers in this experiment. The comparison of these drift detectors needs to analyzed based on how often false alerts are given when there is no change and also compute how long it takes for the system to responds on changes once they have occurred. Finally, it must be estimated how often a warning was not received when there was a drift present in data stream. These experiments are only possible when the drifted points are known from before.

Objective 3 Proposed new adaptation strategies combining with drift detector

Concept drift adaption strategies after detecting drift need to be examined. To deal with the observed drifts and preserve good learning performance, an effective drift adaptation method should be devised. To solve the performance limitations of current concept drift approaches, a unique drift adaption method can be derived. The proposed framework should be tested against real-world data to understand it's practicality.

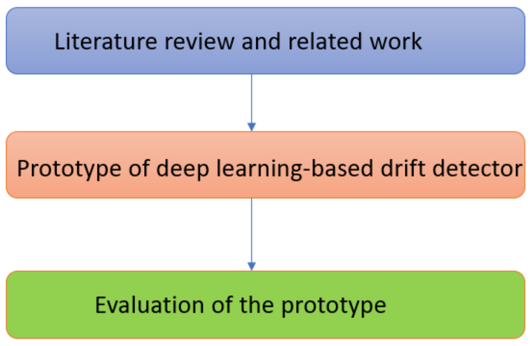

Literature review and related work
Prototype of deep learning-based drift detector
Evaluation of the prototype

Figure 4.1: Research methodology

4.3 Research Methods

This section explains the research procedure used to answer the research questions mentioned before. Figure 4.1 gives an overview of that research process performed. We started by conducting a literature review and related work (Chapter 3) to summarise the state of art algorithms, challenges when working with data streaming with machine learning and possibility of betterment. Based on the results of these steps, a deep learning-based drift detector algorithm was proposed and implemented. Afterward, a concrete analysis was developed and evaluated on real-world datasets.

4.3.1 Literature review and related work

We conducted a thorough review of concept drift-related papers in the initial phase from Google Scholar. Many relevant literature were obtained related to keyword ”concept drift”, ”data stream” and ”non-stationary environment”. The summary of the state of the art and the most salient methods are described in Chapter 3.

4.3.2 Prototype of deep learning-based drift detector

From the literature review, it was evident that the application of ML models such as neural networks, random forest, etc. is widely used for drift detection. Still, the application of autoencoder was very rare. So, I have decided to incorporate an autoencoder-based drift detector for identifying drift. The details of the main underlying concepts are described in Chapter 5.

4.3.3 Evaluation of the prototype

Finally, based on the autoencoder algorithm designed to detect drift, I ran several experiments to validate the concept. There were 3 separate experiments carried out that looked into several aspects like comparison of performance against existing methods, adaption capability and accuracy in terms of identifying drift. The details of the experiments carried out, and the results are compiled in Chapter 6.

5 Proposed Methodology

Standard drift detection techniques like ADWIN , DDM, and Page-Hinkley are not suitable drift detectors for scenarios when the labels are costly. Their application is restricted because these algorithms identify drifts associated with a change in the predicted error rate (and therefore require true labels). There are several possibilities for dealing with concept drift when restricted label availability (depicted in Chapter 3). This study presents a unique method for detecting drifts that do not require access to true labels. Here, an unsupervised autoencoder is used to detect drifts.

5.1 Main idea

Autoencoder is a generative unsupervised deep learning system. We have already discussed about autoencoder in the Section 2.7. It has been demonstrated in [ZP17] that the reconstruction error relates to anomaly detection.

The reconstruction error can be a good indicator of concept drift. If the data distribution change with time, the new incoming data instances will be different from training data instances. The autoencoder will have trouble reconstructing the instances, and hence the reconstruction error will be high. If the reconstruction error is higher for a prolonged time, we can assume this as an indication of drift. We have extended this idea and proposes in this study that model reconstruction error may be used as a proxy for error rate and, as a result, should be a valuable predictor of concept drift.

The encoder's goal is to change incoming data in a nonlinear manner while capturing the most significant information [WYZ16]. Since, the encoder captures the most significant features, we can extent this to the idea that autoencoder can ignore the changes in distribution for those feature that doesn't contribute in the model performance degradation in predictive models. Therefore, it can avoid the virtual drift.

To test this idea, the following strategy has been developed:

First, the reconstruction error is measured for each data instance by the autoencoder. Afterwards, the ADWIN drift detector is used to identify change by using the reconstruction error value as input.

5.2 Construction of the autoencoder

Before we delve further, lets have a look at the detail mathematical notion of autoencoder. The architecture of autoencoders is considered with $L + 1$ layers and value of the j-th neuron in the l-th layer acquired for input data x_n is denoted by $y_j^{(l)}(x_n)$. Using the Equation 5.1 [JRA20], the values of neurons in the l-th layer can then be calculated.

(5.1) $y_j^{(l)}(x_n) = \sigma\left(\sum_{i=1}^{N_{l-1}} w_{ij}^l y_j^{l-1}(x_n) + b_j^l\right), \ where\ l = 1, \ldots, L.$

The weights of synapses between the (l-1)-th and l-th layers are $w_i j^l$, the size of the l-th layer is N_l and the biases are b_j^l . An activation function is denoted by $\sigma(z)$. In autoencoder, several different forms of activation functions can be used. The most frequent one is used in this study, which is the sigmoid function, which is defined in Equation 5.2.

(5.2) $\sigma(z) = \frac{1}{1 + e^{-z}}$

The output layer has the same size as the input layer, i.e. $N_L = N_0 = D$ and the input layer, $y_j^0 = x_n^j$, is the 0-th layer. The middle layer should be the smallest because no sparsity constraints are applied during autoencoder learning, i.e.

(5.3) $D = N_0 > \ldots\ldots > N_{L/2} < \ddot{\ldots}\ldots < .N_L = D$

By minimizing a cost function, the autoencoder is taught like a traditional feed-forward neural network. The cost function obtained for the x_n data element is $C(x_n)$. The weight $w_{ij}^{(l)}$, for example, would be changed in each step using Equation 5.4 [JRA20].

(5.4) $w_{ij}^{(l)} = w_{ij}^{(l)} - \eta \cdot \frac{\delta C(x_n)}{\delta w_{ij}^{(l)}}$

where η is the rate of learning. For biases, the similar formula applies. In practice, rather than single data instance, learning is frequently done on mini batches of data. Assumed that the mini batch M_t is made up of B subsequent data, defined as:

$$M_t = \left(x_{tB}, \ldots\ldots, x_{(t+1)B-1}\right)$$

The arithmetic average of gradients for all mini batch data are used to calculate the neural network parameters.

(5.5) $w_{ij}^{(l)} = w_{ij}^{(l)} - \eta \cdot \frac{1}{B} \cdot \sum_{m=t_B}^{t_{(B+1)-1}} \frac{\delta C(x_m)}{\delta w_{ij}^{(l)}}$

Next, the reconstruction error has been considered. The cost function of the reconstruction error is defined in Equation 5.6

$$(5.6)\quad C_{RE}(x_n) = \sqrt{\left(\sum_{j=1}^{D}(x_n)_j - y_j^{(L)}(x_n)\right)^2}$$

This reconstruction error C_{RE} will be monitored to identify drift in the data stream.

5.3 Idea of Adaptive Windowing method for concept drift detection

Input: In general, ADWIN accepts numeric value which is related to concept change and can be used for analysis. Here we have used the reconstruction error as input to the ADWIN.

Importance of ADWIN in this context: ADWIN has been chosen because it can function with any real-valued input and requires no prior knowledge of the input distribution. Alternative drift detection techniques, such as DDM or EDDM, are built for data with a binomial distribution and are thus they might be inapplicable to taking reconstruction error[AC15] as input. Therefore, we choose ADWIN, as it is more relevant in context of identifying change in the reconstruction error distribution [BG07]. We have already discussed about working mechanism of ADWIN in chapter 3. ADWIN is more robust in terms of false positive and false negative for detecting concept drift than other exiting drift detectors[BG07]. It has unique mechanism for adjusting window size to provide quick response by shrinking or expanding the windows.

Work mechanism of ADWIN in this context ADWIN [BG07] utilizes the idea of confidence value δ which is between range of 0 to 1 to identify drift in between two sub windows. It monitors the reconstruction error between two consecutive windows and declares when retraining of autoencoder is necessary.

5.4 Implementation details

In this section the AEDD methodology is explained in brief. Two dependent algotithms are employed in the proposed methodology. The inputs to Algorithm 1 are the following: $W1$ - denotes the current window, which consists of existing test data, and $W0$ - indicates the reference window consisting of reference data steam. The function will return whether or not drift is detected.

We train initially an autoencoder AE with a dataset of the reference window and calculate reconstruction error (Equation 5.6) RE_t for each instance in the current window. This calculated reconstruction error is the input to the $ADWIN$. If $ADWIN$ finds the statistical change of distribution in the incoming data stream, it informs the algorithm to retrain the autoencoder AE with the current data. Otherwise, it will keep the autoencoder AE as it is and return that no drift is detected. Once the algorithm finishes calculating the reconstruction error for all the data instances in the current data frame (see the Figure 5.1) the current data frame slides towards right to discover the new incoming data.

Symbols:

Algorithm 5.1 AEDD: Autoencoder based drift detection

```
**Input**: Trained model AE; **Data stream** of current window $W1$;
Data stream of previous window $W0$
**Output**: Prediction whether drift is present or not
**repeat**
Receive incoming instance $x_t$ from $W1$
$RE_t \leftarrow AE.predict(x_t)$
ADD $RE_t$ to $ADWIN$
**if** $ADWIN$ detects change **then**
    $AE$.train $(W0 \cup W1)$
    Return Drift
**end if**
**Until** $W1$ **end**
```

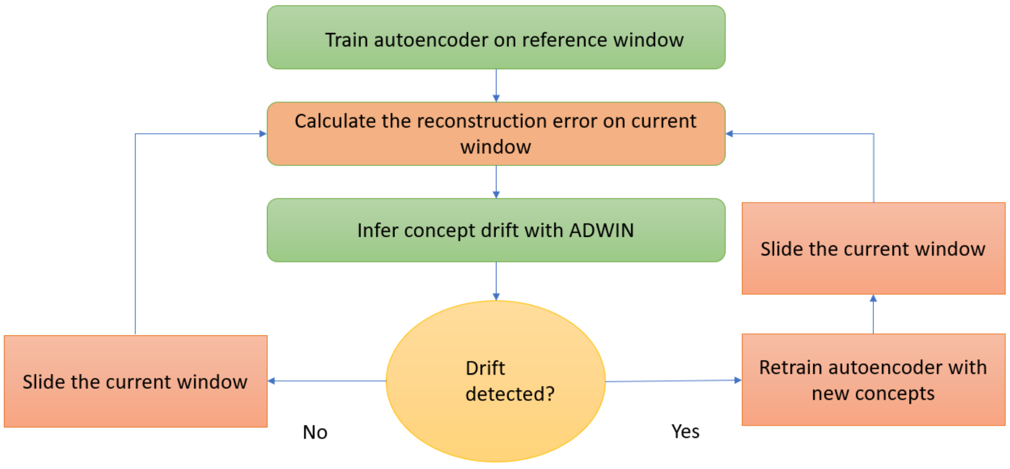

Train autoencoder on reference window
Calculate the reconstruction error on current window
Infer concept drift with ADWIN
Drift detected?
Slide the current window
No
Yes
Retrain autoencoder with new concepts
Slide the current window

Figure 5.1: Workflow of the proposed AEDD methodology for drift detection

RE_t	reconstruction error
x_t	input feature of the data instances
$W0$	reference window
$W1$	current window
AE	autoencoder
$ADWIN$	adaptive windowing

Base learner We are mainly working on classification task. Therefore, we need a base model for the classification of the incoming data stream. This can be any type of classifier that can be trained and tested on data. We have considered an adaptive tree classifier(the Hoeffding tree) [HSD01] as a base classifier in our investigation. The Hoeffding Tree is competent of dealing with an enormous data stream, and it can endure high dimensional data. It allows for incremental updates, which is excellent for our tests because it allows us to compare it to other drift detection systems that employ streaming data.

Next, the Algorithm 5.2 implements the drift adaptation strategies. First 10% of the data in the stream is used for the reference window $W0$ and the subsequent 10% of the following data stream is gathered in the current window $W1$. We train our base learner $base\_classifier$ for classification with the data stream from the reference window($W0$). Then we check if there is any drift in the data of the current window($W1$) with the help of Algorithm 5.1. The current window slides towards the further right if no drift is found. This workflow continues as long as the stream generates data. But if the algorithm identifies drift, base learner need to be updated with the new concepts. It requests the data label of the corresponding data in the current window and updates the base learner($base\_classifier$) incrementally. The workflow of the proposed adaptive AEDD methodology is illustrated in Figure 5.2.

Algorithm 5.2 Online adaptation with AUTOENCODER based drift detector

```
Initialize window W where |W| = w(1+p)
Initialize Unsupervised Neural Network i.e autoencoder AE
Train the base-classifier with initial 10% of data
for (X,y) in D .class label (y) is not used do
    if W is not full then
        $W \leftarrow W \cup X$ .i.e., add $X$ to the head of $W$
    else
        $W0 = w$
        $W1 = wp$
        Trained model AE with W0
        if AEDD(AE, W1, W0): then
            base-classifier.retrain(W1)
            drift=true
            number-of-drift =number-of-drift + 1
            remove w element from W
        else
            drift = false
            remove wp from W
        end if
    end if
end for
```

Symbols:

W fixed window size

D data stream

$AEDD$ function name to the Algorithm 5.1

$base\_classifier$ baseline model

p percentage of new data

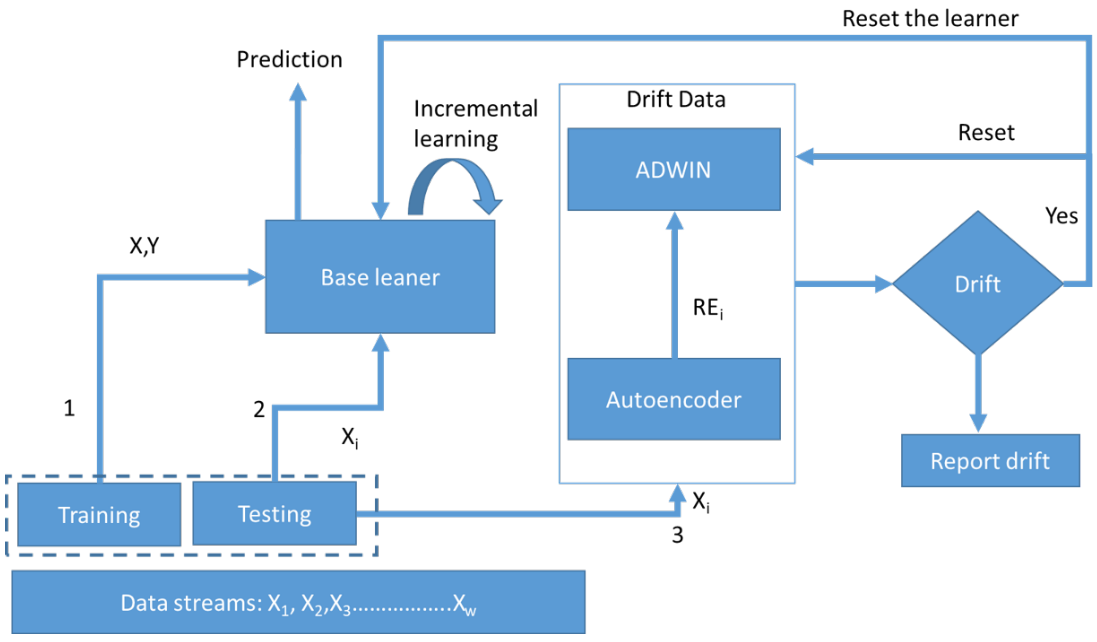


Figure 5.2: Workflow of the proposed adaptive AEDD methodology

6 Experimental Results

This study conducted extensive experiments to compare AEDD with other competing benchmark procedures for evaluation purposes. We used a publicly available dataset in our investigation. The thesis aims to achieve higher accuracy for the predictive model in a concept drift environment. The experimental analysis and results of the proposed AEDD approach to these datasets are presented in details. The first part of this chapter helps to understand how efficiently the proposed framework's drift detection works. This is done by artificially inducing drift explicitly in the real-world dataset. The primary motivation behind explicitly causing drift within the dataset is that we don't know where the drifted position within the real-world datasets is. So, comparing different drift detectors will be complicated. For example, even if there are false alarms, there is no way to identify them because the true drifted position are unknown. The second motivation is to present the adaptation strategy within the AEDD results and its application in the real-world dataset to demonstrate its practicality. Finally, this chapter will illustrate the result of different drift detectors' performance and final accuracy after combining drift detection and drift adaptation strategies together.

6.1 Experiment goal

Following are the goal of the experiment:

- Research question 1 (RQ1) is addressed primarily in this experiment. Unsupervised drift detection is a challenging task for many real-world applications. Since these methods are prone to false alarms, it can lead to unwanted model retraining. The objective of this experiment is to approach the 1_{st} research question to identify genuine concept drift, ignoring the false alarms when the true labels are unavailable. So, with experiment 1, we want to demonstrate how proficiently AEDD can handle false alarms by creating different testing scenarios.
- In the second experiment, we will try to approach 2_{nd} research question (RQ2), which is related to comparing different drift detectors. The main goal is to understand how well our proposed model can perform as a robust drift detector compared to other state-of-the-art algorithms. We will be using performance metics (discussed in Experiment 2 in details) to evaluate unsupervised drift detectors in a meaningful way concerning real-world scenarios.
- The 3_{rd} experiment concerns the performance of the proposed AEDD-based adaptive algorithm with other existing adaptation algorithms. It is geared towards solving the $_{\text{rd}}$ research question (RQ3) by comparing the performance of the proposed AEDD and different state-of-the-art drift adaptation algorithms.
- The $_{\text{th}}$ experiment is the visual representation of the working mechanism of the proposed AEDD for better understanding. Here, the change of width of ADWIN window and fluctuation of the reconstruction errors by application of AEDD are demonstrated.

6.2 Parameter setup

- The Autoencoder architecture is defined and checked whether the bottleneck of the encoder is smaller than the input dimensions of the dataset. The Autoencoder consists of four fully connected layers. The encoder consists of the first 2 layers, and the decoder consists of the last two. Relu is an activation function in the encoder, and the sigmoid activation function is used for the decoder.
- At first, the parameters for the AEDD-based drift detector need to be defined for the experiment. Following are the details about the parameters for the Autoencoder: The number of epochs is 100, batch size of the Autoencoder is 64. Early stopping is used as a regularization technique. The value for ADWIN is set to 0.002, and the window size is 64. Here, Adam optimizer is used. Once ADWIN detects drift, AEDD is retrained with 100 epochs with a batch size of 32, and the ADWIN window also drops the previous reference data.
- Parameter optimization for AEDD requires the simultaneous ADWIN algorithm to detect one drift on a given validation data set, resulting in a approximate value for the delta. Assume there are no drifts in the validation data while the delta is set to the initial value. Once the validation data is considered free of drift, the delta is set to the default value of 0.002, defined by scikit-multiflow.
- On a system with an i7-8700 processor and 16 GB of memory, the suggested framework was developed using Python 3.9 version by extending the Scikit-Multiflow [MRBA18] framework.

6.3 Dataset description

This section gives a supervised experimental evaluation of the AEDD method on static datasets produced by concept drift. By exploiting the position of drifts in these datasets, researchers can better grasp the different approaches' drift detection abilities in a real-world context.

6.3.1 Datasets

Three datasets from the UCI machine learning library [Lic+13; SK17] were chosen. Bank data is related to direct marketing campaigns of a Portuguese banking institution used for classification task [Lic+13]. Wine data results from a chemical analysis used for quality classification[Lic+13]. Phishing dataset contains data about malicious web pages [SK17] . These were pre-processed to have only numeric and binary values, normalized in [0,1] (refer to Table 6.1). The multi-class datasets were reduced to binary class issues, and the data instances were randomized (sequence of the data changed at random) to remove any existing unwanted concept drifts.

Data Set	Instances	Attributes	Problem
Bank	45211	48	Classification
Phishing	11055	46	Classification
Wine	6497	12	Classification

Table 6.1: Description of the dataset

Data	Train accuracy	Test accuracy	Detectability test (accuracy)	False Alarm (test accuracy)
Bank	85.4	85.2	67.45	85.5
Phishing	90.9	90.7	87.29	90
Wine	99.9	99.7	81.79	99.7

Table 6.2: Effect of performance on predictive model due to artificial drift

6.3.2 Drift injection

Experiments with artificial drift are carried out to understand better the relative behaviour of drift detectors in various settings. This is accomplished utilizing a similar strategy proposed by this study [Žli10a]. According to the author [Žli10a], the drift induction approach permits the insertion of concept drift in static datasets at a precise moment in the data stream. This enables managed drift analysis while maintaining the attributes of the application areas from which the sample was created. To eliminate any undesirable concept drift and to prepare the dataset for the drift induction procedure, it is first shuffled. The drift induction technique suggested by [Žli10a] creates feature drift within the static dataset at the point known as the Change of Point in the stream. Drift is created by selecting a subset of features at random and rotating their values for a specific class. As per this method, if the feature sequence(2,4,6) is chosen for class label 0, after the ChangePoint, the rearranged feature sequence would be (4,6,2). This simple methodology guarantees significant feature drifts can be formed, whereas the dataset's initial data attributes are maintained. This method is, however, reliant on the features chosen for rotation, and if the 'correct' set of features is not chosen, it produces inconsistent results. The core idea of [Žli10a] is extended in this drift induction approach, which provides for more flexibility over the form of change. Rather than selecting a set of characteristics at random, it is preferable to choose features based on their significance to the classification assignment. It would be implemented by ranking individual features as per its information gain metric [DWBB04]. Afterward, it would select top-ranked features for inducing real drift and selecting the lowest rank feature depending on the desired change.

6.3.3 Detectability & false alarm

Here, two experiments are carried out: a) Detectability studies, in which the top 25% of features from the ranked list are chosen, and b) False alarm tests, in which the lowest 25% are chosen. This was done to evaluate the detection performance and the robustness against irrelevant modifications. Because the top 25% of ranking features are ranked based on their information content, and changing these features causes performance to degrade, which must be noticed and rectified, they significantly impact the classification process. Modifying the bottom 25% of the features has a minor impact on the classification process. It leads to false alarms, which the detectors should ignore—the effect of shifting the top 25% and lowest 25% of the population. The ChangePoint was triggered at 50% of the stream in all datasets. A model is trained on data before the ChangePoint and then evaluated on samples afterward. The model's accuracy on the training and original test sets (before induction) is similar, indicating a static dataset. For detection of ChangePoint studies, the top 25% of the features are rotated, resulting in a considerable decline in test accuracy. (Table 6.2). Real drifts, which must be discovered by a drift detection method, are indicated by this. In the False alarm trials, rotating the bottom 25% of the features did not result in a substantial reduction in test accuracy. Even though the same number of characteristics is turned in both circumstances, the importance of the features varies when it comes to the classification task. To examine the behaviour of the algorithms under different change scenarios, this study has conducted experimental analysis on the top 25% and bottom 25% datasets.

6.4 Experiment 1

The detectability and False alarm test(discussed in 6.3.2) can be used to understand the performance of different drift detectors in terms of real drift detection and false alarm. Following aspects are examined with these tests:
1. No retraining (keep the static model) concept

2.ADWIN (supervised drift detector, discussed in 3.1) drift detector

3. Unsupervised drift detector D3 (discussed in 3.3)

4. AEDD (proposed unsupervised drift detector) drift detector

A false alarm can be caused by drift detector because of the change in the feature space, but such differences might not have a meaningful effect on predictive analysis. If the detector can avoid false alarms, then valuable resources like cost and time for model retraining can be saved. With the above-mentioned drift induction technique, the real drift is induced in the data stream once. The model performance is expected to degrade significantly in case of the detectability test, but for false alarm, performance has a slight change overall (refer to Table 6.2). The results and the finding will be discussed in the following sub-section.

6.4.1 Results and discussion on experiment 1

The experiment (refer to Table 6.3) shows a vast accuracy drop if the model doesn't incorporate any drift detection and retraining mechanism. Since the no retraining method has the lowest accuracy, the need for a drift detection mechanism is evident. Next, the standard supervised drift detector ADWIN is applied to the dataset and found drift exactly once. This proves its resilience toward false alarms. The approaches D3 and AEDD successfully discover 3 and 2 drifts respectively, in Detectability tests and achieve overall good accuracy. D3 and AEDD are unsupervised approaches, but their overall accuracy was similar to supervised drift detector methods. This demonstrates that AEDD techniques can be utilized rather than supervised approaches without reducing predictive

Datasets	Algorithms	Induced drift in top 25% of feature			Induced Drift in bottom 25% of feature
		Accuracy	Drifts	False Alarms	detected drifts
Bank	No retraining	67.45	0	0	0
	ADWIN	88.86	1	0	0
	D3	91.50	3	2	1
	AEDD	92.87	2	1	0
Phishing	No retraining	87.29	0	0	0
	ADWIN	93.57	1	0	0
	D3	92.87	2	1	2
	AEDD	93.59	1	1	0
Wine	No retraining	81.79	0	0	0
	ADWIN	97.96	1	0	0
	D3	99.01	0	1	1
	AEDD	99.57	0	1	0

Table 6.3: Detection results of different methodologies for induced drift dataset

accuracy considerably. But it can necessitate unnecessarily model retraining as these methods are not resilient to false alarm. ADWIN and AEDD methods are both immune to a false alarm but the D3 approach flags them as a significant shift that should be investigated further.

6.5 Experiment 2

Due to some fundamental constraints such as true label unavailability, drift detection is a complicated process [Bif17]. When creating a drift detection system, balancing false and genuine alarms is essential while decreasing the time between transition and detection must also be focused. A drift detector's design compromises the ability to identify actual changes while avoiding false alarms. This information is formally apprehended in the specified benchmarks [GG00] to estimate the performance of change detection systems. The four performance metrics related to the drift detector are listed below:

- Mean Time Between False Alarms (MTFA): How frequently a false alarm occurs while no change occurs (the higher, the better). The distance between consecutive (false) alarms before the change point is used to calculate MTFA. In addition, this score is averaged over the 5 repetitions.
- Mean Time to Detection (MTD): How long does it take on average for a method to identify a modification after this occurs (the lower, the better)? The number of points between the change point and the following alert triggered by the method, is counted. MTD's score is averaged across 5 repetitions, similar to MTFA's.
- Missed Detection Ratio (MDR): This metric represents likelihood of missing a drift. The fraction of repetitions (across the 5 simulations) in which the drift method fails to raise an alarm following the start of the drift is used to calculate this. This value should ideally be zero, indicating that all drifts are collected, regardless of duration.

Method	MTFA	MTD	MDR	ND
D3	2238	562	0.0	9
STUDD	5400	1618	0.0	2
UDD	13276	2374	0.0	3
AEDD	12765	1031	0.2	5

Table 6.4: Performance metric for AEDD and baseline models for bank dataset

Method	MTFA	MTD	MDR	ND
D3	811	154	0.14	12
STUDD	4765	1654	0.08	6
UDD	8778	1254	0.03	5
AEDD	7686	910	0.18	9

Table 6.5: Performance metric for AEDD and baseline models for Phishing datasets

- Number of Detections (ND) A total number of detected drift in the model has been taken into account.

6.5.1 Results and discussion on experiment 2

To find out how good our proposed drift detection algorithm is in terms of performance and efficiency, we used the idea of performance metric (discussed in Experiment 2). The state of art methods for this study considered D3[CGBT21], STUDD[CGBT21], UDD[BSSK21] are discussed in related work. The results are shown in Table 6.4, Table 6.5 and Table 6.6 for their respective datasets. We induce drift in the bank, phishing and wine dataset (discussed in the experiment section) and calculate the performance metrics. This experiment has been done 5 times to calculate the average of these metrics.

Based on the results obtained for the Bank, phishing and the wine dataset, we can say that proposed method AEDD have a MDR value of 20%, 18% and 5% repectively that means it has low probability of failing to identify a drift. Therefore, AEDD is able to obtain artificial drift that has been induced in externally. But in terms of MDR values (ideal value is 0) terms of other state-of-art methods, STUDD and UDD has done better job than AEDD.

Method	MTFA	MTD	MDR	ND
D3	549	62	0.30	6
STUDD	657	543	0.23	4
UDD	983	376	0.03	4
AEDD	884	176	0.05	3

Table 6.6: Performance metric for AEDD and baseline models for Wine datasets

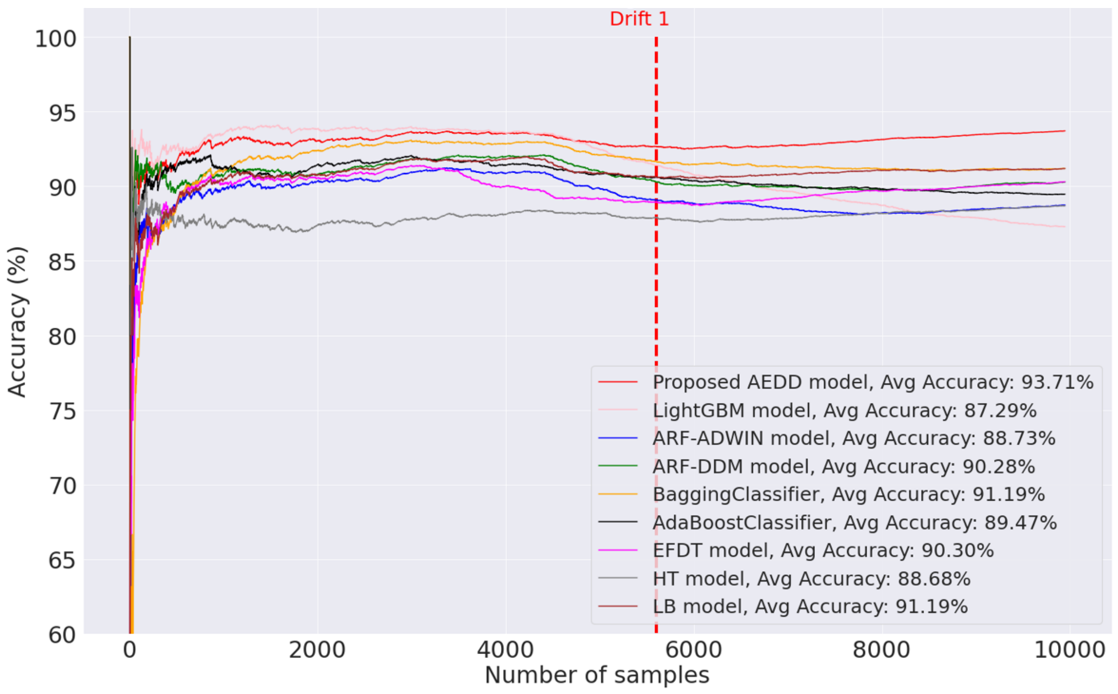


Figure 6.1: Accuracy comparison among proposed AEDD and other existing state of art algorithms: for Phishing dataset

In terms of MTD, the value for AEDD is relatively lower than the UDD and STUDD which means after the drift was induced in the dataset it has taken relatively lower time than other methods to identify drift.

Concerning the MTFA values, AEDD has has relatively higher value than D3 and STUDD, which indicates there is relatively long distance between two false alarms, therefore it is more robust to false alarms.

The number of detected drift by AEDD is relatively fewer than D3. So, regarding our research question 2, how we can compare different drift detector's performance, we can conclude that the proposed approach with this performance metrics the unsupervised drift detector algorithm's performance can be soundly compared. The comparison study clearly demonstrates that the AEDD method perform better than D3 but is not superior to UDD. In few cases has given even better performance that STUDD.

6.6 Experiment 3

The main goal is to approach the RQ3 by comparing the accuracy, precision, recall and average test time values among proposed AEDD and other existing state of art algorithms (detailed in the literature review), including ARF [GBR+17], HT [HSD01], EFDT [DH00], LB [BHP10], Bagging Classifier [OR01a], Adaboost [YQJL13], LightGBM [KMF+17].

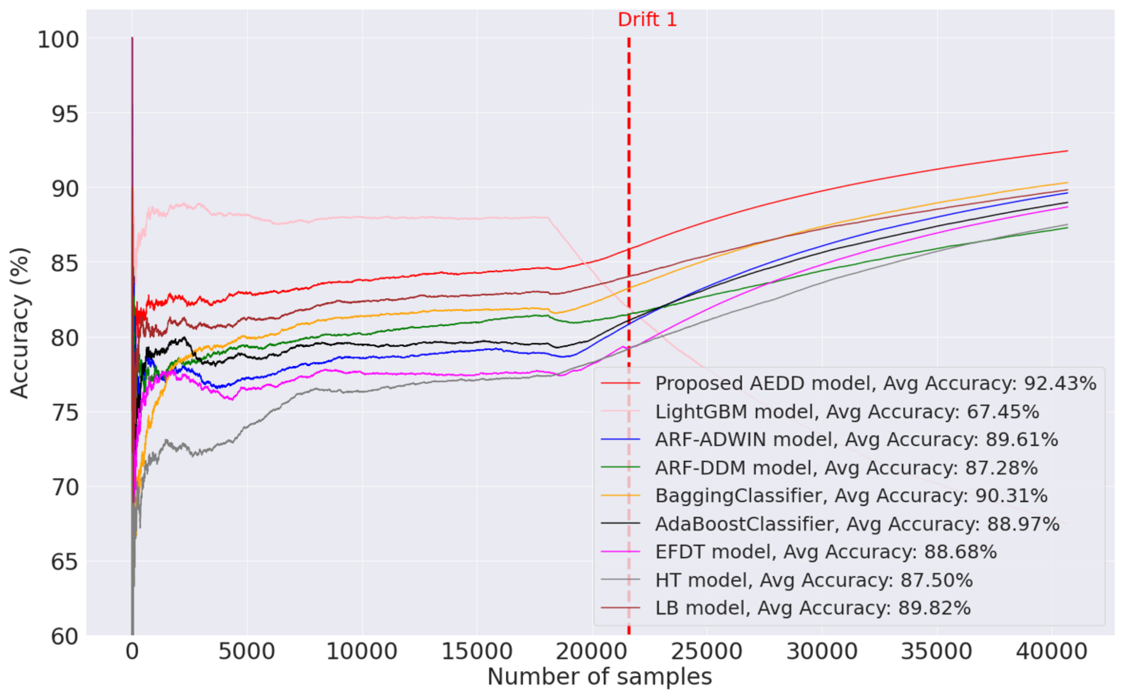


Figure 6.2: Accuracy comparison among proposed AEDD and other existing state of art algorithms: for Bank dataset

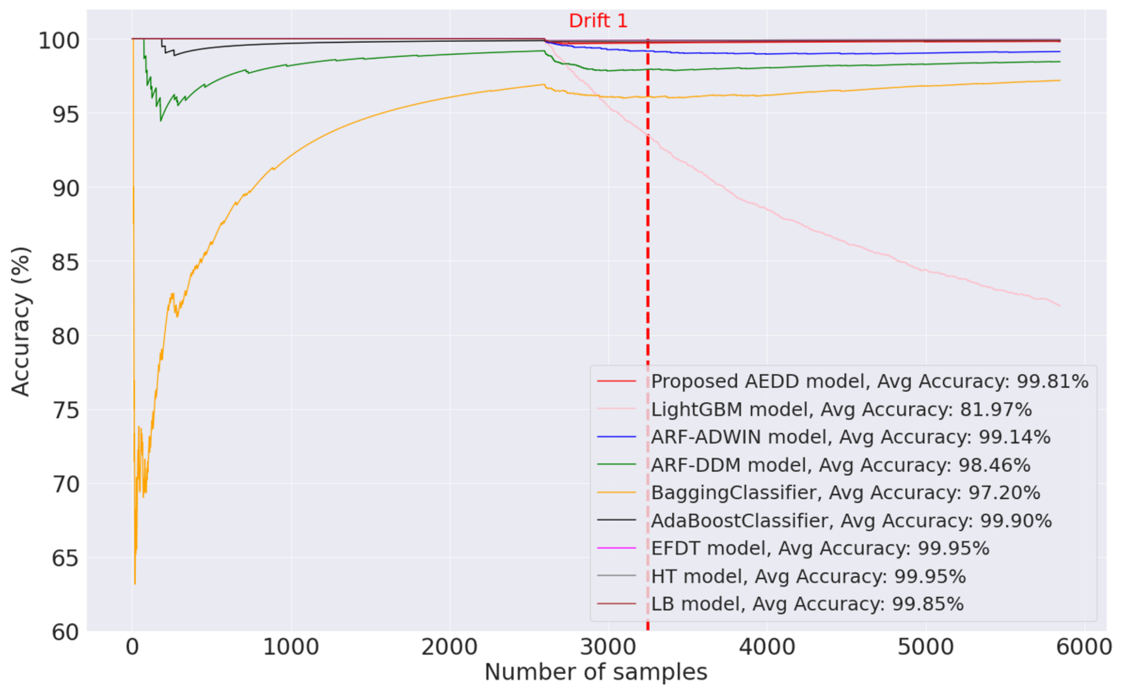


Figure 6.3: Accuracy comparison among proposed AEDD and other existing state of art algorithms: for Wine dataset

Method	Accuracy(%)	Precision(%)	Recall(%)	Avg. Test time(ms)
Proposed AEDD	92.87	91.99	92.98	15.6
LightGBM	67.45	76.21	50.79	0.13
ARF ADWIN	88.86	88.77	88.96	1.4
ARF DDM	87.74	85.56	90.56	1.55
EFDT	88.68	90.62	86.97	5.4
HT	87.50	89.91	85.91	0.3
LB	89.70	88.65	91.76	7.4
AdaBoostClassifier	88.97	87.97	90.67	6.5
BaggingClassifier	90.31	90.64	90.94	3.1

Table 6.7: Plot accuracy for adaptive algorithms on Bank Dataset

Method	Accuracy(%)	Precision(%)	Recall(%)	Avg. Test time(ms)
Proposed AEDD	93.57	85.55	81.85	5.5
LightGBM	87.29	84.34	94.67	0.3
ARF ADWIN	88.85	87.85	92.85	0.9
ARF DDM	89.59	89.76	94.76	1.2
EFDT	90.30	88.98	94.98	2.4
HT	88.68	90.23	88.23	0.7
LB	90.97	90.45	93.45	1.4
AdaBoostClassifier	89.47	90.67	90.67	1.5
BaggingClassifier	91.19	90.16	94.16	3.1

Table 6.8: Plot accuracy for adaptive algorithms on Phishing Dataset

Method	Accuracy(%)	Precision(%)	Recall(%)	Avg. Test time(ms)
Proposed AEDD	99.57	99.57	99.53	15.6
LightGBM	81.97	81.97	81.56	0.3
ARF ADWIN	97.96	98.96	98.56	0.9
ARF DDM	99.57	99.32	99.76	1.2
EFDT	99.95	99.75	99.43	0.4
HT	99.56	99.23	99.64	0.5
LB	99.85	99.85	99.65	5.4
AdaBoostClassifier	99.90	99.90	99.26	1.5
BaggingClassifier	97.20	97.20	97.20	3.1

Table 6.9: Plot accuracy for adaptive algorithms on Wine Dataset

6.6.1 Results and discussion on experiment 3

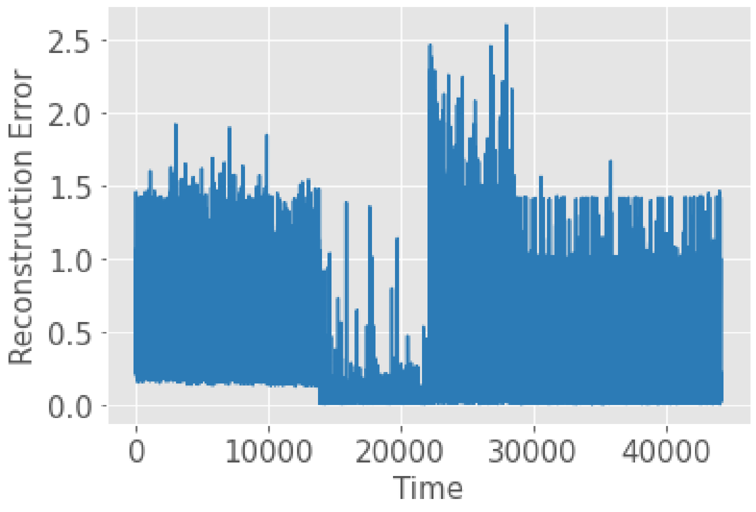
Figure 6.1, Figure 6.2 and Figure 6.3 demonstrates the accuracy, precision, recall and avg. test time comparison among proposed AEDD and other existing state of art algorithms. Table 6.7 lists the comparison on key parameters of all studied algorithms. The study shows that the proposed AEDD has outperformed the other algorithms in precision, accuracy, and recall value for all dataset.

6.7 Visual explanation of AEDD

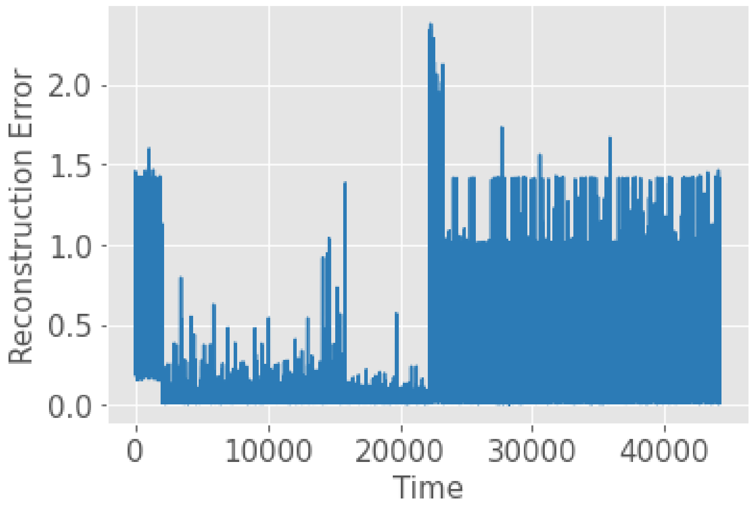
Finally, the visual representation of the working mechanism of autoencoder based drift detector is described.

6.7.1 Reconstruction error change

The working principle of the autoencoder based drift detector is elucidated in Section 5. Here, we will demonstrate visually how does the reconstruction error vary with time and what is the impact of using an autoencoder based drift detector in terms of reconstruction error change. The Figure 6.4a shows the original reconstruction error of the real world dataset when no drift detection mechanism is taken account and the Figure 6.4b demonstrate the refreshed reconstruction error calculated by the autoencoder after drift is detected. If we look into the Figure 6.4b carefully, we can see that reconstruction error changes with time. As per the explanation in 5, if the pattern of the data streams varies over time, the input and reconstructed output will be significantly different. Figure 6.4a shows reconstruction error is substantially high for a change in distribution. If reconstruction error changes and it continuously high for prolong period then AEDD identifies that as drift and initiate an autoencoder retraining. Therefore, autoencoder learns the new distribution and the autoencoder has a new refreshed reconstruction error over time shown in Figure 6.4b. Figure 6.4 and Figure 6.5 represents Bank and Phishing dataset respectively.



(a)

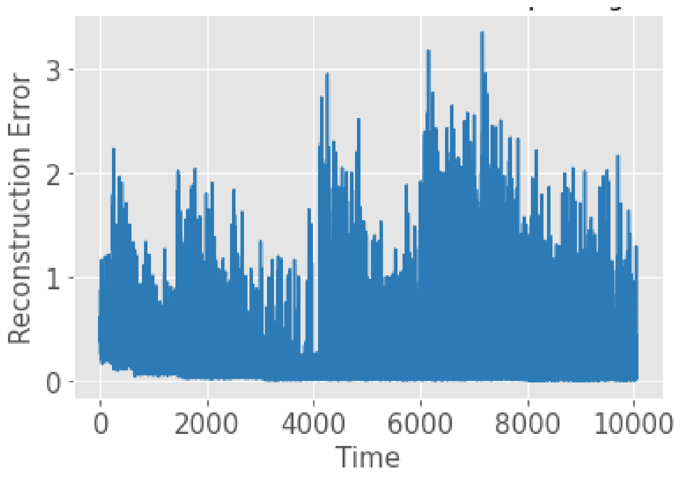


(b)

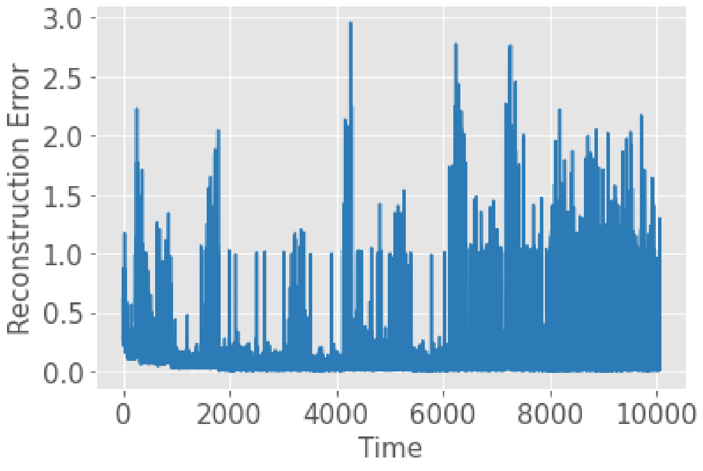
Figure 6.4: a) Reconstruction and b) refreshed reconstruction error with respect to time for Bank Dataset

6.7.2 Change of width for ADWIN

We have chosen ADWIN to monitor the relative change in the reconstruction error. If ADWIN finds that the incoming dataset belongs to the same distribution,its window grows. However, once ADWIN detects a change in the data distribution, the window shrinks and starts gathering the data from the new distribution. The evolution of ADWIN's window width over time is shown in Figure 6.6. Figure 6.6a and Figure 6.6b shows the ADWIN width change for the Phishing and Bank dataset respectively.



(a)



(b)

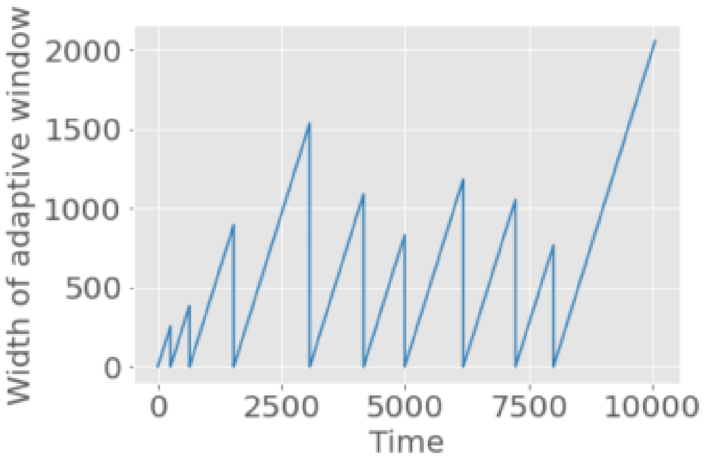
Figure 6.5: Plot of Reconstruction error and refreshed reconstruction error over Time for Phishing dataset

6.7.3 Autoencoder training

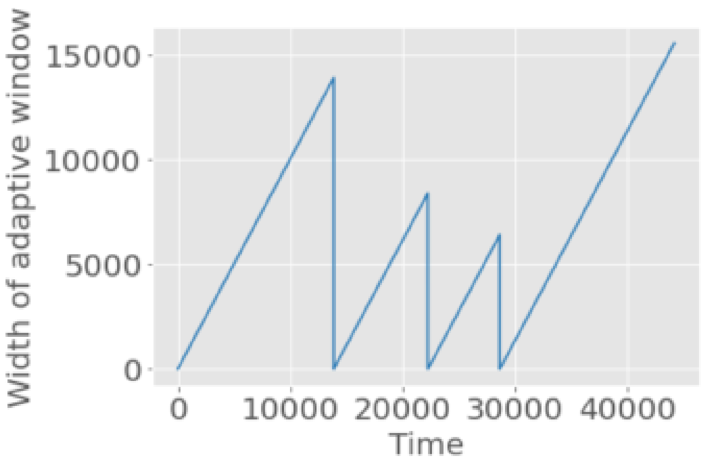
Autoencoder is trained with training data initially. Since then, whenever ADWIN detect drift the autoencoder has to retrain with the new incoming data. The model was trained over 100 epochs. Figure 6.7a demonstrate the loss of autoencoder over epochs and 6.7b shows the loss of auto encoder after each gradient decent over epochs respectively.

6.7.4 Detected drift points

In the final stage, drift is signaled once ADWIN identifies the significant change in the reconstruction error with its adaptive windowing technique. Autoencoder must retrain again with the new incoming dataset. Figure 6.8a shows the detected drift over time. We also wanted to dive deep into the

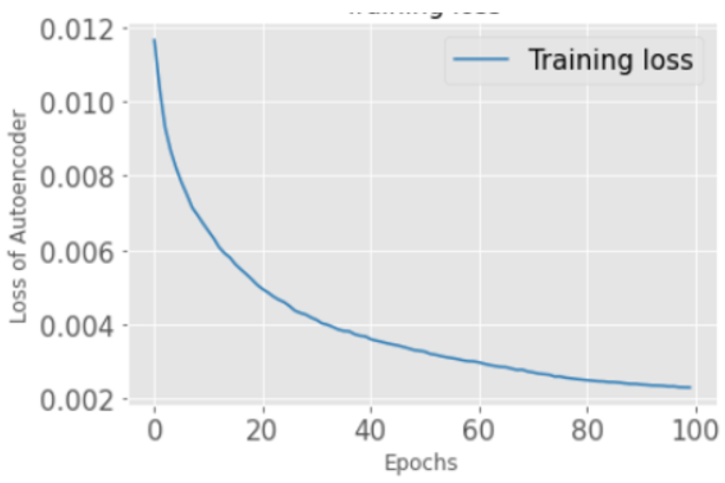


(a)

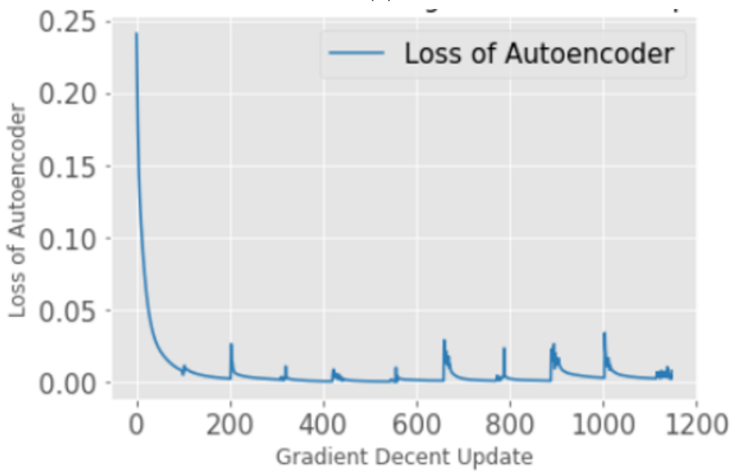


(b)

Figure 6.6: The change of ADWIN window width with respect to the time for a) Phishing b) Bank dataset



(a)



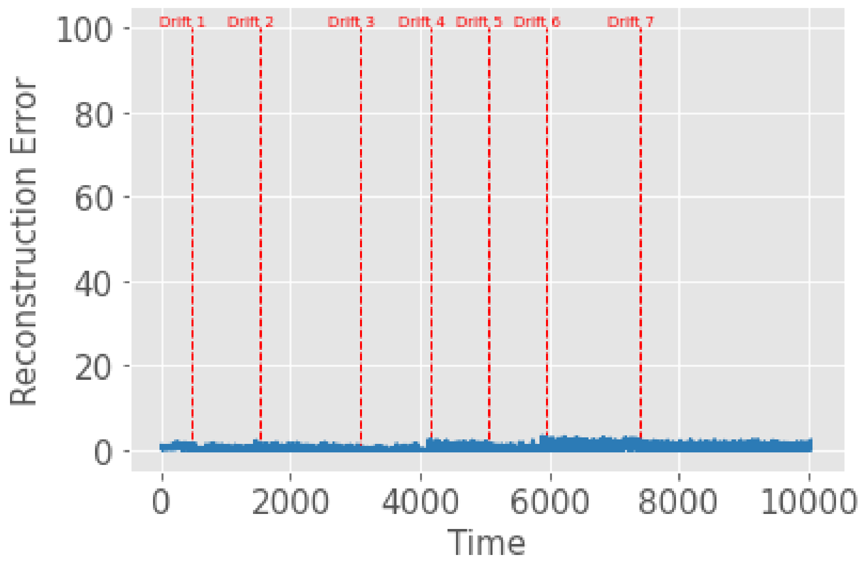
(b)

Figure 6.7: a) loss of autoencoder for training loss and b) loss of auto encoder after each gradient decent over epochs for Phishing dataset

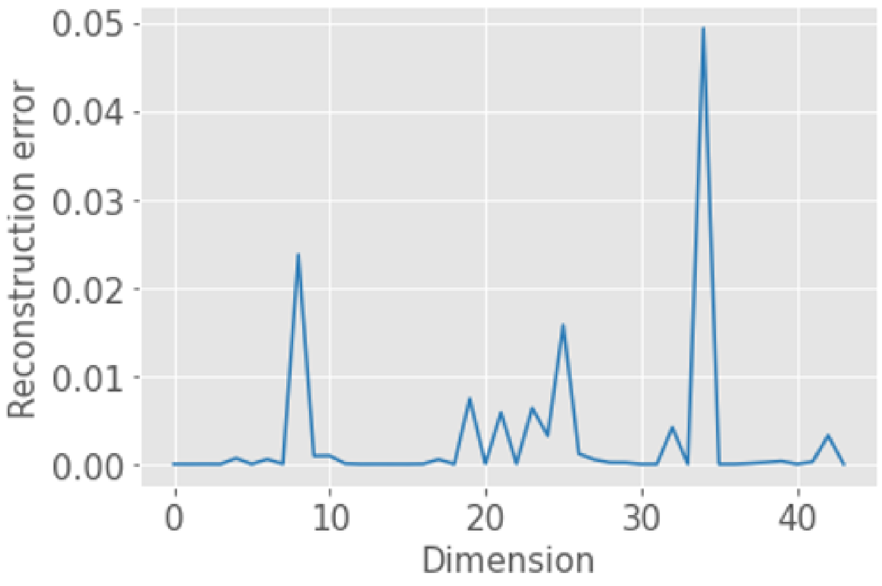
reconstruction error change over individual feature dimensions. Figure 6.8b demonstrates the features 8, 27, and 34 that have the highest reconstruction error on the detected point from the experiment.

6.7.5 Summary of experimental results

Due to drift detection, maintaining the precision of a high-performing machine learning model over time is difficult. Numerous attempts and techniques have identified drift in streaming data sets. Our findings show that AEDD can compete with other cutting-edge unsupervised drift detection



(a)



(b)

Figure 6.8: Plots of a) detected drift over time and b) reconstruction error of specific point for phishing dataset

techniques. Finally, we demonstrate that the AEDD tend to give better test results, more reliable drift threshold parameter adjustments and more extensive and more complicated auto encoder architectures.

6.7.6 Limitations

The suggested method does not require the class labels during detection and gives the domain expert the flexibility to set the label. However, when there is a feature change does not occur, its performance can be degraded since we cannot locate a feature that significantly correlates with the true label. The results of the dataset experiments are validated on the assumption that there is a significant change in the feature's space. However, given a real-world dataset, this assumption may not be always valid and more research is needed to look at the intricacies of this methodology.

7 Conclusion and Future work

This thesis proposes a novel unsupervised drift detection method. The state-of-the-art unsupervised drift detection mechanisms are often ill equipped to deal with drift detection and often provides poor results due to lack of label availability, false alarms and lack of adaptability. Because these methods often require access to the whole collection of true labels, standard drift detection methods like DDM and ADWIN are inapplicable of providing reliable results in such situations.

To deal with these problems, the AEDD algorithm for concept drift detection is proposed in this thesis. This approach does not rely on true labels to detect concept drift, and only requires access to a small selection of true labels to retrain the prediction model if drift is discovered. As a result, this approach is particularly well suited to drift handling in deployed ML settings in real-world scenarios where correct label acquisition is costly. The problem solving approach is based on reconstruction error estimation obtained by an autoencoder.

To validate the performance of the proposed AEDD method, a series of experiments are carried out. In Experiment 1, we have explicitly induced drift to identify if the detector can find drift in proper positions or if it gives false alarms. We see that the performance of AEDD is comparable with other supervised drift detectors. It manages to ignore false alarms in most of the cases, outperforming other unsupervised drift detection algorithms. The performance was tested on different test scenarios to compare the output of AEDD in conjunction with other unsupervised drift detectors and obtained similar results. In experiment 2, we tried to determine how accurately different drift detectors can identify drift and the time required to react against drift. The proposed methodology can compete with other existing drift detectors to identify drift more accurately and faster. Finally, in Experiment 3, we tried to understand how well the proposed adaptive algorithm adapts with new incoming data distribution so that even if drift is present, the model performance won't be affected. The AEDD-based drift adaptation strategies outperformed other existing drift adaptation strategies in terms of accuracy, precision, and recall of all three datasets.

The AEDD approach can be adopted in many real-life situations, for example, in the production line where quality control of products is determined using machine learning using sensor and observation data. This method can be incorporated to the existing ML algorithms used in the monitoring system without significant cost and computational resources. The future work includes extending the suggested method to make it more resource-aware for various fast-changing data streams containing multiple types of drift and capable of dealing with missing values and unbalanced datasets. Although the study has been conducted in three known data sets, a more comprehensive analysis must be carried out on several unbiased datasets to test the concepts and its applicability.

Bibliography

[AC15] J. An, S. Cho. "Variational autoencoder based anomaly detection using reconstruction probability". In: *Special Lecture on IE* 2.1 (2015), pp. 1–18 (cit. on p. 49).

[Alt07] H. Altınçay. "Ensembling evidential k-nearest neighbor classifiers through multi-modal perturbation". In: *Applied Soft Computing* 7.3 (2007), pp. 1072–1083 (cit. on p. 29).

[APHW03] C. C. Aggarwal, S. Y. Philip, J. Han, J. Wang. "A framework for clustering evolving data streams". In: *Proceedings 2003 VLDB conference*. Elsevier. 2003, pp. 81–92 (cit. on p. 15).

[BCF+06] M. Baena-Garcıa, J. del Campo- Avila, R. Fidalgo, A. Bifet, R. Gavalda, R. Morales-Bueno. "Early drift detection method". In: *Fourth international workshop on knowledge discovery from data streams*. Vol. 6. 2006, pp. 77–86 (cit. on p. 33).

[BG07] A. Bifet, R. Gavalda. "Learning from time-changing data with adaptive windowing". In: *Proceedings of the 2007 SIAM international conference on data mining*. SIAM. 2007, pp. 443–448 (cit. on pp. 16, 17, 32, 35, 49).

[BHP10] A. Bifet, G. Holmes, B. Pfahringer. "Leveraging bagging for evolving data streams". In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2010, pp. 135–150 (cit. on pp. 37, 59).

[Bif17] A. Bifet. "Classifier concept drift detection and the illusion of progress". In: *International Conference on Artificial Intelligence and Soft Computing*. Springer. 2017, pp. 715–725 (cit. on p. 57).

[BM08] S. H. Bach, M. A. Maloof. "Paired learners for concept drift". In: *2008 Eighth IEEE International Conference on Data Mining*. IEEE. 2008, pp. 23–32 (cit. on p. 35).

[BS13] D. Brzezinski, J. Stefanowski. "Reacting to different types of concept drift: The accuracy updated ensemble algorithm". In: *IEEE Transactions on Neural Networks and Learning Systems* 25.1 (2013), pp. 81–94 (cit. on p. 38).

[BSSK21] L. Baier, T. Schlör, J. Schöffer, N. Kühl. "Detecting concept drift with neural network model uncertainty". In: *arXiv preprint arXiv:2107.01873* (2021) (cit. on pp. 39, 58).

[CGB20] V. Cerqueira, H. M. Gomes, A. Bifet. "Unsupervised concept drift detection using a student–teacher approach". In: *International Conference on Discovery Science*. Springer. 2020, pp. 190–204 (cit. on p. 17).

[CGBT21] V. Cerqueira, H. M. Gomes, A. Bifet, L. Torgo. "STUDD: A Student-Teacher Method for Unsupervised Concept Drift Detection". In: *arXiv preprint arXiv:2103.00903* (2021) (cit. on p. 58).

[CZ04] F. Chu, C. Zaniolo. "Fast and light boosting for adaptive mining of data streams". In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer. 2004, pp. 282–292 (cit. on p. 37).

[DH00] P. Domingos, G. Hulten. “Mining high-speed data streams”. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2000, pp. 71–80 (cit. on pp. 35, 59).

[DP11] G. Ditzler, R. Polikar. “Hellinger distance based drift detection for nonstationary environments”. In: *2011 IEEE symposium on computational intelligence in dynamic and uncertain environments (CIDUE)*. IEEE. 2011, pp. 41–48 (cit. on p. 33).

[DP12] G. Ditzler, R. Polikar. “Incremental learning of concept drift from streaming imbalanced data”. In: *IEEE transactions on knowledge and data engineering* 25.10 (2012), pp. 2283–2301 (cit. on p. 29).

[DWBB04] W. Duch, T. Wieczorek, J. Biesiada, M. Blachnik. “Comparison of feature ranking methods based on information entropy”. In: *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*. Vol. 2. IEEE. 2004, pp. 1415–1419 (cit. on p. 55).

[EP11] R. Elwell, R. Polikar. “Incremental learning of concept drift in nonstationary environments”. In: *IEEE Transactions on Neural Networks* 22.10 (2011), pp. 1517–1531 (cit. on p. 38).

[FB13] W. Fan, A. Bifet. “Mining big data: current status, and forecast to the future”. In: *ACM SIGKDD explorations newsletter* 14.2 (2013), pp. 1–5 (cit. on p. 16).

[GBBC19] Ö. Gözüaçık, A. Büyükçakır, H. Bonab, F. Can. “Unsupervised concept drift detection with a discriminative classifier”. In: *Proceedings of the 28th ACM international conference on information and knowledge management*. 2019, pp. 2365–2368 (cit. on pp. 16, 33, 39).

[GBR+17] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, T. Abdessalem. “Adaptive random forests for evolving data stream classification”. In: *Machine Learning* 106.9 (2017), pp. 1469–1495 (cit. on pp. 37, 59).

[GCGD20] R. N. Gemaque, A. F. J. Costa, R. Giusti, E. M. Dos Santos. “An overview of unsupervised drift detection methods”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 10.6 (2020), e1381 (cit. on pp. 19, 34).

[GG00] F. Gustafsson, F. Gustafsson. *Adaptive filtering and change detection*. Vol. 1. Citeseer, 2000 (cit. on p. 57).

[GMCR04] J. Gama, P. Medas, G. Castillo, P. Rodrigues. “Learning with drift detection”. In: *Brazilian symposium on artificial intelligence*. Springer. 2004, pp. 286–295 (cit. on pp. 16, 27, 33).

[GRB+19] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, J. Gama. “Machine learning for streaming data: state of the art, challenges, and opportunities”. In: *ACM SIGKDD Explorations Newsletter* 21.2 (2019), pp. 6–22 (cit. on p. 16).

[GŽB+14] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia. “A survey on concept drift adaptation”. In: *ACM computing surveys (CSUR)* 46.4 (2014), pp. 1–37 (cit. on p. 31).

[HGL15] D. Han, C. Giraud-Carrier, S. Li. “Efficient mining of high-speed uncertain data streams”. In: *Applied Intelligence* 43.4 (2015), pp. 773–785 (cit. on p. 35).

[HSD01] G. Hulten, L. Spencer, P. Domingos. “Mining time-changing data streams”. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001, pp. 97–106 (cit. on pp. 35, 50, 59).

[HZ93] G. E. Hinton, R. Zemel. “Autoencoders, minimum description length and Helmholtz free energy”. In: *Advances in neural information processing systems* 6 (1993) (cit. on p. 17).

[JRA20] M. Jaworski, L. Rutkowski, P. Angelov. “Concept drift detection using autoencoders in data streams processing”. In: *International Conference on Artificial Intelligence and Soft Computing*. Springer. 2020, pp. 124–133 (cit. on pp. 47, 48).

[KM07] J. Z. Kolter, M. A. Maloof. “Dynamic weighted majority: An ensemble method for drifting concepts”. In: *The Journal of Machine Learning Research* 8 (2007), pp. 2755–2790 (cit. on p. 37).

[KMF+17] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu. “Lightgbm: A highly efficient gradient boosting decision tree”. In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 59).

[KRW10] M. Kantardzic, J. W. Ryu, C. Walgampaya. “Building a new classifier in an ensemble using streaming unlabeled data”. In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2010, pp. 77–86 (cit. on p. 34).

[Kun04] L. I. Kuncheva. “Classifier ensembles for changing environments”. In: *International Workshop on Multiple Classifier Systems*. Springer. 2004, pp. 1–15 (cit. on p. 29).

[KŽB+14] G. Krempl, I. Žliobaite, D. Brzezi nski, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou, et al. “Open challenges for data stream mining research”. In: *ACM SIGKDD explorations newsletter* 16.1 (2014), pp. 1–10 (cit. on p. 16).

[Lic+13] M. Lichman et al. *UCI machine learning repository, 2013*. 2013 (cit. on p. 54).

[LSZL17] A. Liu, Y. Song, G. Zhang, J. Lu. “Regional concept drift detection and density synchronized drift adaptation”. In: *IJCAI International Joint Conference on Artificial Intelligence*. 2017 (cit. on p. 20).

[LWHW15] P. Li, X. Wu, X. Hu, H. Wang. “Learning concept-drifting data streams with random ensemble decision trees”. In: *Neurocomputing* 166 (2015), pp. 68–83 (cit. on p. 37).

[LWJ16] D. Liu, Y. Wu, H. Jiang. “FP-ELM: An online sequential learning algorithm for dealing with concept drift”. In: *Neurocomputing* 207 (2016), pp. 322–334 (cit. on p. 35).

[LZL14] N. Lu, G. Zhang, J. Lu. “Concept drift detection via competence models”. In: *Artificial Intelligence* 209 (2014), pp. 11–28 (cit. on p. 20).

[MRA+12] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodr ıguez, N. V. Chawla, F. Herrera. “A unifying view on dataset shift in classification”. In: *Pattern recognition* 45.1 (2012), pp. 521–530 (cit. on p. 21).

[MRBA18] J. Montiel, J. Read, A. Bifet, T. Abdessalem. “Scikit-multiflow: A multi-output streaming framework”. In: *The Journal of Machine Learning Research* 19.1 (2018), pp. 2915–2914 (cit. on p. 54).

[MY11] L. L. Minku, X. Yao. “DDD: A new ensemble approach for dealing with concept drift”. In: *IEEE transactions on knowledge and data engineering* 24.4 (2011), pp. 619–633 (cit. on p. 29).

[OR01a] N. C. Oza, S. Russell. “Experimental comparisons of online and batch versions of bagging and boosting”. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001, pp. 359–364 (cit. on pp. 37, 59).

[OR01b] N. C. Oza, S. J. Russell. “Online bagging and boosting”. In: *International Workshop on Artificial Intelligence and Statistics*. PMLR. 2001, pp. 229–236 (cit. on p. 37).

[Pag54] E. S. Page. “Continuous inspection schemes”. In: *Biometrika* 41.1/2 (1954), pp. 100–115 (cit. on pp. 27, 32).

[PV16] A. Pesaranghader, H. L. Viktor. “Fast hoeffding drift detection method for evolving data streams”. In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2016, pp. 96–111 (cit. on p. 32).

[PVP18] A. Pesaranghader, H. Viktor, E. Paquet. “Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams”. In: *Machine Learning* 107.11 (2018), pp. 1711–1743 (cit. on p. 32).

[RCNW18] A. G. Roy, S. Conjeti, N. Navab, C. Wachinger. “Inherent brain segmentation quality control from fully convnet monte carlo sampling”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2018, pp. 664–672 (cit. on p. 39).

[RFMB16] D. M. dos Reis, P. Flach, S. Matwin, G. 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*. 2016, pp. 1545–1554 (cit. on p. 33).

[SA16] S. G. Soares, R. Ara ujo. “An adaptive ensemble of on-line extreme learning machines with variable forgetting factor for dynamic system prediction”. In: *Neurocomputing* 171 (2016), pp. 693–707 (cit. on p. 35).

[SG86] J. C. Schlimmer, R. H. Granger. “Incremental learning from noisy data”. In: *Machine learning* 1.3 (1986), pp. 317–354 (cit. on p. 20).

[Sha+12] S. Shalev-Shwartz et al. “Online learning and online convex optimization”. In: *Foundations and Trends® in Machine Learning* 4.2 (2012), pp. 107–194 (cit. on p. 16).

[SK17] T. S. Sethi, M. Kantardzic. “On the reliable detection of concept drift from streaming unlabeled data”. In: *Expert Systems with Applications* 82 (2017), pp. 77–99 (cit. on pp. 34, 54).

[STM+16] Y. Sun, K. Tang, L. L. Minku, S. Wang, X. Yao. “Online ensemble learning of data streams with gradually evolved classes”. In: *IEEE Transactions on Knowledge and Data Engineering* 28.6 (2016), pp. 1532–1545 (cit. on p. 37).

[Sto09] A. Storkey. “When training and test sets are different: characterizing learning transfer”. In: *Dataset shift in machine learning* 30 (2009), pp. 3–28 (cit. on p. 21).

[Wal73] A. Wald. *Sequential analysis: Courier Corporation*. 1973 (cit. on p. 32).

[WK96] G. Widmer, M. Kubat. “Learning in the presence of concept drift and hidden contexts”. In: *Machine learning* 23.1 (1996), pp. 69–101 (cit. on p. 21).

[WLH12] X. Wu, P. Li, X. Hu. “Learning from concept drifting data streams with unlabeled data”. In: *Neurocomputing* 92 (2012), pp. 145–155 (cit. on p. 34).

[WYZ16] Y. Wang, H. Yao, S. Zhao. “Auto-encoder based dimensionality reduction”. In: *Neurocomputing* 184 (2016), pp. 232–242 (cit. on p. 47).

[XW17] S. Xu, J. Wang. “Dynamic extreme learning machine for data stream classification”. In: *Neurocomputing* 238 (2017), pp. 433–449 (cit. on p. 35).

[XXYA17] Y. Xu, R. Xu, W. Yan, P. Ardis. “Concept drift learning with alternating learners”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 2104–2111 (cit. on p. 38).

[YF12] H. Yang, S. Fong. “Incrementally optimized decision tree for noisy big data”. In: *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*. 2012, pp. 36–44 (cit. on p. 36).

[YHH15] X.-C. Yin, K. Huang, H.-W. Hao. “DE2: Dynamic ensemble of ensembles for learning nonstationary data”. In: *Neurocomputing* 165 (2015), pp. 14–22 (cit. on p. 38).

[YQJL13] C. Ying, M. Qi-Guang, L. Jia-Chen, G. Lin. “Advance and prospects of AdaBoost algorithm”. In: *Acta Automatica Sinica* 39.6 (2013), pp. 745–758 (cit. on p. 59).

[YWP18] S. Yu, X. Wang, J. C. Pr ıncipe. “Request-and-reverify: Hierarchical hypothesis testing for concept drift detection with expensive labels”. In: *arXiv preprint arXiv:1806.10131* (2018) (cit. on p. 34).

[Žli10a] I. Žliobaite. “Change with delayed labeling: When is it detectable?” In: *2010 IEEE International Conference on Data Mining Workshops*. IEEE. 2010, pp. 843–850 (cit. on pp. 16, 55).

[Žli10b] I. Žliobaitė. “Learning under concept drift: an overview”. In: *arXiv preprint arXiv:1010.4784* (2010) (cit. on pp. 23–26).

[ZP17] C. Zhou, R. C. Paffenroth. “Anomaly detection with robust deep autoencoders”. In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 665–674 (cit. on pp. 30, 47).

[ZZS08] P. Zhang, X. Zhu, Y. Shi. “Categorizing and mining concept drifting data streams”. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 812–820 (cit. on p. 29).

Declaration