Open Peer Review on Qeios

A Novel Framework for Concept Drift Detection using Autoencoders for Classification Problems in Data Streams

Usman Ali[1], Tariq Mahmood[1]

1 Institute of Business Administration Karachi

Funding: No specific funding was received for this work.

Potential competing interests: No potential competing interests to declare.

Abstract

In streaming data environments like weather forecasting, health care monitoring, network traffic monitoring and energy consumption etc., data characteristics and probability distributions are likely to change over time, posing challenges for classification models to predict accurately. In such non-stationary environments where the patterns in data can change quickly, the pre-trained machine learning models may become outdated and hence, there is a need to update the model to maintain an acceptable predictive performance. Existing approaches to drift detection, particularly supervised and unsupervised, have inherent problems. Supervised methods detect drift based on the error rate and assume that labels are available immediately after prediction which may not be possible in several real-world scenarios. Unsupervised methods on the other hand suffer from high rate of false alarms and curse of dimensionality (i.e., the complexity faced in detecting drift across an increased number of features). To develop a more applicable technology, in this paper, we are concerned with unsupervised drift detection i.e., to detect drift independent of the labels. A recent set of experiments has revealed that it is possible to make an autoencoder learn the data distribution of a pre-defined set of classes and distinguish between the different class samples as they arrive. Based on this, we propose an Autoencoder-based Drift Detection Method (AE-DDM) which monitors the distribution of reconstruction loss in an incoming batch stream based on a thresholding mechanism to generate warnings and detect drift. AE-DDM has been tested on four synthetics (RBM, Hyperplane, Stagger and Gaussian)and one real word dataset (NOAA) with sudden and gradual drifts. AE-DDM starts generating warnings and then conforms the drift with zero delays in case of sudden drift with reduced false alarms. The classification results of ten most used machine learning classifiers on the NOAA dataset show that the detected drift is real. In this context, the potential contributions of our research are: 1) a novel framework for unsupervised drift detection based on the deep learning autoencoder technology, 2) A brief comparison of available drift detection approaches with proposed properties of an ideal drift detector, c) creation of a synthetic Gaussian dataset for drift detection research, and d) use of count threshold in conjunction with a batch threshold to combat false alarms in drift detection.

First Author / Corresponding Author

Usman Ali, PhD Scholar

School of Mathematics and Computer Science, Institute of Business Administration (IBA), Karachi, Pakistan

uali@iba.edu.pk, +92-3362090484

Second Author

Tariq Mahmood, Professor

School of Mathematics and Computer Science, Institute of Business Administration (IBA), Karachi, Pakistan

tmahmood@iba.edu.pk

Keywords: Concept drift, Machine learning, Deep learning, Autoencoder, Streaming data

1. Introduction

It has been observed that the predictive performance of machine learning models is often impacted when they are deployed in production to model the real world (Schelter et al., 2018) (Oladele, 2021) (Schröder & Schulz, 2022). Due to this variability in performance in tests in operational environments, there have been numerous concerns in wide scale applicability of these machine learning solutions in industry regarding the trust and confidence in the prediction of these models. The reason behind this performance degradation is related to the unique characteristics of the data (Wares et al., 2019). Specifically, in real-world machine learning scenarios, data characteristics and distribution change over time, termed as “non-stationary environment”, which poses challenges for classification models to predict accurately. In such non-stationary environments where the patterns in data can change very quickly, the pre-trained machine learning models can become outdated and there is a need to update the model so that it learns the new patterns in the data and stays updated. This phenomenon where the data characteristics and distribution change resulting in a need to update the model is called “concept drift” and the adaption of the model to the new changes is called “concept drift adaptation” (Gama et al., 2004) (Schröder & Schulz, 2022). The whole problem is referred to as “Concept Drift Detection and Adaptation” in the literature.

In classification-based machine learning problems, the term concept refers to the target variable. For example, in the case of credit card fraud detection, the transaction data may have a target concept as a binary attribute fraudulent with possible values “yes” and “no”. Similarly, in the case of a weather prediction application, we may have a target concept in the form of a binary variable “*rainy”* with possible values yes and no (Schlimmer & Granger, 1986). The term “Concept Drift”refers to the phenomenon when the statistical properties of the class or target variable change with time in a random manner (N. Lu et al., 2014b) and by the first time proposed by (Schlimmer & Granger, 1986), the work commonly knowns as STAGGER in literature.

To understand concept drift and its different types, let us consider a data stream observed at time S_t represented by (X_1, Y_1), (X_2 ,Y_2), (X_t, Y_t) where X_t represents the feature vector and Y_t represents the corresponding label. In classification, we have labelled historical data and the task is to predict the label for X_{t+1}. This X_{t+1} can be a single data point or a

collection of data points in the form of a small batch. For that, we train a classifier L_t on the available labelled data (X_1, Y_1), (X_2, Y_2), (X_t, Y_t) to predict the class label for the data point at S_{t+1}. Let us assume that for the classification task the target variable Y can take the values from the set $\{y_1, y_2, \ldots y_n\}$, then by Bayesian decision theory (Richard O. Duda, Peter E. Hart, 2000), the class for data point X can be determined based on maximal a posteriori probability and is given by:

$$P(y_i/X) = \frac{P(X/y_i)P\left(y_i\right)}{P(X)}$$

Where $P(y_i/X)$ is the posterior probability representing the class membership to one of the possible classes, $P\left(y_i\right)$ represents the prior probability of the class y_i, $P(X/y_i)$ is the likelihood and $P(X)$ is the evidence. Concept drift occurs between time S_t and S_{t+1} if $P_t(X, y) \neq P_{t+1}(X, y)$, where P_t is the joint probability distribution of the feature vector X and the target class label y at time S_t and P_{t+1} is the joint probability distribution at S_{t+1} (Gama et al., 2014). Hence, concept drift at S_{t+1} can be defined as the change in the joint probability distribution of features and target variables with respect to S_t.

Based on the Bayes equation, concept drift can occur in three different ways (Kelly et al., 1999); the prior probability of a class P (y) changes over time, the likelihood P (X/y) changes over time or the posterior probability distribution of a class P(y/X) changes over time. The joint probability distribution of feature set X and target class y is given by the equation $P_t(X, y) = P_t(X) \times P_t(y/X)$. Thus, the above Bayes equation can be written as $P_t(X, y) = P(X/y_i)\, P\left(y_i\right)$. Based on joint probability equation, concept drift can occur either due to change in P(X) alone, or due to a change in P (y /X) alone or due to change in both (J. Lu et al., 2019).

Based on the source of drift and its impact on the joint probability distribution of features and the target, concept drift can be virtual or real. If the joint probability distribution of features and target concept changes due to a change in P(X) (change in the distribution of features) without impacting the decision boundary P (y /X) then such type of drift is called "virtual drift" *(Figure 3b)*. Mathematically (J. Lu et al., 2019) $P_t(X) \neq P_{t+1}(X)$ and $P_t(y/X) = P_{t+1}(y/X)$. If the decision boundary P (y/X) changes due to a change in P(y) (Prior probability of the target concept) or P (X/y) (prior conditional) or due to a change in both, then such a drift is called a "real drift" (figure 3c). Mathematically (J. Lu et al., 2019):

$P_t(X, y) \neq P_{t+1}(X, y) : P_t(X) = P_{t+1}(X)$ *but* $P_t(y/X) \neq P_{t+1}(y/X)$, or

$P_t(X, y) \neq P_{t+1}(X, y) : P_t(X) \neq P_{t+1}(X)$ *and* $P_t(y/X) \neq P_{t+1}(y/X)$

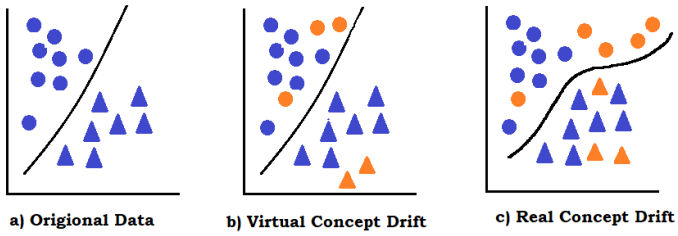


Figure 1. *Real and Virtual Concept Drift*

In case of real drift, the performance of the classifier will degrade (J. Lu et al., 2019) in terms of evaluation metrics and a model update will be required. Changes in data distribution can occur at different patterns which may or may not affect the decision boundary (real or virtual concept drift) in a supervised learning scenario. These changes can be measured using some statistical measures like mean or variance etc. Based on the change pattern, concept drift can be categorized as “sudden” (abrupt), “incremental”, “gradual” and “recurring” (Gama et al., 2014).

Apart from these drift types, other patterns of noise and outliers (blip) may also occur over a period. In case of sudden / abrupt drift, the data (distribution) changes instantly without alteration (Iwashita & Papa, 2019), for example, readings from a newly replaced sensor. In incremental drift, the data distribution shifts incrementally passing through various intermediate concepts, for example, sensors getting worn off and becoming less accurate. In case of gradual drift, the concept, or data distribution alternates between old and new levels randomly before stabilizing on the new level, for example, changes in user preferences in general. In recurring concept drift, previous concepts reoccur randomly, for example, in fashion (Gama et al., 2014).

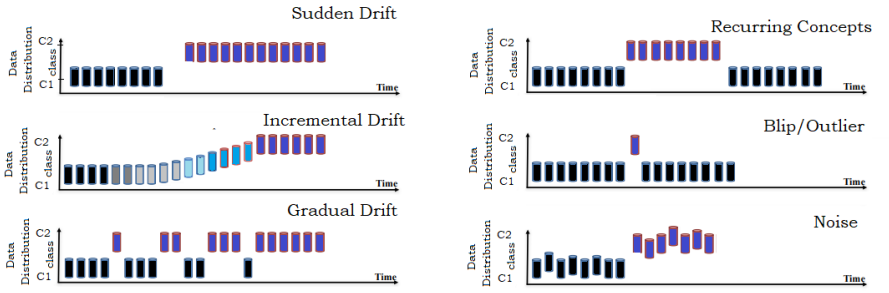


Figure 2. *Types of Concept Drift and other Change Patterns*

Thus, due to this non-stationary nature of data, concept drift detection and adaptation is required to be a vital component of machine learning solutions working in data streams to maintain their integrity, confidence, and trust levels.

Recent approaches to drift detection have some inherent problems. Supervised drift methods like DDM, RDDM, LFR and others (see Table 1) monitor the model’s performance measures like accuracy, error rate and recall, etc. to detect drift. In data streams, these labels are not available immediately after the prediction most of the time and it is not possible to

detect drift in real time using these techniques. Semi-supervised methods like SAND, ECHO, OLINDA, and others (see Section 2.2) rely on the confidence level of predictions to detect drift. These methods are classifier dependent as different classifiers will have different confidence levels. Unsupervised drift detection techniques like (Gu et al., 2016), (Qahtan et al., 2015a) and others (see Table 2) detect drift by monitoring the changes in the data distribution. These unsupervised techniques suffer from the complexity of density estimation from high dimensional data and a high false positive rate.

Motived by the recent success of deep learning-based techniques in addressing complex real-life problems and their inherent capability to deal with high dimensional data, we propose an unsupervised, Autoencoder based Drift Detection Method (AE-DDM) to detect drift in data streams with a focus to address the problems in current supervised, semi-supervised and unsupervised drift detection techniques.

An autoencoder is an artificial neural network that learns efficient data encodings for the input data by ignoring the noise to re-generate the input at the output layer (Goodfellow, 2016) (Soppin et al., 2021). It uses a set of recognition weights to map the input into a code vector at the hidden layer and then uses a set of generative weights to reconstruct the coded vector into the original input at the output layer (Hinton & Zemel, 1994). A simple auto-encoder consists of an input layer, one or more hidden layers, and an output layer of the same size as of input layer (see Figure 3).

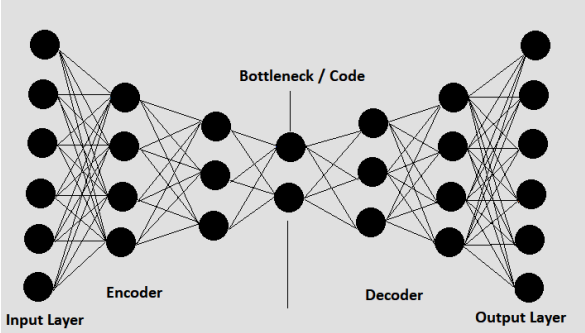


Figure 3. *Autoencoder Neural Network Architecture*

It has an encoder part that consists of an input layer and one or more hidden layers. In case of more than one hidden layer, the later hidden layers are smaller in size so that the network can encode the original input onto a smaller space. The last hidden layer in the encoder part is called the bottleneck. The decoder part is the exact replica of the encoder part. The encoder layer uses a non-linear function **f** to encode the input layer values to a latent and compressed representation as given by the equation h= f(x) while decoding the latent representation, the decoder uses another function **g** to reconstruct the original input as given by the equation g(h)= x' The reconstruction loss is defined as the mean squared difference between the original input and the reconstructed input over all training instances and can be represented as:

$$L(x, x') = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - x_i'\right)^2$$

Minimizing the reconstruction loss acts as the objective function in training an autoencoder. Consider an autoencoder with only one hidden layer. Inputs are encoded to a latent representation at the hidden layer by using a nonlinear activation function as given by the equation h= σ (Wx + b) where σ is an element-wise sigmoid or Rectified Linear Unit (ReLU) activation function, W and b are weight and bias vectors respectively, which are initialized randomly during the training phase. Decoding takes place through the decoder by using the encoded representation at the bottleneck and is given by the equation $X' = \sigma'\left(W_h' + b\right)$. During the forward pass in training through backpropagation, the difference is calculated between the original input X and reconstructed input X' and weights and biases are updated in the backward pass based on the computed error. The training continues till the number of epochs elapses and the objective function is minimized.

Some recent research papers on drift detection using autoencoders have shown the effectiveness of deep learning technology in the drift detection domain. (Jaworski, Rutkowski, & Angelov, 2020) (Yong et al., 2020b) (Jaworski et al., 2018) (Menon & Gressel, 2021) but these methods too have some limitations. The former three research papers do not consider the classification scenario and are limited to change detection in the data distribution using reconstruction error plots etc. without any explicit mechanism (algorithm) to generate warnings and detect drift. Although ADD (Autoencoder based Drift Detection) proposed by (Menon & Gressel, 2021) uses a thresholding mechanism to detect drift but it uses a single autoencoder to monitor the changes in the data distribution whereby two classes can be modeled with two different autoencoders as proposed in this research. ADD confirms drift if a single batch exceeds the threshold thus not considering the possibility of false alarms. Another limitation of this work is that the same threshold is used for different datasets and we have experimentally observed that each dataset as wells as each class data has its own reconstruction loss pattern and threshold.

Considering the bottlenecks in current supervised, semi-supervised, unsupervised, and deep learning-based unsupervised drift detecting techniques, our proposed AE-DDM methodology for drift detection as the following potential research contributions:

An Autoencoder-based Drift Detection Method (AE-DDM) for unsupervised drift detection that uses a layered mechanism to monitor the distribution of both classes in a binary classification setting.

An explicit algorithm that monitors the changes in data distribution based on a thresholding mechanism to generate warnings and then confirm the drift.

Use of two different thresholds namely batch threshold and count threshold, thus incorporating extrinsic as well as intrinsic measures to monitor the changes in the incoming batch stream which makes the drift detector robust to false alarms.

An unsupervised drift detector that can be considered as effective in detecting real drift as supervised drift detection

techniques with the added advantage of no need of true class labels.

Creation of a new synthetic gaussian dataset that can be used by the research community.

We are specifically interested in drift detection rather than adaptation. After a successful drift detection, any standard adaptation strategy can be followed to update the model. The rest of the paper is organized as follows: Section 2 describes the related work performed in the concept drift detection domain; Section 3 details our proposed methodology. Experimental results are presented in Section 4 followed by Section 5 which covers future research directions.

2. Background and Related Work

The extensive research being done in the machine learning domain and the associated challenges related to machine learning models in operations reveal that unexpected changes in data distribution make the trained model unreliable. This unreliability has been a major hindrance in wide-scale adaptation of machine learning technology in real life. To make the machine learning model updated and reliable, a drift detection mechanism needs to be incorporated as a part of the whole machine learning-based analytical framework. The need for this drift detection framework drove us to review all the available drift detection techniques and evaluate their shortcomings and to come-up with a mechanism and framework that could address this problem in the best possible way.

Concept drift detection methods can be classified as supervised, unsupervised, and semi-supervised based on the drift detection mechanism. Various review papers on concept drift detection techniques are available which classify different methods available in the literature into different categories like statistical-based, window-based, ensembles, etc. among classification-based drift detectors (Wares et al., 2019), unsupervised drift detection methods (Gemaque et al., 2020) and active and passive drift detection methods (Ditzler et al., 2015). We provide a comprehensive literature review of existing drift detection techniques in this section from supervised, unsupervised, and semi-supervised viewpoints along with some deep learning-based techniques.

2.1 Supervised Concept Drift Detection

In supervised drift detection and supervised machine learning scenarios, it is assumed that the class labels are available right after the prediction and drift are detected based on the classifier's performance. If the accuracy or other evaluation metrics like precision and recall fall below a threshold over a defined window size, then it is assumed that a drift has occurred. This assumption regarding the availability of true labels is not realistic and, in most cases, true labels are not available instantly after the prediction (Gemaque et al., 2020). In such situations, these drift detectors lose their significance in case a real and quick response is required based on the criticality of the use case. Supervised drift detection methods can detect real drift only. i.e., the P(Y|X). This method is also known as error-rate-based drift detection.

Supervised drift detection methods can be categorized into statistical, window-based, and ensembles (block-based & Incremental based) (Wares et al., 2019).

2.1.1. Statistical Methods

Statistical methods of drift detection apply statistical tests to a window of performance scores of the classifier to detect any significant changes in its performance. The Sequential Probability Ratio Test (SPRT) introduced by (Wald, 1973) based on his early works in the 1940s founded the basis of many drift detection algorithms built later on. SPRT is used to test the hypothesis (based on sequential sampling) whether the incoming data belongs to a distribution P_0 or P_1 or whether more samples are needed to reach a conclusion. Based on SPRT, Cumulative Sum (CUSUM), a continuous inspection scheme to detect the changes in the distribution parameters θ uses the residuals as an input to detect the change if the mean changes by more than a set threshold (Page, 1954). A variation of CUSUM is the Page-Hinckley (PH) test which is used to detect an abrupt change in the average of a gaussian signal (Bifet, 2017). Although CUSUM and PH are similar algorithms, they are used in different streaming environments. CUSUM is applied on residual from a predictor and is used for anomaly detection whereas PH is better suited to signal processing environments to detect sudden changes.

One of the preliminary works regarding concept drift detection was published in “Machine Learning” under the title “Incremental Learning from Noisy Data”, commonly referred to as stagger (Schlimmer & Granger, 1986) which employs concept evaluation and then refinement using Boolean characterization and changes the concept definition (characterization) if the existing definition is not able to provide satisfactory results. Stagger was observed to be sensitive to overfitting taking longer times to adopt to a new concept after being trained for a longer time on an old concept.

The drift Detection Method (DDM) (Gama et al., 2004) is one of the most popular and highly cited works in the concept drift detection domain. DDM models the error-rate of an online classifier with a binomial distribution and generates a warning level and then a drift alarm if the error-rate exceeds predefined thresholds. DDM can detect sudden or abrupt drift but suffers incase the concept is changing gradually which goes un-noticed and no warning is generated. A work like DDM is the Early Detection Method (EDDM) (Baena-García et al., 2006) which uses the distance between two classification errors to detect the drift and is able to detect slow gradual changes more effectively as compared to DDM. EDDM also defines warning levels and drift levels to detect drift like DDM bur requires 30 classification errors to occur before it can detect any drift. In some cases, it may take many examples to conceive 30 classification errors and may cause memory overflow.

As DDM and EDDM both consider the change in accuracy or error rate as an indication of drift, these techniques are sensitive to class imbalance scenarios and thus unable to properly monitor the changes in the minority class. In a class imbalance scenario, the minority class may contribute very little to accuracy like performance measures and it may take long for the occurrence of instances from the minority class. To address these problems with DDM and EDDM, (S. Wang et al., 2013) proposed a Drift Detection Method for Online Class Imbalance Learning (DDM-OCI) which uses the change in recall of the minority class (true positive rate) to detect drift. DDM-OCI assumes that the presence of class imbalance is known in advance and was evaluated on binary classification problems. It suffers from false positives as well. Another problem with DDM-OCI is that it is quite possible for a drift to occur without changing the recall of the minority class, for example in the case of a drift from an imbalance class distribution to a balance class distribution.

Another approach that works like DDM and EDDM is Statistical Tests for Equal Proportions (STEPD) (Nishida & Yamauchi, 2007) which uses two additional parameters as significance levels for warning and drift levels based on changes in the accuracy of an old and a recent window. STEPD maintains two windows; an old window that contains all the data points observed since the last drift, and a recent window with a default size 30, and compares the accuracies of these two windows by using a hypothesis test of equal proportions. A major problem with STEPD is that with a small sample size, the used statistical test was ineffective. (Bifet & Gavaldà, 2007) proposed an adaptive-window based drift detection method (ADWIN2) as an improvement of their initial work (ADWIN) which uses a variable size window that adjusts its size based on changes in the data distribution. In ADWIN, the user needs to specify only the size of one larger window W which is split up into two optimal sub-windows W_{hist} and W_{new} based on the detected significant change in the means of any such two windows. Windows size grows if no change is detected and shrinks when a drift or change is detected.

(Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds, 2015) proposed a Hoeffding's bounds Drift Detection Method (HDDM) which uses moving averages and weighted moving averages for sudden and gradual drift detection and Hoeffding's inequality to confirm a drift.

To address the problems like class imbalance and false positives faced with earlier approaches, (Heng Wang & Abraham, 2015) proposed an effective drift detection method based on monitoring all four elements of a confusion matrix namely TPR, TNR, FPR, and FNR referred to as Linear Four Rates (LFR) along with recall and precision of both minority as well as majority class. LFR works with both batch and streaming data and outperforms DDM and DDM-OCI in early drift detection, high detection rate and low false alarms.

An improvement to LFR in case of false alarms was proposed by (Yu & Abraham, 2017) commonly referred to as Hierarchical Linear Four Rates (HLFR) which uses a hierarchical hypothesis testing to detect and validate drift. Layer 1 is used to detect drift based on threshold levels of all four elements in the confusion matrix whereas layer 2 uses a permutation test to validate the drift as a true drift or a false positive. HLFR outperformed all existing methods like DDM, EDDM, and LFR in early detection and low false alarms on both balance and imbalanced datasets. Another work based on DDM is Reactive Drift Detection Method (RDDM) (Barros et al., 2017) which addresses the problem of less sensitivity to gradual drifts for larger concepts (faced by DDM) by discarding older examples from the window.

An improvement to STEPD was proposed by (Cabral & Barros, 2018) to address the problem faced by STEPD where the statistical test for equal proportions was considered inappropriate for small sample size, sparse and imbalanced data by proposing three variants based on Fisher's Exact test. Fisher Proportion Drift Detector (FPDD) uses Fisher's Exact test if errors or correct predictions are less than 5 in either window else it works like STEPD. Fisher-based Statistical Drift Detector (FSDD) uses Fisher's Exact test if errors or correct predictions are less than 5 in either window but uses a chi-square test for homogeneity of proportions instead of a statistical test of equal proportions in the other case. Fisher Test Drift Detector (FTDD) uses Fisher's Exact test explicitly. All three methods were tested on four artificial and three real-world datasets with abrupt and gradual drifts and proved to be better than STEPD and DDM in terms of drift detection and accuracy improvement.

The McDiarmid Drift Detection Methods (MDDMs) proposed by (Pesaranghader et al., 2018) maintain a sliding window to store the prediction results as 0 if the prediction is incorrect and 1 otherwise and assigns weights to these values such that $w_i < w_{i+1}$ thus giving more importance to the recent examples as compared to older ones. It maintains the mean of the sliding windows as u m and compares it with the mean of the window including the current prediction t. A significance difference between the two means which is determined by McDiarmid Inequality indicates a drift. Based on the weighing schemes, three variants namely MDDM-A, MDDM-G, and MDDM-E based on arithmetic, geometric, and Euler weighting schemes were proposed and tested on different artificial and real-word datasets and proved to be better than EDDM, CUSUM, and Page-Hinckley in terms of detection delay and classifier's accuracy.

2.1.2. Ensemble Methods

Instead of using a single classifier' error rate for concept drift detection, ensemble methods use a group of classifiers and their average error rate to detect such changes in underlying concepts. As the use of ensemble methods in machine learning has proved to give better performance as compared to a single classifier, a considerable effort has been made by the research community to detect drift in the ensemble's performance and to decide how and when to update the ensemble to incorporate the changes occurred in data distribution resulting in a change in ensemble's performance. One of the early works related to concept drift detection in ensembles was presented by (Nick Street & Kim, 2001) commonly known as Streaming Ensemble Algorithm (SEA). The SEA algorithm builds K C4.5 classifiers sequentially on a fixed chunk size to build an ensemble C. When the ensemble is full, it is used for the prediction of the incoming data chunk. It also builds a single C4.5 classifier and compares the performance of the ensemble and the single classifier. If the performance of the single classifier is better than the

ensemble, then the worst performing classifier is removed, and the new classifier is added to the ensemble. An artificially generated dataset with four sudden drifts was tested and SEA was able to recover quickly as compared to a single classifier. One drawback of SEA lies in its mechanism to remove the worst performing classifier from the ensemble without considering the recency of the data it was trained on. An ensemble with pre-determined size can still have many poor performing classifiers trained on quite older concepts. This problem was addressed by (Haixun Wang et al., 2003a)'s Accuracy Weighted Ensemble (AWE) which builds a new classifier on each arriving chunk like SEA, but instead of removing the worst performing classifier from the ensemble, it entirely builds a new ensemble including only those classifiers with MSE less than a pre-defined threshold thus resulting in a variable size ensemble real word datasets.

An obvious drawback in AWE was the silencing effect resulting in no class prediction if none of the classifiers meet the MSE threshold in case of a sudden drift. An improvement to AWE was made by (Brzeziński & Stefanowski, 2011)'s Accuracy Updated Ensemble (AUE) algorithm enabling individual classifiers to be updated directly instead of just weights updating. An incremental ensemble method based on Dynamically Weighted Majority (DWM) was proposed by (J. Zico Kolter & Maloof, 2007) which maintained a weighted pool of experts as base learners. If the ensemble made a mistake, a new expert was added to the pool, and if an expert or base learner made a mistake, its weight was decreased. If an expert continuously made mistakes, it was removed from the ensemble based on a set threshold. DWM uses the prediction from each base learner and its weight to compute the ensemble prediction.

Learn++ family is a series of algorithms using incrementally trained classifiers working on batches of data with weighted majority voting. (Polikar et al., 2001) introduced an incremental-based ensemble algorithm Learn ++ which employed various Neural Network-based weak learners to generate multiple hypotheses and then used a majority vote to classify the instance. Learn ++ used a weighting mechanism like used by Adaboost and supported incremental learning by retaining the knowledge learned from previous data without the need to store historical data. One drawback with Learn++ is that by retaining old learners trained on old patterns indefinitely may outweigh the learners trained on new data and thus drift detection may not be possible without forgetting the old information. Apart from that, instances from a new class may generate many classifiers as all the old classifiers are also retained. This problem was resolved by (M. Muhlbaier et al., 2004) who proposed a Learn ++. MT which used Dynamic Weighted Voting (DWV) to adjust the weights of base learners based on their performance on test data. If a base learner has not seen a class before then its weight in the prediction of that

Table 1. *Supervised Drift Detection Methods*

S.No	Drift Detection Technique	Type	Reference
1	STAGGER	Statistical	(Schlimmer & Granger, 1986)
2	FLORA	Statistical	(Widmer, 1996)
3	CVFDT (Concept Adapting Very Fast Decision Trees)	Window-based	(Domingos & Hulten, 2000)
4	SEA (Streaming Ensemble Algorithm)	Ensemble-based	(Nick Street & Kim, 2001)
5	AWE (Accuracy Weighted Ensembles)	Ensemble-based	(Haixun Wang et al., 2003b)
6	DDM (Drift Detection Method)	Statistical	(Gama et al., 2004)
7	ACE: Adaptive Classifiers-Ensemble System	Statistical	(Nishida et al., 2005)
8	EDDM (Early Drift Detection Method)	Statistical	(Baena-García et al., 2006)
9	STEPD (Statistical Test of Equal Proportions)	Statistical	(Nishida & Yamauchi, 2007)
10	ADWIN (Adaptive Windowing)	Window-based	(Bifet & Gavaldà, 2007)
11	AUC (Accuracy Updated Ensembles)	Ensemble-based	(Brzeziński & Stefanowski, 2011)
12	DDM-OCI (DDM for Online Class Imbalance Learning)	Statistical	(S. Wang et al., 2013)
13	E-CVFDT (Efficient CVDFT)	Window-based	(G. Liu et al., 2013)
14	LFR (Linear Four Rates)	Statistical	(Heng Wang & Abraham, 2015)
15	FHDDM (Fast Hoeffding's Drift Detection Method)	Window-based	(Pesaranghader & Viktor, 2016)
16	SAND (semi-supervised Adaptive Novel Class Detection)	Ensemble-based	(Haque, Khan, & Baron, 2016)
17	ECHO (Efficient Handling of Concept Drift and Evolution)	Ensemble-based	(Haque, Khan, Baron, et al., 2016)
18	RDDM (Reactive Drift Detection Method	Statistical	(Barros et al., 2017)
19	HLFR (Hierarchical Linear Four Rates)	Statistical	(Yu & Abraham, 2017)
20	FPDD (Fisher Proportions Drift Detector	Statistical	(Cabral & Barros, 2018)
21	FSDD (Fisher-based Statistical Drift Detector	Statistical	(Cabral & Barros, 2018)
22	FTDD (Fisher Test Drift Detector)	Statistical	(Cabral & Barros, 2018)
23	MDDM (McDairmid Drift Detection Method)	Statistical	(Pesaranghader et al., 2018)
24	DWM (Dynamic Weighted Majority)	Ensemble-based	(Jeremy Z. Kolter & Maloof, 2003)
25	Learn++	Ensemble-based	(Polikar et al., 2001)
26	Learn ++.MT	Ensemble-based	(M. Muhlbaier et al., 2004)
27	Learn++ .NC (New Class)	Ensemble-based	(M. D. Muhlbaier et al., 2009)
28	Learn ++. NSE (Non-Stationary Environment)	Ensemble-based	(M. D. Muhlbaier & Polikar, 2007)
29	Learn ++. NIE (Non-Stationery and Imbalance Environment)	Ensemble-based	(Ditzler & Polikar, 2010a)
30	Learn ++. CDS (Concept Drift with SMOTE)	Ensemble-based	(Ditzler & Polikar, 2013b)

instance is reduced. An improvement to Learn++.MT was made by Learn++.NC (New Class) (M. D. Muhlbaier et al., 2009) which uses Dynamically Weighted Consult and Vote (DW-CAV) mechanism where individual base learners consult with each other before voting based on which they either update their weight or withdraw from voting. All these Learn++ algorithms assume a stationarity data environment and do not detect and adapt concept drift explicitly. (M. D. Muhlbaier & Polikar, 2007) proposed another Learn++ algorithm Learn++NSE, an incremental learning algorithm in Non-Stationary Environments which modifies the weights of the base learners based on their performance on the current data.

Learn++NSE retains all the previous learners meanwhile generating a new classifier on the new data chunk that becomes available. To handle the class imbalance problem, (Ditzler & Polikar, 2010b) proposed Learn++NIE (Non-Stationary and Imbalanced Environments) which builds new -sub-ensembles on each new data chunk and uses recall to evaluate the ensembles and formulate the composite hypothesis. (Ditzler & Polikar, 2013a) proposed Learn++.CDS (Learn++ for Concept Drift with SMOTE) which instead of using recall or F-Measure to handle class imbalance, uses Synthetic Minority Oversampling Technique (SMOTE) to add more samples of the minority class. It has been found that the performance of the Learn++ series of algorithms is heavily dependent on the base learners (Liao et al., 2016). Although the approach in Learn ++. NSE favors newly created classifiers still it can be used for concept drift detection and can from a basis for further improvements (Wares et al., 2019). Supervised drift detection techniques have been summarized in Table 1.

The dependence of drift detectors on the availability of class labels in supervised drift detection techniques and the associated cost and delays in the availability of the true labels in real-world applications drew the attention of the research community towards framing techniques that either do not rely on class labels or have limited reliance. Some of these works require labeled data for the initial training of classifiers and initialization of drift detectors and the classifier's confidence levels are only needed for drift detection. These techniques have been categorized as semi-supervised drift detection methods (Iwashita & Papa, 2019) and a brief overview is presented in the next section.

2.2. Semi-Supervised Drift Detection

Semi-supervised drift detection methods require limited labeled data for the initial training of the classifier or the ensemble. Drift is detected based on changes in the level of confidence of the prediction. To update the model, only labels are required for those instances where the confidence level is low.

(Haque, Khan, & Baron, 2016) introduced a semi-supervised drift detection method in their work Semi-Supervised Adaptive Novel Class Detection (SAND) which uses the classifier's confidence to detect concept drift and the chunk size. SAND employs a semi-supervised ensemble classifier which needs only limited data for model updating where the confidence level is below some defined threshold. SAND is also able to detect outliers and to detect a novel class if such outliers are in number and are close to each other. Due to change detection after calculating each confidence, SAND becomes inefficient in terms of execution time. To overcome this, (Haque, Khan, Baron, et al., 2016), proposed ECHO (Efficient Handling of Concept Drift Evolution over Stream Data) which uses dynamic programming and performs change detection selectively. Experimental results showed improvements in execution time and accuracy as compared to SAND. (Pinagé et al., 2020) proposed a semi-supervised drift detection method that uses self-annotation and ensemble learners like SAND and uses dynamic classifier selection in an online setting.

(Iwashita & Papa, 2019) which provided a comprehensive overview on concept drift learning have categorized the existing methods to drift detection and handling as supervised, semi-supervised, and unsupervised. Among these, semi-supervised approaches include Online Novelty and Drift Detection Algorithm (OLINDA) (Spinosa et al., 2007) which uses k-means clustering to detect drift and Enhanced Classifier for data Streams with novel class Miner (ECSMiner). Other techniques discussed or more focused on novel class detection than drift detection. The categorization of drift detection

techniques into semi-supervised methods is not clear and is indeed in its infancy. Semi-supervised term in drift detection stems from the adaptation mechanism based on limited labeling proposed along with drift detection mechanism in some of the approaches. Due to this, SAND and DSDD have been categorized as unsupervised by (Gemaque et al., 2020).

2.3. Unsupervised Drift Detection

The limitations in supervised and semi-supervised drift detection methods like dependency on class labels or classifier's confidence levels drove the research community towards unsupervised drift detection techniques aimed at detecting concept drift by monitoring the changes in the data distribution (Gemaque et al., 2020). Algorithms in unsupervised drift detection methods usually maintain two windows namely the reference (historical) window and detection window (new data) and use a distance measure to quantify the difference between the distribution of historical data and new data. The historical window is kept fixed while the detection window is sliding. (see Figure 4). If the difference in data distributions of the two windows is significant, then a drift is detected with an indication of drift points (J. Lu et al., 2019). Unsupervised drift detection is also known as "data distribution-based drift detection".

Although various review papers (Khamassi et al., 2018), (Hu et al., 2020), (J. Lu et al., 2019) and (Iwashita & Papa, 2019) on drift detection discuss unsupervised drift detection methods in some detail along with supervised or ensemble methods, (Gemaque et al., 2020) provides an explicit comprehensive overview of only unsupervised methods. The author has divided these methods broadly into batch-based and online-based methods based on the environment under consideration.

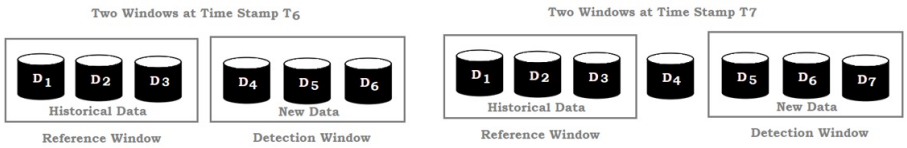


Figure 4. *Unsupervised Drift Detection*

Most of the unsupervised drift detection methods maintain a reference window that contains data instances on which the most recent classifier has been trained and a detection window that is under consideration form a drift detection point of view. If drift is detected based on a batch of data elements as a detection window, then these methods are called batch-based drift detection methods and if drift is detected based on each individual instance in the detection window, then it is labeled as online drift detection.

2.3.1. Batch-Based Unsupervised Drift Detection Methods

Batch-based unsupervised drift detection methods detect drift based on changes in the data distribution of a fixed-size or dynamic-size batch. Some of these methods use the complete batch for drift detection while others use a partial batch or

a subset of the samples in the detection window. A general framework for un-supervised batch-based drift detection is given in Figure 5 below (Gemaque et al., 2020).

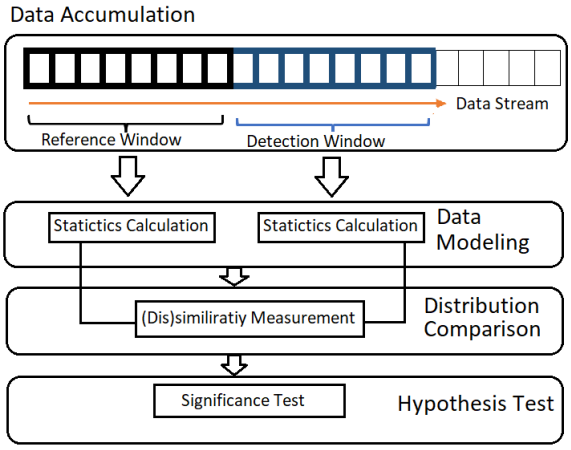


Figure 5. *A General Framework for Unsupervised Batch-based Drift Detection*

One of the initial works related to unsupervised batch-based drift detection methods is based on margin density estimation and is referred to as Margin Density Drift Detection (MD3) (Sethi & Kantardzic, 2015). Margin density is the ratio of samples lying in the margins to the total number of samples. MD3 uses a trained SVM classifier with a known set of minimum and maximum density values $[\ _{min},\ _{max}]$ along with a threshold θ . For an incoming batch of data, MD3 checks every instance whether it lies in the margins or not, and counts a number of instances lying in the margin. It compares this count with previous $_{min}$, $_{max}$ values and updates $_{min}$, $_{max}$ for the current batch. If $_{max}$ - $_{min} > \theta$ then a drift is detected. An increase or decrease in margin density indicates a drifting condition.

One major problem with MD3 is that it is classifier dependent and works only with SVM. (Sethi & Kantardzic, 2017) proposed a classifier-independent version of MD3 using an ensemble of classifiers. For classifiers not having an explicit notion of margin, generalization regions or blind- spots, which are regions with high uncertainty, can be monitored for drift detection. This modified MD3 approach tracks these blind spots of multiple classifiers and if there is a considerable disagreement between individual classifiers then it is an indication of high uncertainty. It uses margin density in the case of linear SVM and blind-spot density in the case of other classifiers like decision trees and nearest neighbors as base learners in an ensemble. A significant difference between the margin or blind-spot densities of two data windows (reference and detection window) indicates a drift in the dataset. This modified MD3 technique is referred to as MD3-RS (Random Subspace) in comparison with the former MD3-SVM. MD3-RS requires labeled samples to confirm the detected drift which makes this approach practically less applicable in data streaming environments as such labels are not readily available.

Sometimes the changes in the data distribution are not due to the non-stationarity nature of the system under study but

are made deliberately by attackers to break the drift detection system and to stay un-caught. This type of drift is called adversarial drift where the attackers exploit the vulnerabilities of the classifier or defender system to gain access to the system by beating the detection mechanism. (Sethia & Kantardzic, 2018) proposed a predict-detect streaming framework - an adversarial drift detection mechanism- to detect such adversarial attacks on the classifier or drift detection system. The proposed method uses two independent classifiers $C_{predict}$ and C_{detect} trained on disjoint features subset where C_{detect} is completely hidden from the adversary. When the adversary learns the vulnerabilities of $C_{predict,}$ it creates attack samples hitting the classifier's blind-spots or weak areas resulting in a high level of disagreement between $C_{predict}$ and $C_{detect.}$ When the disagreement exceeds a set threshold a drift is detected, and labels are requested for the samples having disagreement to confirm the drift.

(Costa et al., 2018) proposed an unsupervised drift detection method based on active learning which does not require labeled samples for drift detection. Initially, a classifier is trained on fully labeled training data, and minimum and maximum densities of the most significant instances based on virtual margins and fixed uncertainty (Zliobaite et al., 2014) are computed as a reference by using k-fold cross-validation. For each detection window, it monitors the density of the most significant instances, and a drift is detected if the different exceeds the drift threshold. After a drift has been detected, a reaction module is triggered which updates the classifier based on the labeled samples (when they become available) of the current batch. This new training updates the minimum and maximum values of densities of the most significant instances. This work is referred to as Drift Detection Method based on Active Learning (DDAL) and can be used with any classifier providing confidence in its prediction.

All un-supervised drift detection methods described above namely MD3, MD3-RS, Predict-Detect, and DDAL use partial batch to detect concept drift. Apart from these methods detecting concept drift using partial batch, several methods have been proposed that use complete batch to detect concept drift and are referred to as whole-batch detection methods.

(Bashir et al., 2017) proposed an unsupervised drift detection technique in activity recognition called U_{Detect} which uses the change in the distribution of class membership as predicted by the classifier compared to reference distribution from labeled training data. Although this method does not require the truth labels, it depends on the prediction of a classifier. A classifier is required to predict the labels to detect the drift in the dataset. A drift detection method based on dissimilarity in regional densities was proposed by(A. Liu et al., 2018) which uses K-Nearest Neighbour (KNN) to identify the variation in regional densities and is referred to as NN-DVI. (Li et al., 2019) proposed a Fast and Accurate Anomaly Detection (FAAD) method to detect anomalies in multidimensional sequence data without being affected by the occurrence of concept drift. FAAD, being an application-dependent approach toward anomaly detection focuses on anomaly detection rather than drift detection. (André G. Maletzke et al., 2018) proposed an extended version of SQSI (Stream Quantification by Score Inspection) (Andre G. Maletzke et al., 2017) that uses instance selection to detect the changes in the distribution of classes in an unsupervised way. In coming data is divided into batches and predicted by a classifier. KS test is performed to test whether the class distribution is same as the references batch or not. If test statistic < 0.001 then another KS test is performed by transforming both reference and detection windows so that both windows have same mean and standard deviation. If this test fails, then drift is confirmed, and labels are requested for selected instances to update the model with

current detection window. SQSI-IS needs classification by a classifier which can predict the confidence intervals or class probabilities before it can detect drift, thus cannot be regarded as a fully unsupervised method.

Apart from batch-based methods, online unsupervised drift detection methods also have been proposed by the research community. These methods have been reviewed next.

2.3.2. Online-Based Unsupervised Drift Detection Methods

In contrast to batch-based methods which accumulate instances into a batch of a certain size and then use some drift detection algorithm to compare the current batch with some reference batch, online methods perform drift detection on the arrival of every single instance. For initialization, these methods need to accumulate some instances at the start. After initialization, a reference window is maintained which can be fixed on training instances or sliding over the incoming stream of data. Another sliding window is defined over the incoming data stream called detection window. To detect drift, distributions of both reference and detection windows are compared and a statistical test is performed to check the significance of similarity or dissimilarity. Figure 6 below shows a general framework of online-based drift detection methods (Gemaque et al., 2020).

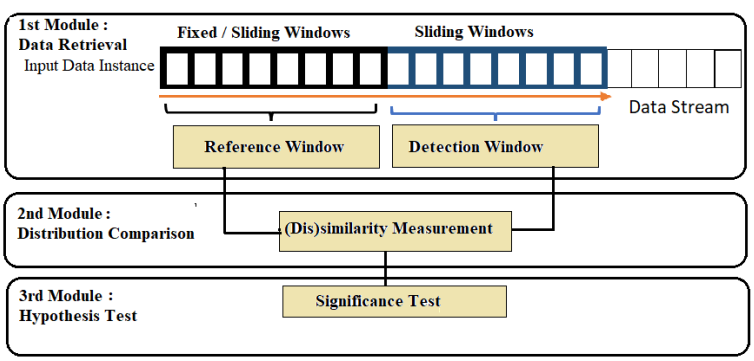


Figure 6. *A General Framework of Unsupervised Online-Based Drift Detection Methods*

Based on the reference window as either fixed or sliding, online-based methods can be categorized as fixed reference window or sliding reference window and are reviewed in the next two sub-sections.

2.3.2.1. Fixed Reference Window

Online-based drift detection techniques based on a fixed reference window maintain the reference window (based on training data) fixed. (Dos Reis et al., 2016b) proposed an online-based drift detection technique based on Incremental Kolmogorov Smirnov (IKS-bdd) test to check whether two samples belong to the same data distribution or not. With the addition of each new sample, KS test is performed to test the similarity / dissimilarity between the two windows. In case the null hypothesis is rejected, drift is detected. IKS-bdd tries to detect drift in each attribute of the detection window rather

than using a multivariate. (Koh, 2016b) proposed CD-TDS (Change Detection in Transactional Data Streams) which detects local as well as global drifts in item sets. Changes in items only represent local drift whereas changes among connections between items represent a global drifting scenario. The local drift detection module maintains two windows using Hoeffding's bound. If the difference between the means of two windows is greater than Hoeffding's bound defined value ϵ, then drift is detected. To detect global drift, a pairwise statistical test is used to compare the tree structure of both windows using Levenshtein edit distance.

2.3.2.2. Sliding Reference Window

In Sliding Reference Window approaches to online drift detection, the reference window slides over the incoming data stream. (Lughofer et al., 2016b) proposed an online unsupervised drift detection that uses a modified version of Page-Hinkley test to detect drift and can be abbreviated as OMV-PHT (Online Modified Version of Page-Hinkley Test) (Gemaque et al., 2020). The proposed technique is intended to detect drift in semi-supervised as well as completely unsupervised ways. In the case of the semi-supervised part, active learning is used to select the most significant samples in terms of classifiers' confidence in prediction, and labels are sought for the most uncertain predictions, and in the case of the unsupervised version, the change in the distribution of classifier's confidence is monitored with the help of a modified version of Page Hinkley test to detect and signal the decline in a performance like accuracy etc.

A novel class detection technique as well as a non-parametric multidimensional drift detection method (NM-DDM) was proposed by (Mustafa et al., 2017) which is based on denoising autoencoders and uses the log likelihood ratio between a reference and a detection window for each feature for drift detection. If the highest ratio for a feature is greater than the pre-defined threshold, a drift is detected. (de Mello et al., 2019) proposed a drift detection algorithm called Plover which provides theocratical learning guarantees and uses the divergence between the reference and detection window to detect drift. To detect drift, different measure functions like Power Spectrum and Statistical Moments are used. To use Plover, data should be identically and independently distributed, and measure functions should be independent of the input data. (Kim & Park, 2017) proposed a Distribution based Drift Detection Approach (DbDDA) which uses the classifier's posterior estimate to detect concept drift. DbDDA uses a sliding detection window and both fixed and sliding reference windows as well as an ensemble reference window in its variants.

All the above unsupervised drift detection techniques have been reviewed and mentioned in (Gemaque et al., 2020). Another interesting and comprehensive review of concept drift detection has been provided by (J. Lu et al., 2019) who have termed unsupervised drift detection techniques as data distribution-based methods. These techniques have been summarized in Table 2 from serial number 15 to 25.

Table 2. *Unsupervised Drift Detection Techniques*

S.No	Drift Detection Technique	Type	Detection Mechanism	Reference
1	MD3 (Drift Detection using Margin Density)	Partial Batch	SVM Margin Density	(Sethi & Kantardzic, 2015)
2	MD3-RS (MD3 using Random Subspace Model	Partial Batch	Blindspot Density	(Sethi & Kantardzic, 2017)
3	Predict-Detect (Handling Adversarial Concept Drift)	Partial Batch	Disagreement Density	(Sethia & Kantardzic, 2018)
4	DDMAL (Drift Detection Method Based on Active Learning)	Partial Batch	Density of most significant Instances	(Costa et al., 2018)
5	UDetect (Unsupervised Change Detection for Activity Recognition)	Whole Batch	Classifier's Predicted Class Density	(Bashir et al., 2017)
6	NN-DVI (Nearest Neighbour based Density Variation Identification)	Whole Batch	KNN based Regional Densities	(A. Liu et al., 2018)
7	FAAD (Fast and Accurate Anomaly Detection)	Whole Batch	Anomaly Density	(Li et al., 2019)
8	SQSI-IS (Stream Quantification by Score Inspection - Instance Selection)	Whole Batch	Class Densities	(André G. Maletzke et al., 2018)
9	IKS-bdd (Incremental KS based drift detection)	Online - FRW*	Feature Densities	(Dos Reis et al., 2016a)
10	CD-TDS (Change Detection in Transactional Data Streams)	Online - FRW	Sample means / Edit Distance	(Koh, 2016a)
11	OMV-PHT (Online Modified Version Page Hinkley Test)	Online -SRW**	Change in Classifier's Confidence	(Lughofer et al., 2016a)
12	NM-DDM (Nonparametric Multidimensional Drift Detection Method)	Online -SRW	Log Likelihood Ratio	(Mustafa et al., 2017)
13	Plover (On Learning Guarantees to Unsupervised Concept Drift Detection on Data Streams)	Online -SRW	Divergence	(de Mello et al., 2019)
14	DdDDA (Distribution based Drift Detection Approach)	Online FRW/SRW	Classifiers Posterior Estimates / Confidence	(Kim & Park, 2017)
15	An Information-Theoretic Approach to Detecting Changes in Multi-Dimensional Data Streams	Whole Batch	KL Divergence	(Dasu et al., 2006)
16	Concept Drift Detection via Competence Models	Whole Batch	Competence Distance	(N. Lu et al., 2014a)
17	Detecting Change in Data Streams	Online- FRW	Relativized Discrepancy	(Kifer et al., 2004)
18	Statistical Change Detection for Multi-Dimensional Data	Whole Batch	log likelihood	(Song et al., 2007)
19	Concept Drift Detection Based on Equal Density Estimation	Online FRW/SRW	Density Estimation	(Gu et al., 2016)
20	Sync Stream (Prototype-based Learning on Concept-drifting Data Streams)	Online SRW	PCA / Prototype-Tree	(Shao et al., 2014)
21	A PCA-Based Change Detection Framework for Multidimensional Data Streams	Whole Batch	PCA based Density Estimation	(Qahtan et al., 2015b)
22	A pdf-Free Change Detection Test Based on Density Difference Estimation	Online- FRW	Least Square Density Difference	(Bu et al., 2018)
23	An incremental change detection test based on density difference estimation	Online -SRW	Least Square Density Difference	(Bu et al., 2017)
24	Regional Concept Drift Detection and Density Synchronized Drift Adaptation	Whole Batch	Local Drift Degree	(A. Liu et al., 2017)
25	A Concept Drift-Tolerant Case-Base Editing Technique	Whole Batch	Competence Distance	(N. Lu et al., 2016)

2.4. Deep Learning based Methods for Drift Detection

Some recent works on drift detection are based on deep learning methods including autoencoders and Restricted Boltzmann Machine (RBM). (Yong et al., 2020a) used Bayesian autoencoders to detect drift in sensors data of an

industrial environment. Three different measures including reconstruction loss, aleatoric and epistemic uncertainties have been used to detect drift. In the case of a real drift (drift already present in data due to sensors degrading conditions) all three measures show a considerable deviation. This paper, however, deals with drift detection in an unsupervised way i.e., without considering the presence and impact of different classes in the data.

(Jaworski et al., 2018) applied RBM on a synthetic binary dataset generated with the help of RBM to detect sudden and gradual drift. Two indicators, reconstruction loss, and free energy has been used to detect drift in the data. In the case of drift, both measures indicate a considerable difference from the normal data. (Jaworski, Rutkowski, Angelov, et al., 2020) applied autoencoders to the same dataset and used reconstruction error and cross-entropy using autoencoders and proved that sudden and gradual drift can be detected with autoencoders. However, both works deal with drift detection in an unsupervised way without considering the impact of changes in data distribution on the classifier's boundary.

All the above three techniques are focusing on virtual drift i.e., detecting changes in data distribution without considering the impact of change on the classifier. Autoencoder-based Drift Detection (ADD) (Menon & Gressel, 2021) is another recent work that uses autoencoders to detect drift in phishing data. To detect drift, reconstruction loss is compared with user-defined thresholds and Hoeffding's tree is used as a classifier. The author has shown significant improvement in accuracy after drift detection and adaptation. However, this work too has some limitations like using a single autoencoder to model the distribution of all the classes in a classification dataset, use of a single threshold for all the datasets, and not considering the possibility of false positives. The table below shows the summary of deep learning-based methods for drift detection.

Table 3. *Deep Learning based Drift Detection Techniques*

S.NO	Research Paper	DL Type	Cost Function	Detection Mechanism	Drift Types
1	Concept Drift Detection using Autoencoders in Data Streams Processing	Vanila AE	Reconstruction Error, Cross Entropy Loss	Trend Analysis	Sudden, Gradual
2	Concept Drift Detection in Phishing using Autoencoders	Vanila AE	Reconstruction error	Thresholds	Sudden, Gradual
3	Bayesian Autoencoders for Drift Detection in Industrial Environments	Bayesian AE	Reconstruction Error, Epistemetic Uncertainty, Aleatoric Uncertainty	Boxplots, 3D Plots	Real, Virtual
4	On Applying Restricted Boltz Machines to Active Concept Drift Detection	RBM	Reconstruction Error, Free Energy	Line Plots	Sudden, Gradual

2.5. Summary

A summary of four drift detection approaches reviewed in this section is shown in Table 4 (all columns are self-

explanatory). Deep Learning based approaches can work independently of the availability of true labels for drift detection. These techniques are also independent of the classifier used for prediction in supervised machine learning scenarios and have the inherent capability to summarize multidimensional data. As limited work has been done in this area till now, this opens a new research direction in the drift detection domain.

Table 4. *Summary of Drift Detection Techniques*

Comparison Criteria	Detection Approach			
	Supervised	Semi-Supervised	Unsupervised	Deep Learning Based
Dependency on true labels in detection		×	×	×
Dependency on true labels in adaptation		Limited		
Classifier dependency	×		×	×
False alarms	×			Under research
Types of drift detected	Surely Real	Probably Real	Probably virtual	Under research
Signal Analyzed	Univariate	Univariate	Multivariate	Univariate
Ability to summarize high dimensional data	×	×	Limited	Inherent
Need to check statistical significance	×	×		

This review helped us to craft the properties of an idea and effective drift detector as given below:

An ideal drift detector should be as reliable as is drift detection based on supervised drift detection techniques

Drift detection process should not be dependent on class labels as in the case of unsupervised drift detection techniques.

Any available labeled should be leveraged in training a drift detector like in semi-supervised drift detection techniques

A mechanism like confidence prediction should be incorporated without the disadvantage of different confidence levels by different classifiers

The power of deep learning should be used to summarize high dimensional data

Keeping in view the derived properties of an ideal drift detector, we propose an Autoencoder-based Drift Detection Method (AE-DDM) in the next section.

3. Proposed Methodology

Considering the issues and challenges with existing supervised, semi-supervised and unsupervised drift detection techniques like requirements of actual class labels in the case of supervised techniques, dependence on the classifier to predict the confidence levels in the case of semi-supervised methods and class-independent nature of detected drift, high

false positive rate and difficulty to learn the difference in high dimensional data distributions perfectly in case of current unsupervised techniques, we propose an Autoencoder based Drift Detection Method (AE-DDM) to address and overcome the aforementioned issues.

3.1. AE-DDM Conceptual Framework

Autoencoders are deep learning models that have the capability to learn the encoded representation at the bottleneck and then reproduce the input at the output layer. Using this capability of autoencoders, we can make the autoencoders learn the data distribution of each class in a classification problem. We have developed a novel and generic approach for drift detection using autoencoders for a binary classification problem.

Our initial version of AE-DDM follows a batch-based unsupervised drift detection mechanism. At an architectural level, it has three components; an offline component where we train the autoencoders on each class data and compute thresholds; an ensemble component which defines the order of the autoencoders, i.e., which autoencoder to place at layer 1 and which one at layer2; and an online component where data arrives in batches and drift detection is performed for the whole batch data stream as shown in figure 7.

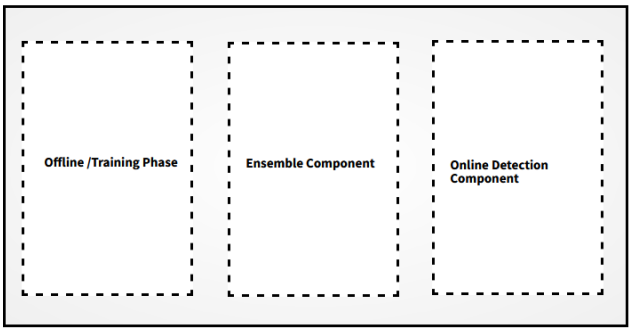


Figure 7. *AE-DDM Framework*

3.1.1. Offline / Training Phase

In the offline or training phase of AE-DDM, two autoencoders are trained on normal (non-drifted) labeled data; one for the positive class and the other one for the negative class. Validation data is divided into batches and thresholds are computed.

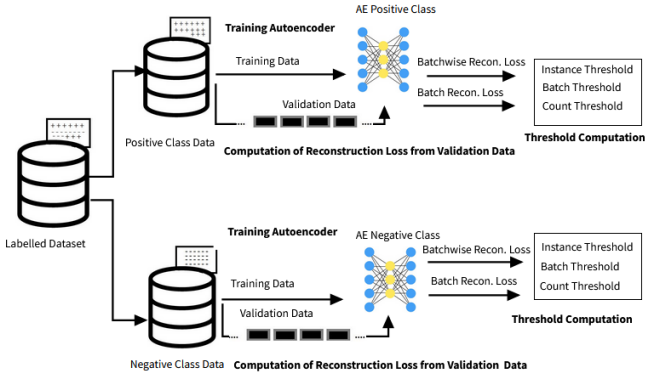


Figure 8. *Offline Phase of AE-DDM*

Offline or Training Phase has the following components :

a. Training Autoencoders
b. Computation of Reconstruction Loss from normal(non-drifted) data
c. Threshold Computation

a. Training Autoencoders

Available labelled dataset is separated into positive and negative class data based on labels in the dataset and the class column is removed. Both positive and negative class data are each split up into training (training & validation) and validation sets. Two autoencoders are trained; one on positive class data and the other on the negative class. We have used the simplest possible structure of an autoencoder as shown in Table 5.

Table 5. *Autoencoder Architecture used in AE-DDM*

Type of Autoencoder	Vanilla Autoencoder
Number of Layers	3 (Input, Bottleneck, Output)
Input Dimension	Number of Features in the Dataset Excluding Class
Bottleneck Dimension	One-third of Input Dimension
Activation	Sigmoid
Optimizer	Adam
Loss	MSE
Batch Size	32

b. Computation of Reconstruction Loss from Normal (non-drifted) Data

The validation data of each class is divided into batches of size 32 and passed to the respective autoencoder. Reconstruction loss for each instance in a batch, called batchwise reconstruction error, and average reconstruction loss for the whole batch, called batch avg reconstruction error, are computed using Algorithm1.

Input

- **batch** // A batch of data
- **encoder** // Trained Autoencoder

Output

- **inst_loss_list= []** // A list containing reconstruction loss values for each instance in a batch
- **avg_batch_loss** // Average reconstruction loss for the whole batch

Process

1. **loss_sum**=0 // Sum of reconstruction loss values for the whole batch
2. for each instance **row** in **batch**, do
   - 1.1 **pred**= get prediction from **encoder** on **row**
   - 1.2 **loss**= reconstruction loss on **pred**
   - 1.3 append this **loss** to **inst_loss_list**
   - **1.4 loss_sum=loss_sum+loss**
3. **avg_batch_loss=loss_sum/batch_size**
4. return **inst_loss_list, avg_batch_loss**

Algorithm 1. *Compute Instance Loss and Batch Loss (Batch Average Reconstruction Loss)*

c. Threshold Computation

Using the reconstruction loss values, three different thresholds namely instance threshold, batch threshold and count thresholds are computed for both positive class and negative class data. These thresholds are described below:

Instance Threshold

Instance threshold on normal (non-drifted) data is computed as mean+3 (standard deviation) of reconstruction error values. It is assumed that any data point with reconstruction error values greater that the instance threshold will be a drifted data point. Although reconstruction error values are assumed to follow a normal distribution, it is observed that with a small batch size, this assumption may not hold true for all the batches depending on the distribution of the data in a batch. To combat this, we have used a configurable parameter N, which is the number of batches to be considered for instance threshold computation. For N batches, we compute the instance threshold and then take the average over N. Algorithm2 has been used to compute instance threshold

Input

- N: Number of batches to Consider
- Reconstruction Loss Values for N Batches

Output

- Instance Threshold

Process

1. **threshold_loss_list**= [] // An empty list to hold threshold batch loss for N batches
2. For i=1 to N do
   - 1.1 Compute mean **m** // mean reconstruction loss for a batch
   - 1.2 Compute **sigma** // Standard Deviation
   - 1.3 **thresh=m+3(sigma)**
   - 1.4 Append **thresh** to **threshold_loss_list**
3. **Instance Threshold** = sum(**threshold_loss_list**) / **N** // Compute Average over N Batches
4. Return **Instance Threshold**

Algorithm 2. *Compute Instance Threshold*

Count Threshold

For each batch in validation data, AE-DDM compares the reconstruction loss value for each instance with the instance threshold. The number of instances in a batch exceeding the instance threshold is counted so that AE-DDM gets a dictionary of values with keys as the batch ids and values as the number of instances in that batch exceeding the instance threshold. ***Count Threshold*** is taken as the maximum of the dictionary values. It is assumed that in the case of normal (non-drifted) data, the number of instances exceeding the instance threshold will be less than this count threshold. Any batch where this count threshold exceeds is assumed to be a batch with drifted data.

Batch Threshold

Batch threshold is computed using batch average reconstruction loss values. It is taken as mean+ 3 (standard deviation) over batch average reconstruction error values for all the batches over the entire validation data. It is assumed that for large validation data and number of batches, batch average reconstruction error values follow a normal distribution.

Input

- **avg_batch_loss // a list containing average reconstruction error for all the batches in validation data**

Output

- **batch_threshold** // batch threshold

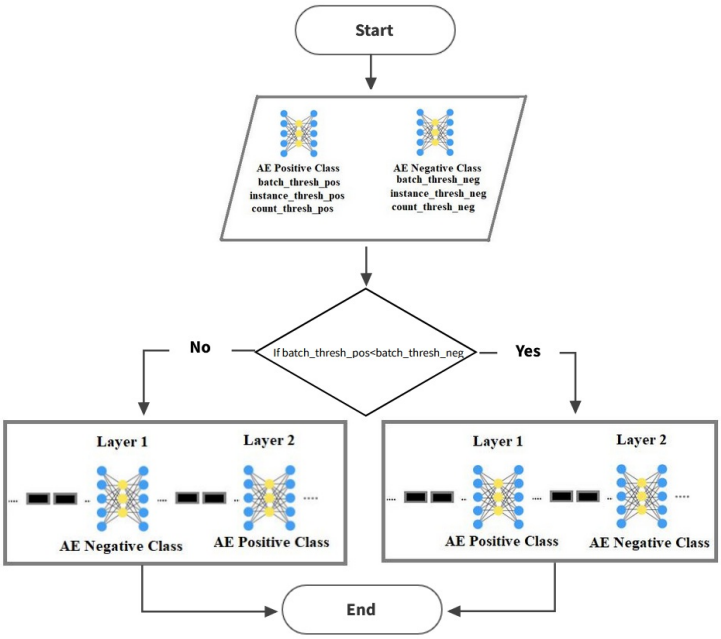
Process

1. **mean**= mean(**avg_batch_loss**) // Mean
2. **sigma**=std((**avg_batch_loss**) // Standard Deviation
3. **batch_threshold** =**mean**+ 3 * **sigma**
4. return **batch_threshold**

Algorithm 3. *Compute Batch Threshold*

3.1.2. Ensemble Component

Ensemble component decides which autoencoder to place at layer one and which one at layer 2. Autoencoder with a lower batch threshold is placed at layer1 while autoencoder with a higher batch threshold is placed at layer 2 and corresponding thresholds (batch, instance, and count) are used at respective layers as shown in Flowchart 1. This assembling of autoencoders is once carried out initially when autoencoders are trained and thresholds are computed from validation data. Every time drift is detected, autoencoders will be re-trained on the most recent data, thresholds will be recomputed, and re-ensembling will take place.



Flowchart 1. AE-DDM Ensemble Component Flow Chart

3.1.3. Online Detection Component

Detection component in AE-DDM works in batch mode. It receives data in batches, processes each batch, and keeps a record of each batch. The available data stream is divided into batches of size 32 and passed to Algorithm 5 along with layer 1 and layer 2 trained autoencoders with corresponding batch thresholds, count thresholds, and instance thresholds (already computed in the offline component). Circles in Figure 9 indicate the order of execution of algorithms 4, 5, and 6. Algorithm 5 executes first and passes each batch along with layer 1 trained autoencoder to Algorithm 1 which computes reconstruction loss for each instance in the batch. Each reconstruction error value is compared with the instance threshold and the number of instances exceeding the layer 1 instance threshold (exceed_count_layer1) is counted. The average batch reconstruction error (avg_error_layer1) for each batch is also computed.

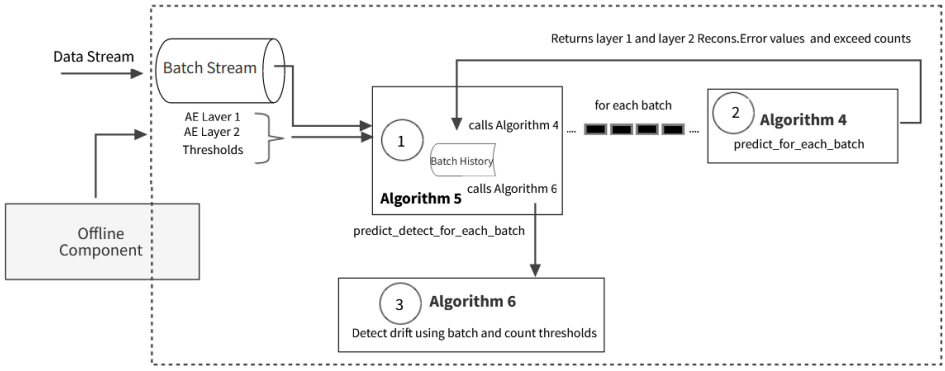


Figure 9. *Online Detection Component*

If avg_error of a batch exceeds layer_one_batch_thres and exceed_count_layer1 exceeds layer_one_count_threshold, then that batch is passed to layer_two_encoder. For each instance, reconstruction error is computed from layer_two_encoder prediction and this error is compared with layer2_ins_thresh to compute exceed_count_layer2. Average batch reconstruction error avg_error_layer2 is also computed for this batch and if this avg_error_layer2 and exceed_count_layer2 both exceed their respective thresholds (layer_two_batch_thres and layer_two_count_threshold) then this batch is assumed to be a drifted batch, and its index is appended to all_excede_list. Algorithm 4 returns the following set of outputs for each batch back to Algorithm 5 where it is stored in batch history.

```
1. all_excede_list                          // Updated List of batch indices exceeding layer 2 thresholds
2. error_list_layer1                // Recons. Error values for each instance of a batch from layer 1 autoencoder
3. exceed_count_layer1                      // Number of instances exceeding layer 1 count threshold
4. exceed_count_layer2                      // Number of instances exceeding layer 2 count threshold
5. len(layer_two_instance_exceed_list)      // Number of instances exceeding layer2 thresholds
6. layer_one_instance_exceed_list           // Holds the indices of instances exceeding layer 1 instance threshold
7. layer_two_instance_exceed_list           //Holds the indices of instances exceeding layer 2 instance thresholds
8. avg_error_layer1                         // Average batch reconstruction error from layer 1 AE for this batch
9. avg_error_layer2                         // Average batch reconstruction error from layer 2 AE for this batch
```

Input : test_batch, b, layer_one_batch_thres ,layer_two_batch_thres, layer_one_encoder, layer_two_encoder, layer_one_count_threshold, layer_two_count_threshold ,layer1_ins_thresh, layer2_ins_thresh, all_excede_list (initially empty)

Output: all_excede_list, error_list_layer1 , exceed_count_layer1, exceed_count_layer2, len(layer_two_instance_exceed_list) layer_one_instance_exceed_list , layer_two_instance_exceed_list, avg_error_layer1 ,avg_error_layer2

Process:

1. Initialize counters and variables
2. **For** i=1 to N (for each instance in a batch) **do**
   - 2.1 Predict and compute **recons_error** using **layer_one_encoder**
   - 2.2 Append this **recons_error** to **error_list_layer1**
   - 2.3 **If recons_error > layer1_ins_thresh, then**
     - 2.3.1 Increment **exceeds_count_layer1**
     - 2.3.2 Append instance index (**i**) to **layer_one_instance_exceed_list**

     end if
   - 3.4 **error_sum** += **recons_error**
3. Compute **avg_error_layer1** for this batch
4. Set **error_layer2=0**
5. **if** ((**avg_error_layer1**>**layer_one_batch_thres**) and (**exceed_count_layer1** >**layer_one_count_threshold**)) **Then**
   - 5.1 Append this batch id (**b**) to **layer1_excede_list**
   - 5.2 Set **exceed_count_layer2**=0
   - 5.3 **For** i=1 to N (for each instance in this batch b) **do**
     - 5.3.1 Predict and compute **recons_error** using **layer_two_encoder**
     - 5.3.2 Append this **recons_error** to **error_list_layer2**
     - 5.3.3 **If recons_error > layer2_ins_thresh then**
       - 5.3.3.1 Increment **exceed_count_layer2**
       - 5.3.3.2 Append instance index (**i**) to **layer_two_instance_exceed_list**

       end if
     - 5.3.4 **error_layer2+=recons_error**
   - 5.4 Compute **avg_error_layer2** for this batch **b**
   - 5.5 if (**avg_error_layer2** > **layer_two_batch_thres**) and (**exceed_count_layer2**>**layer_two_count_threshold**) then
     - 5.5.1 Append index **b** of this batch to **all_excede_list**
6. Return **output**

Algorithm 4. *perdict_for_each_batch*

Input : Batches ,layer_one_encoder ,layer_two_encoder, layer_one_batch_thres ,layer_two_batch_thres ,layer_one_count_threshold layer_two_count_threshold, layer1_ins_thresh, layer2_ins_thresh

Output: Display the output of the drift detector, warning levels and drift

Process

1. Initialize counters and variables
2. **for** all batches in data stream **do**
   - 2.1 Pass each batch to **Algorithm 4.0: predict_for_each_batch**
   - 2.2 Get the following output for each batch
     - 2.2.1 **all_excede_list** // Updated List of batch indices exceeding layer 2 thresholds
     - 2.2.2 **error_list_layer1** // Recon. error values for each instance of a batch from layer 1 AE
     - 2.2.3 **exceed_count_layer1** // Number of instances exceeding layer 1 count threshold
     - 2.2.4 **exceed_count_layer2**
     - 2.2.5 **avg_error_layer2** // Average batch recons. error from layer 2 AE for this batch
     - 2.2.6 **len(layer_two_instance_exceed_list** // Number of instances exceeding layer2 thresholds
     - 2.2.7 **layer_one_instance_exceed_list** // Holds the indices of instances exceeding layer 1 instance threshold
     - 2.2.8 **layer_two_instance_exceed_list** //Holds the indices of instances exceeding layer 2 instance thresholds
     - 2.2.9 **avg_error_layer1**
   - 2.3 Display index of instances exceeding layer 1 instance threshold
   - 2.4 Display index of instances exceeding layer 2 instance threshold
   - 2.5 Display number/ count of instances exceeding layer 2 instance threshold
3. **Detect drift at batch level** // Use Algorithm 6.0
   - 3.1 **avg_error_list**= average recons. loss values from **avg_error_layer2** // get mse values from python dictionary
   - 3.2 **exceed_list=** exceed count for each batch from **exceed_count_layer2** // get a list of counts from python dictionary
   - 3.3 detect_drift (avg_error_list, exceed_list, layer_two_batch_thres, layer_two_count_threshold)
4. **End**

Algorithm 5. *predict_detect_for_each_batch*

After all available batches have been processed, Algorithm 5 calls Algorithm 6 to detect drift. Algorithm 6 takes batch average reconstruction error (batch_avg_eror), exceed count (exceed_list) batch threshold (thresh), and count threshold (count_thresh) of layer 2 autoencoder. It compares batch average reconstruction error and exceeds count (from layer 2 autoencoder) for each batch from batch history with batch threshold and count threshold. If three consecutive batches exceed both thresholds, then drift is confirmed and if a single batch exceeds these thresholds a warning is generated

Input : batch_avg_eror, exceed_list, thresh, count_thresh
Output: Detected warning level if any, detected drift point if any
Process:

```
1.  Initialize counters and variables
    1.1   count=0              // counts the number of consecutive batches exceeding thresh_zscore
    1.2   w_index_list= []     //contains index number of batches where thresholds exceed
    1.3   w_count=0            //count of elements in w_index_list
    1.4   drift_batch=None     // Batch number where drift occurs, to be found
    1.5   w_level=None         // Warning level
    1.6   n=0                  // batch count exceeding threshold

2.  For i=1 to N (number of batches in data stream) do
    2.1   if(((batch_avg_error[i])>thresh) and (exceed_list[i]>count_thresh)):
    2.2           n=n+1
    2.2           if (it is the first batch, or the previous batch index is present in w_index_list) then
    2.3                   append this batch index (i) in w_index_list
    2.4                   increment count
    2.5                   Display the elements of w_index_list
    2.6           end if
    2.7           if it is first or second consecutive batch exceeding thresholds then generate warning end if
    2.8   if count > 2 then   // if three consecutive batches exceed thresholds
    2.9                   drift_batch=i-2
    2.10                  Print: Drift confirmed at batch number drift_batch
    2.11          end if
    2.12  end if
    2.13  if batch_mse [i] < thresh then reset count and w_index_list
```

Algorithm 6. *Detect Drift using Batch and Count Thresholds*

4. Experimental Evaluation

This section provides the experimental evaluation of the proposed AE-DDM method for drift detection. We have tested our approach on three datasets namely RBM datasets (Jaworski, Rutkowski, Angelov, et al., 2020) (Jaworski et al., 2018), a synthetic gaussian dataset, and a real-world weather dataset (A. Liu et al., 2018) (André G. Maletzke et al., 2018) (Ditzler & Polikar, 2013b). Section 4.1 presents the details of the experiment on RBM datasets, Section 4.2 covers details of experiments on synthetic gaussian datasets while experiment 3 on weather dataset is discussed in Section 4.3

4.1. RBM Dataset with Sudden & Gradual Drift

This dataset has been used by (Jaworski et al., 2018) and (Jaworski, Rutkowski, Angelov, et al., 2020) for drift detection using RBM and autoencoders respectively in an unsupervised setting. As our proposed research is also based on autoencoders, the same dataset is used in our first experiment with the intent to use the same logistics to understand the use of autoencoders with thresholding mechanisms in drift detection domain in the simplest possible way. This dataset is an unlabeled dataset, and here we do not consider the classification scenario. The focus is to check if autoencoders can detect the change in data distribution by generating warnings and then confirming the drift. Two different data streams S1 and S2 are generated with static probability distributions which are then used to generate a data stream with sudden drift and another with gradual drift (Jaworski, Rutkowski, & Angelov, 2020). Data stream with sudden drift contains 800,000 records, an initial 500,000 are taken from S1 and the remaining 300,000 are taken from S2. The following settings have been used to train the autoencoder on the first 400,000 records.

Table 6. *Autoencoder Settings*

Type of Autoencoder	Vanilla Autoencoder
Number of Layers	3 (Input, Bottleneck, Output)
Input Dimension	20
Bottleneck Dimension	One-third of input Dimension
Activation	Sigmoid
Optimizer	Adam
Loss	MSE
Batch Size	32

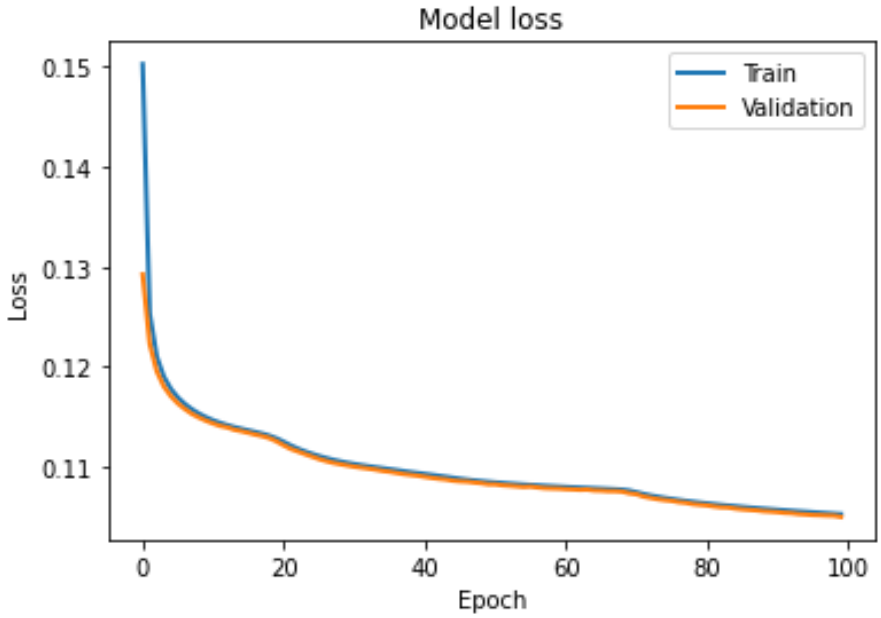


Figure 10:Training and Validation Loss

Out of 400,000 records, initial 320,000 records are used to train the autoencoder, next 40,000 for validation and the remaining 40,000 records (validation set II) are used for testing the trained autoencoder on the normal data stream and threshold computation. Training and validation loss (see figure above) over 100 epochs converge to below 0.11 approximately. We used validation set II, which contains 40,000 data points of non-drifted data for threshold computation. Figure 11 (a) shows the line plot of the reconstruction error from the trained autoencoder. As a single data point in a data stream carries limited information, we divided the validation set-II into batches of size 32 and computed the average batch reconstruction loss (see figure 13 (a)) showing the average signal constrained in the range 0.09 to 0.12 with an exception to some outliers while Figure 12 (a) shows the distribution plot of batch reconstruction error.

These batch average reconstruction error values (see figure 12a) pass the normality test. We computed batch threshold (**thresh**=0.1343) using Algorithm 3. We assume that a batch with drifted data will have average reconstruction error greater than this threshold value. Data stream in range [400,000:800,000] with 400,000 records which contains sudden drift is passed through the trained autoencoder and reconstruction error is computed for each data point. Figure 11(b) shows the line plot of reconstruction error. From this reconstruction error plot, drift can be observed starting from data point 100,000. This can be validated by dividing the whole data stream into batches of size 32. There are 12500 batches of size 32 in this batch stream where sudden drift starts from batch 3125 which can be seen clearly in the batch reconstruction error plot given in Figure 13(b).

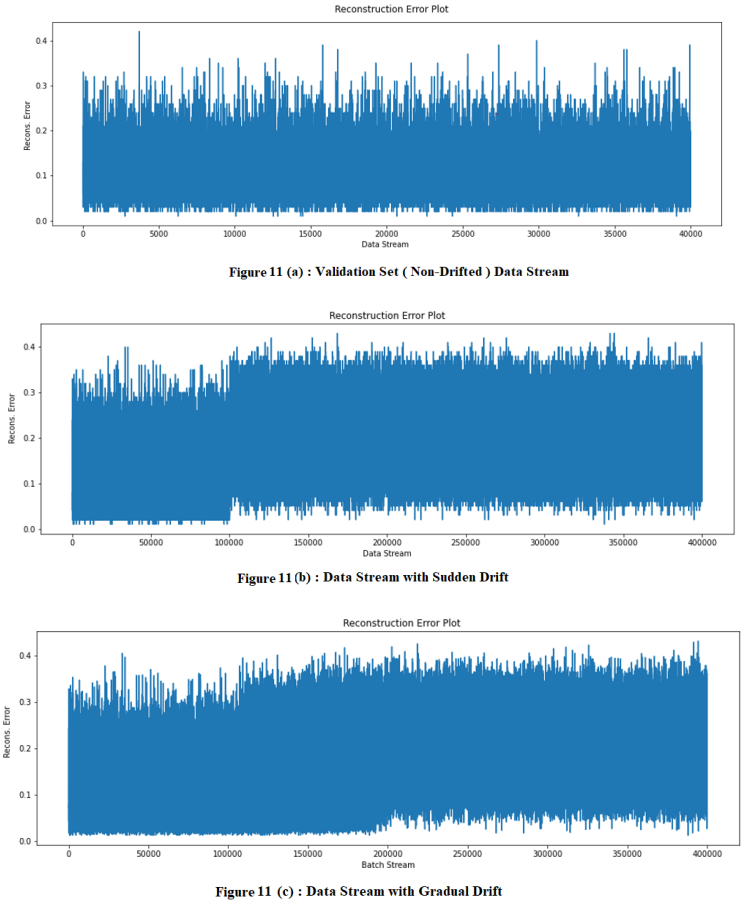


Figure 11 (a) : Validation Set (Non-Drifted) Data Stream

Figure 11 (b) : Data Stream with Sudden Drift

Figure 11 (c) : Data Stream with Gradual Drift

Figure 11. *Reconstruction Error Plots for Validation Set, Data Stream with Sudden Drift and Data Stream with Gradual Drift*

A comparison of normal (non-drifted) batch stream (validation set -II) and batch stream with sudden drift can also be made with the help of distribution plots given in Figure 12(a) and 12(b). In batch stream with sudden drift, the reconstruction error distribution is bimodal (Figure 12b). The first peak corresponds to the reconstruction error for the first 3125 batches while the second peak or distribution represents the drifted part of the batch stream which starts from batch 3125.

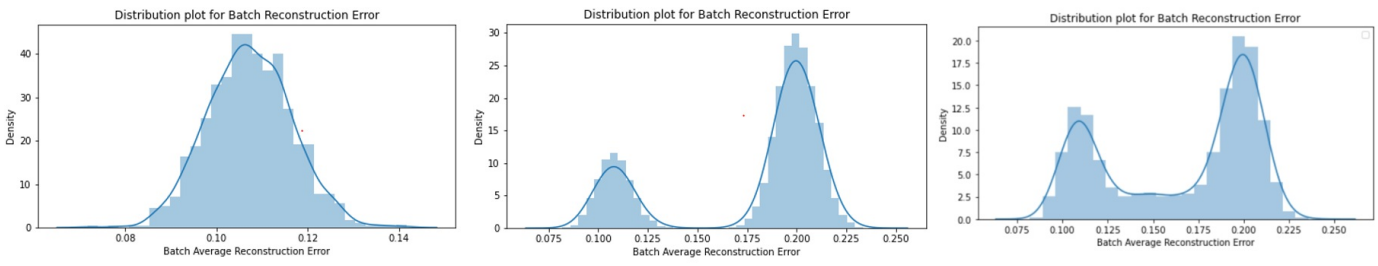


Figure 12 (a) : Validation Set (Non-Drifted) Batch Stream

Figure 12 (b) : Batch Stream with Sudden Drift

Figure 12 (c) : Batch Stream with Gradual Drift

Figure 12. *Batch Reconstruction Error Distribution Plots for Validation Set, Sudden Drift and Gradual Drift Batch Streams*

Rather than only relying on these plots, we used a drift detection algorithm (see Algorithm 7) given below to explicitly generate warnings and confirm drift. A warning is generated if a batch exceeds threshold, or two consecutive batches exceed threshold and drift is confirmed if three consecutive batches exceed this threshold. It is observed that although drift is present at batch number 3125 and onwards, Algorithm 7 generates warning at batch numbers 39, 600, 1606, 1813, 2210, and 2971 which are considered as false positives or false alarms as the drift is not actually present and the data in these batches belongs to the same non-drifted distribution.

Input

- batch_average _error
- batch_thresh

Output

- Detected Warning Level if any
- Detected Drift Point if any

Process Batch Stream

- Compare each batch reconstruction error with **batch_thresh**
    a. **If** a batch exceeds **batch_thresh** or two consecutive batches exceed **batch_thresh, then**
        i. Generate a drift warning
    b. **If** three consecutive batches exceed **batch_thresh, then**
        i. Confirm the drift and drift point

Algorithm 7. Detect Drift using Batch Threshold

To combat this problem, we introduced another threshold called count threshold described in Section 3.1.1 (b) and used both these thresholds namely batch threshold and count threshold in conjunction (see Algorithm 6, Section 3.1.3) which helps in eliminating any false positives and makes the drift detector more robust to noise or outliers. Using both thresholds, drift detector not only considers the average batch reconstruction error as an extrinsic measure but also considering how many instances in that batch exceed instance threshold, thus also incorporating intrinsic measures. Drift detector in Algorithm 6 generates warnings at batch 3125 and 3126 and then conforms the drift at batch 3124 as thresholds exceed at three consecutive batches 3125, 3126 and 3127. There is a zero delay in this drift detection.

In the second part of this experiment, drift detectors given in Algorithm 6 (See Section 3.1.3) and Algorithm 7 are tested on data with gradual drift. In data stream with gradual drift, the initial 500,000 records are taken from S1, the next 100,000 records with a (600000-i) / 100000 probability from S1 and with a (i-500000) /100000 probability from S2 and last 200,000 records are taken from S2 (Jaworski et al., 2018). Data starts drifting gradually from data point 500,000 till 600,000 till it attains a totally different distribution as compared to S1. We passed the data stream in range [400,000:800,000] to the autoencoder trained on normal non-drifted data (validation set II). The reconstruction error plot (see Figure 11 c) shows that error starts gradually increasing from data points in the range [500,000-600,000] or from 100,000 to 200,000 in other words if we consider this data stream independently.

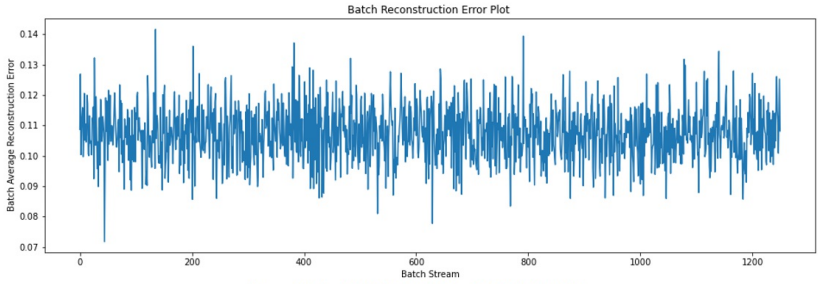


Figure 13 (a) : Validation Set (Non-Drifted) Batch Stream

Figure 13 (b) : Batch Stream with Sudden Drift

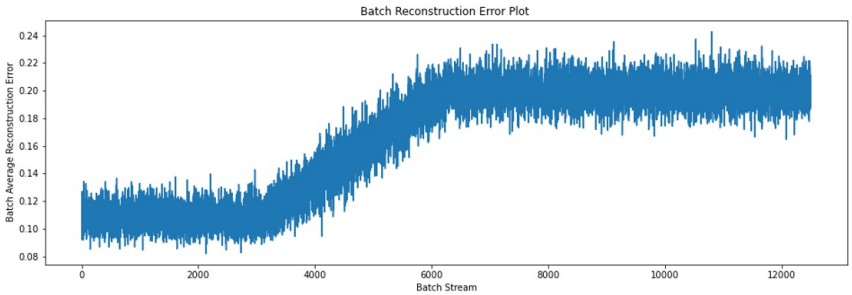


Figure 13 (c) : Batch Stream with Gradual Drift

Figure 13. *Batch Reconstruction Error Plots for Validation Set-II, Batch Stream with Sudden and Gradual Drift*

Again, this data stream is divided into batches of size 32, and batch average reconstruction error is computed for each batch which is shown in figure 13(c) clearly showing a gradual change in batch average reconstruction. This can be further validated by the distribution plot given in figure 12 (c) where the distribution of average batch reconstruction for batches in the data stream range [0:100,000] corresponds to first peak, the flattened region between two peaks corresponds to batches in the range [100,000:200,000] where gradual drift occurs, and the second peak corresponds to batches corresponding to data in the range [200,000:400,000] where its attains a totally different distribution.

In the case of drift detector given in Algorithm 7 which is using only one threshold namely batch threshold drift is confirmed at batch number 3800 although it started occurring at batch number 3125. Drift detector started generating warnings more frequently as drift started to occur at batch 3125. Using Algorithm 6, drift is confirmed at batch 4017 with a smaller number of warnings as compared to Algorithm 7 which shows that count threshold is a bit stricter measure as compared to the batch threshold. We also experimented with Algorithm 6 with a small change i.e., by changing **AND** with **OR** in line 2.1, and almost got the same results further validating that count threshold is a stricter measure as compared to the batch threshold. This delay in gradual drift detection is also in accordance with a general

understanding among the research community that gradual drift and difficult to detect as compared to sudden drift. Results of this experiment are available at https://github.com/Usman07442/AE-DDM-Experiments

4.2. Synthetic Gaussian Dataset

Since our research focus is drift detection in classification problems in data streams, motivated by the results of the drift detector in Experiment 1, we tested this algorithm 8 on two most used synthetic labelled datasets namely hyperplane and stagger. Hyperplane dataset has been frequently used in research papers related to drift detection (Haixun Wang et al., 2003b), (Fan, 2004), (Hulten et al., 2001), (Haque, Khan, & Baron, 2016), (Sethi & Kantardzic, 2017). A hyperplane with d dimensions is represented by the equation $_i x_i = w_0$, If $w_i x_i >= w_0$ then instances are labeled as positive otherwise negative. (Hulten et al., 2001). A change in classification boundary is introduced by changing the weights of the features gradually. We used skmultiflow to generate a hyperplane dataset both normal and drifted. A normal dataset is generated with 10 features with each feature having values from a uniform distribution in the range [0,1]. Using the same architectural settings as in Experiment 1, an autoencoder is trained on normal non-drifted data and a drifted batch stream is passed to the drift detector for drift detection.

Since the way gradual drift is introduced by gradually changing the feature weights, there is no change in the overall distribution of the drifted data as compared to the data on which an autoencoder has been trained. It is only the distribution of the classes which is being changed gradually. Considering this, we trained two separate autoencoders one on the positive class data and the other one on the negative class data using the same autoencoders settings as in Table 6 (wherever possible). We used Algorithm 6 within the proposed AE-DDM framework for drift detection. Since the introduced gradual drift is very slow, this gradual drift goes undetected.

We tested AE-DDM on the stagger dataset as well which is used in research papers (Schlimmer & Granger, 1986) (Jeremy Z. Kolter & Maloof, 2003) (Nishida & Yamauchi, 2007) related to supervised drift detection techniques. Stagger consists of three attributes namely color, shape and size where color $\in$ {green, blue, red}, shape $\in$ {triangle, circle, rectangle}, and size $\in$ {small, medium, large}. (Jeremy Z. Kolter & Maloof, 2003) has used three different concepts over a stream of 120 data points. These concepts are like class is positive if color='red' and size='small' for the first 40 data points, class is positive if shape='green' or shape='circle' for the next 40 data points, and class is positive if size=medium or large for the last 40 data points. A classifier trained on any one of the concepts is supposed to do mistakes when the concept changes and the classifier's performance is monitored to signal the drift in performance (Nishida & Yamauchi, 2007). We generated 3000 data points on concept1 where the class is positive if color='red' and size='small'. This dataset is divided into 80% train and 20% validation datasets.

An autoencoder is trained on positive instances and another one on negative instances of the training set. Validation data is divided into positive and negative instances and passed to its respective trained autoencoder in batches of size=32 for prediction. Based on AE prediction instance threshold, count threshold, and batch thresholds are computed for both autoencoders and summarized in the table given below:

Table 7. *Threshold Values for Stagger Dataset*

Autoencoder	Batch Threshold	Instance Threshold	Count Threshold
Positive AE	0.1526	0.0950	0
Negative AE	0.1296	0.2339	0

A normal data stream of 1920 data points divided in 60 batches of size 32 on the same concept (as used to trained both autoencoders) containing both positive and negative samples is passed to the AE-DDM. No drift is detected and outputs of the drift detector like average mse values and batch wise instance threshold exceed counts for both layer 1 and layer 2 autoencoders are stored.

A drifted dataset is generated with 1920 data points (60 batches of size=32) by changing the definition of the positive class as color='green' or shape='circle'. Drifted data contains both positive and negative samples. This drifted data is also passed through AE-DDM, but no drift or warning is detected. Outputs of the drift detector like batch average reconstruction error and batch exceed counts for both layer 1 and layer 2 autoencoders on drifted data are also recorded. We used two sample t-test to check if the means of average batch reconstruction error and batch wise exceed counts for layer 1 and layer 2 autoencoders for both normal and drifted batch stream are significantly different. Our null hypothesis is that means are same i.e., there is no drift in the data stream and our alternative hypothesis is that means are different or there is a drift in the data stream.

Table 8. *Student's t-test Results for Stagger Dataset*

Normal and Drifted Outputs	t-test Result at alpha=5% (p-values)
Average MSE from Layer 1 AE	0.99 (Accept Ho)
Exceed counts Layer 1 count threshold	0.78 (Accept Ho)
Average MSE from Layer 2 AE	1.0 (Accept Ho)
Exceed counts Layer 2 count threshold	0.46 (Accept Ho)

From these two experiments on Hyperplane and Stagger datasets, we concluded that datasets like Hyperplane, Stagger and Sine1 etc. which have been mainly used in supervised drift detection techniques, the definition of the class or class boundary is explicitly changed which has no impact on the distribution of the data as the data is generated by the same process. These datasets where the class boundary is explicitly changed can only be used in supervised drift detection techniques where the drift detector works on the distribution of the window which stores the performance measures like

accuracy, recall, error rate or TPR, TNR, FPR and FNR etc. In our proposed method which monitors the changes in the data distribution by monitoring the reconstruction error values from autoencoders, these synthetic datasets cannot be used for evaluating our proposed drift detection technique. To overcome this issue of not being able to use the majority of the datasets used by researchers in supervised drift detection techniques, we synthesized our own dataset named as GAUSSIAN dataset.

We generated a synthetic dataset with 10 features. Each feature value is drawn from a standard normal distribution based on a specified range of mean and standard deviation. Since we needed a supervised / labelled dataset, we generated positive samples and negative samples from two significantly distant normal distributions. For each feature in negative samples, we specified the mean range as (0.1,0.6) and the standard deviation range as (0.05,0.45) and generated 30,000 samples. Whereas for each feature in positive samples, we specified the mean range as (2,7) and standard deviation range as (1.5,2.5) and generated 30,000 samples. Both datasets are divided into 70% training and 20% validation set and 10% (validation set -II). Two autoencoders are trained; one on positive class data and the other one on negative class data respectively. Validation set -II data of negative class as well as the positive class is divided into batches of size 32 and passed to the corresponding autoencoder. Based on reconstruction error values, three different thresholds namely ***instance threshold***, ***batch threshold,*** and ***count threshold*** are computed for both autoencoders.

A normal data stream containing 3000 data points from negative class and 3000 data points from positive data points are combined and divided into batches of size 32 and passed through the drift detector. No warning level or drift point is detected by the drift detector. Outputs of the drift detector like batch reconstruction error values and exceed counts for both layer 1 and layer 2 autoencoders are recorded.

Table 9. *Threshold Values of Positive and Negative AE on Gaussian Dataset*

Autoencoder	Batch Threshold	Instance Threshold	Count Threshold
Positive AE	20.15	32.5	1
Negative AE	0.04	0.09	4

Two drifted datasets are generated by introducing a drift in the distributions of both positive and negative samples. Positive drifted data is drawn from a gaussian distribution with mean **(4,9)** and standard deviation range as **(1.5,3.0)** whereas negative drifted data is drawn from a gaussian distribution with mean**(0.3,0.9)** and standard deviation range as **(0.1,0.5).** We generated 3000 samples of both positive and negative drifted data and shuffled the data points to generate a drifted dataset of 6000 data points. This drifted dataset is preceded by 640 datapoints (20 batches of size 32) from the normal data stream containing both positive and negative samples. This drift stream is divided into batches of size 32 (where sudden drift is present starting at batch 20) and passed through the AE-DDM. Based on Flowchart 1, negative autoencoder (autoencoder trained on negative class data) is placed at layer 1 while positive

autoencoder (autoencoder trained on positive class data) is placed at layer 2. Our proposed AE-DDM generates warning level at batch 20 and 21 and confirms this drift at batch 28. The results of the drift detector are available on GitHub.

A detailed comparison of average reconstruction error both from layer 1 and layer 2 autoencoders on normal batch stream as well as drifted batch stream is shown in Figure 14. For normal batch stream, the average reconstruction error from layer 1 autoencoder is 10.5 approximately (Figure 14a) which changes to 23.5 approximately for drifting batches in case of sudden drift (Figure 14b). As the drift start from batch 20, the average reconstruction error pattern for first 20 batches is like normal data stream but changes suddenly from batch 20 as sudden drift occurs. Similarly, the average reconstruction error from layer 2 autoencoder for normal batch stream is approximately 9.0 which changes to 21.0 approximately for the drifting batches.

Figure 14 (a) : Batch Average Reconstruction Error from Layer 1 AE on Normal Data Stream

Figure 14 (b) : Batch Average Reconstruction Error from Layer 1 AE on Data Stream with Sudden Drift starting at Batch 2

Figure 14(c) : Batch Average Reconstruction Error from Layer 2 AE on Normal Data Stream

Figure 14 (d) : Batch Average Reconstruction Error from Layer 2 AE on Data Stream with Sudden Drift starting at Batch 20

Figure 14. *Distribution Plots of Batch Reconstruction Error from Layer 1 and Layer 2 Autoencoders*

Figure 15 shows the distribution plots of batch reconstruction error from layer 1 as well as layer 2 autoencoders both for normal as well as drifted data. In the case of drifted data, there is a clear change in distribution as compared to normal non-drifted data.

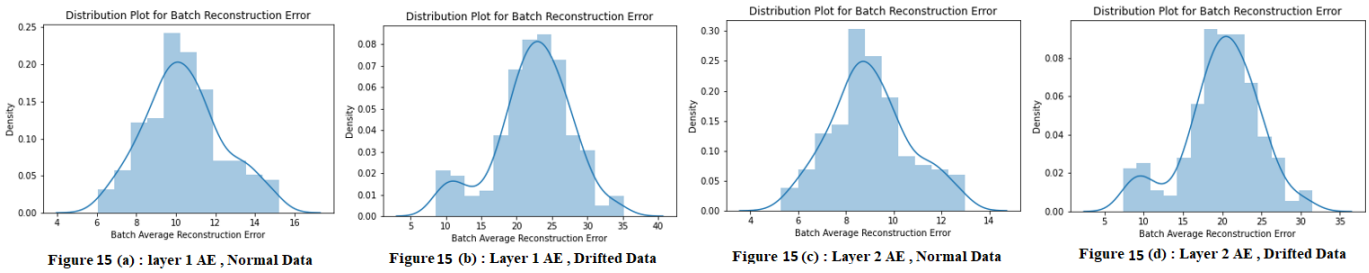


Figure 15 (a) : layer 1 AE , Normal Data

Figure 15 (b) : Layer 1 AE , Drifted Data

Figure 15 (c) : Layer 2 AE , Normal Data

Figure 15 (d) : Layer 2 AE , Drifted Data

Figure 15. *Distribution Plots for Batch Reconstruction Error from Layer 1&2 AEs both for Normal as well as Drifted Data*

Apart from comparing change patterns in batch average reconstruction error, a comparison of exceed counts based on instance threshold both for normal as well as drifted data also confirms this sudden drift (see Figure 18). As the distribution of the class in the dataset is not imbalanced, approximately half of the data points in a batch of 32 belong to the positive class and half of the data points belong to the negative class. As the autoencoder with a low batch threshold is placed at layer 1 in AE-DDM framework, in each batch of normal non-drifted data, approximately half of the data points (mostly 15, 16, or 17) exceed the instance threshold of layer 1 autoencoder (see Figure 18a).

This is in accordance with the fact that data points that exceed layer 1 instance threshold belong to the other class (positive class in this case) whose respective trained autoencoder is placed at layer 2. When this normal batch stream passes layer 2 autoencoder, for most of the batches, there is no data point that exceeds layer 2 instance threshold and a single data point in some cases (see Figure 18c). In the case of drifted data, this pattern changes both at layer 1 as well as at layer 2. As the drift starts from batch 32, the same pattern of exceed counts persists for the first 32 batches for both layer 1 and layer 2 exceed counts which changes suddenly as the drift occurs. Layer 1 exceed counts jumps to on an

average of 30 data points which exceeds layer instance threshold (see figure 18 b) and layer 2 exceed counts jump to on an average of 15 data points (see figure 18d).

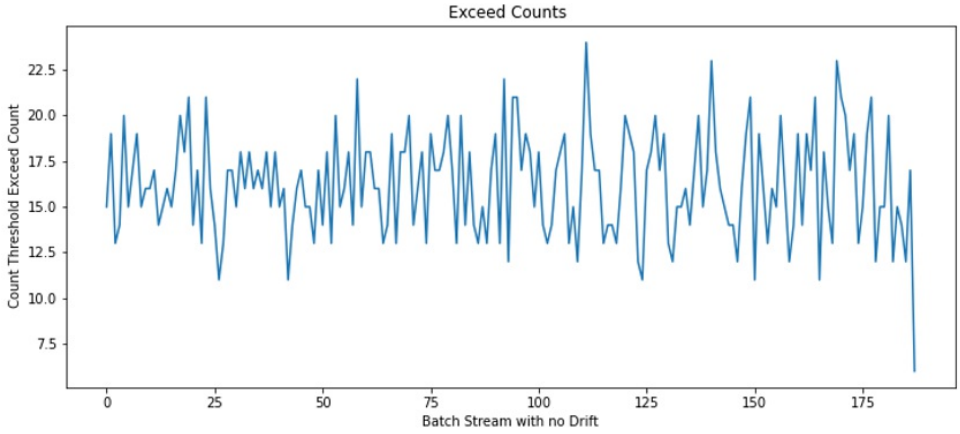


Figure 16 (a) : Layer 1 Exceed Counts for Normal Non-Drifted Data

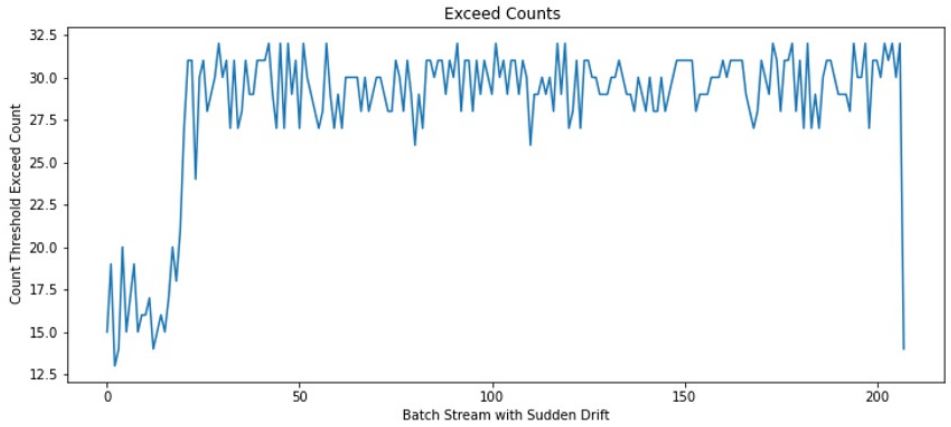


Figure 16 (b) : Layer 1 Exceed Counts for Batch Stream with Sudden Drift

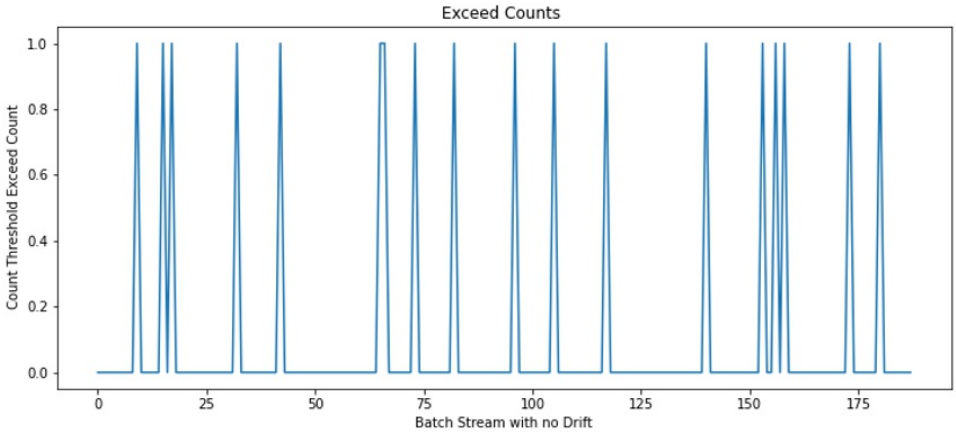


Figure 16 (c): Layer 2 Exceed Counts for Normal Non-Drifted Data

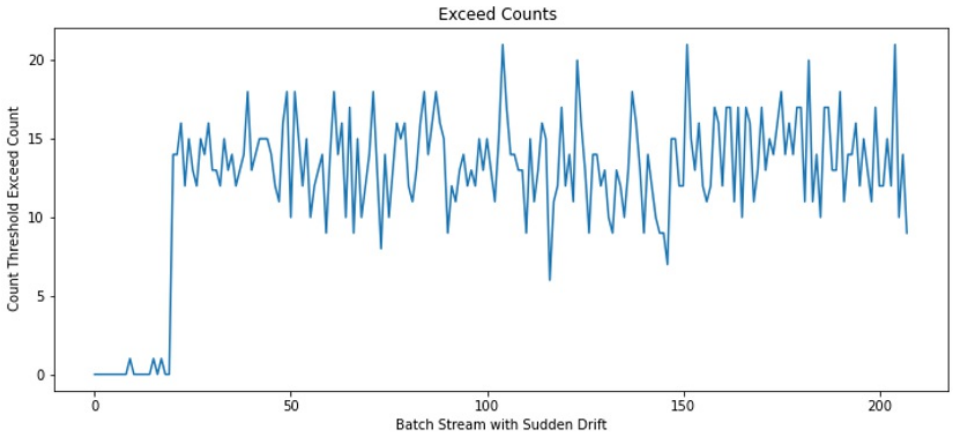


Figure 16 (d): Layer 2 Exceed Counts for Batch Stream with Sudden Drift

Figure 16. *Comparison of Instance Threshold Exceed Counts for both Layer 1&2 Autoencoders for Normal and Drifted Data*

These change patterns can be further validated with the help of distribution plots given in figure 17 below. The distribution of exceed counts changes at both layers when sudden drift occurs.

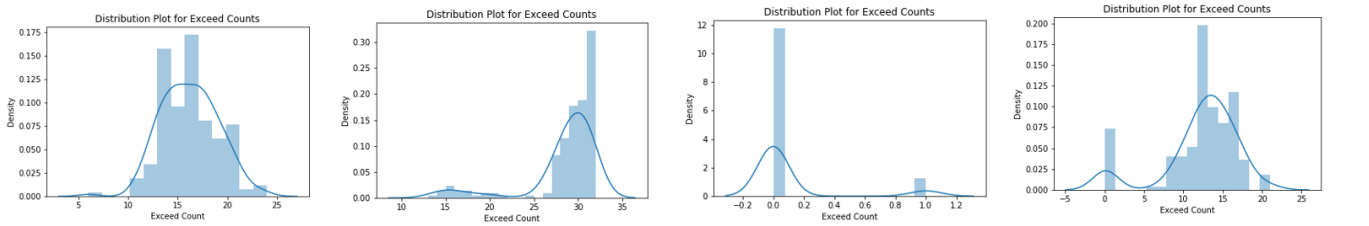


Figure 17(a): Layer 1 Exceed Counts , Normal Data **Figure 17(b): Layer 1 Exceed Counts , Difted Data** **Figure 17(c): Layer 2 Exceed Counts, Normal Data** **Figure 17(d) : Layer 2 Exceed Counts , Drifted Data**

Figure 17. *Distribution Plots of Exceed Counts from Layer 1&2 AEs both for Normal as well as Drifted Data*

We also performed a two samples t-test to compare the average reconstruction error values and instance threshold exceeds counts for both normal and drifted data at bother layer 1 and layer 2. Test results at 5% significance level show that there is a drift in the data.

Table 10. Student t-Test on Layer 1 and Layer 2 Outputs for Gaussian Data

Normal vs Drifted	t-test Result at alpha=5% (P-Values)
Layer 1 Batch Average Recon. Error Values	0.00 (There is a drift in data)
Layer 1 Instance Threshold Exceed Counts	0.00
Layer 2 Batch Average Recon. Error Values	0.00
Layer 2 Instance Threshold Exceed Counts	0.00

4.2.1 Effect of Drift on Classification

This part of the experiment is focused on evaluating the impact of the drift introduced in Gaussian dataset on the classifier's performance. Positive and negative class data is generated as described earlier, concatenated, and shuffled to have 60,000 data points. First 50,000 data points are divided into 70:30 train test and a normal stream of 10,000 data points is used to test classifiers' performance when there is no drift in the dataset. We have used eight different classifiers including Gaussian Naïve Bayes, Logistic Regression, KNN (k=5), Decision Tree Classifier, SVM, Random Forest, XGBoost and MLP classifiers to test the impact of drift on classification. The average results over 10 runs are shown below:

Run	Test Accuracy								Accuracy on Normal Data								Accuracy on Drifted Data							
	NB	LG	DT	KNN	SVM	RF	XGBo	MLP	NB	LG	DT	KNN	SVM	RF	XGBo	MLP	NB	LG	DT	KNN	SVM	RF	XGBo	MLP
1	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	0.49	1.00	1.00	1.00	1.00	1.00	0.99	1.00	0.49	0.99	0.75	1.00	1.00	0.99	0.99	0.99
2	1.00	1.00	0.99	1.00	1.00	1.00	0.99	1.00	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.49	1.00	0.87	1.00	1.00	0.88	0.87	1.00
3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.49	1.00	0.91	1.00	1.00	0.98	0.98	0.99
4	1.00	1.00	0.99	1.00	1.00	1.00	0.99	1.00	0.50	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.50	0.99	0.99	0.99	0.99	1.00	0.99	0.99
5	1.00	1.00	0.99	1.00	1.00	1.00	0.99	0.99	0.64	0.99	0.99	1.00	1.00	1.00	0.99	0.99	0.50	1.00	0.51	1.00	1.00	0.93	0.52	1.00
6	1.00	0.99	0.99	1.00	1.00	1.00	0.99	0.99	0.50	0.99	0.99	1.00	1.00	1.00	0.99	0.99	0.50	0.99	0.99	0.99	0.99	1.00	0.99	0.99
7	1.00	1.00	0.99	1.00	1.00	1.00	0.99	1.00	0.50	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.49	0.99	0.93	0.99	0.99	1.00	0.93	0.99
8	1.00	1.00	0.99	1.00	1.00	1.00	0.99	0.99	0.49	0.99	0.96	1.00	1.00	0.97	0.96	0.99	0.49	0.98	0.93	0.99	0.99	1.00	0.93	0.99
9	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.49	1.00	0.97	1.00	1.00	1.00	0.99	0.99	0.50	1.00	0.65	1.00	1.00	0.99	0.65	1.00
10	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00	0.51	1.00	0.99	1.00	1.00	1.00	1.00	1.00	0.49	0.99	0.99	1.00	1.00	0.99	0.99	0.99
Average	1.00	1.00	0.99	1.00	1.00	1.00	0.99	1.00	0.56	1.00	0.99	1.00	1.00	1.00	0.99	0.99	0.49	0.99	0.85	1.00	1.00	0.98	0.88	0.99

Table 11. Classifiers Accuracy on Normal and Drifted Gaussian Data

Depending on the data generation process, data of both negative and positive class is so well separated that all the classifiers have nearly 100% test accuracy. Accuracy on normal data stream is 99% for DT, XGBoost and MLP classifier while 100% for Logistic Regression, KNN, RF and SVM. The average accuracy of Gaussian Naïve Bayes is 56 % due to the continuous nature of data. Classifiers' performance on drifted data shows that in the case of Naïve Bayes (49%), DT (85%) and XGBoost (88%), there is a considerable decrease in accuracy as compared to other classifiers in the pool. Since this is a synthetic dataset, originally both classes have very different distributions which help all the classifiers to learn the decision boundary very well, and the way drift has been introduced (decision boundary learned by classifiers between two classes is still relevant), there is not much impact on the classifier's performance. This led us to perform another experiment on a real-world dataset which is presented next.

4.3. Experiment 3: NOAA Weather Dataset with Sudden Drift

To evaluate our proposed drift detection technique on a real-world dataset, we have used a weather dataset (Ditzler & Polikar, 2013b) (A. Liu et al., 2018) (André G. Maletzke et al., 2018). This dataset contains weather measurements in

eight dimensions and contains daily records covering a period of 50 years. There are 18,159 records in this dataset and the task is to predict whether it will rain or not. The class distribution is 12,461 records for no rain and 5,698 for rain. Eight attributes are temperature, dew point, sea level pressure, visibility, average wind speed, maximum sustained wind speed, minimum temperature, and maximum temperature. The data has been made available by National Oceanic and Atmospheric Administration (NOAA) as is referred to as NOAA dataset.

In the case of real-world datasets, it is not known whether drift is present or not, and if it is present, the drift point is not known. The presence of real drift can only be confirmed if the classifier's performance degrades suddenly or gradually with time. Since supervised drift detection methods detect drift based on the classifier's performance and unsupervised drift detection methods detect drift without considering the impact on the classifiers performance, to make NOAA dataset relevant to our approach, we injected artificial drift to test whether our approach can detect this drift and whether this detected drift influences the classifier's performance or not.

NOAA dataset is class imbalanced with **rain: no-rain** being 31%:69% approximately. Initial 80% of data is used for training (10% of this data chunk is used for validation) autoencoders, the next 10 % (validation set -II) is used for threshold computation and the last 10% is used as a normal stream to test the drift detector.

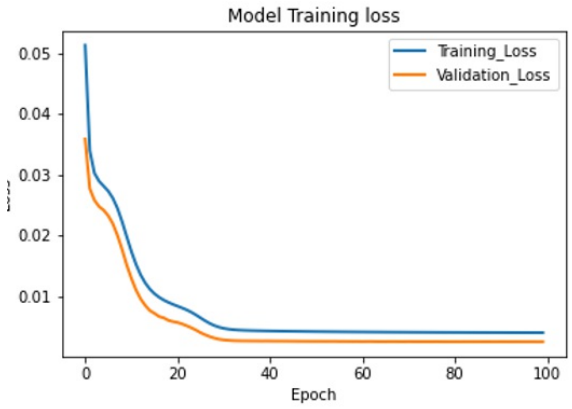


Figure 18 (a) : Autoencoder Training on Non-Rain Data

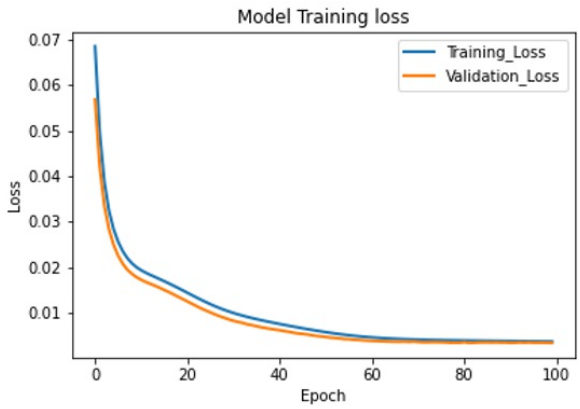


Figure 18(b): Autoencoder Training on Rain Data

Figure 18. *Training and Validation Loss on NOAA Data*

Normal 10 % data (Validation Set -II) is separated class-wise and is passed to respective AE in batches and all three thresholds namely instance threshold, batch threshold, and count thresholds are computed for both Layer 1 and Layer 2 autoencoders (given below):

Table 12. *Threshold Values of Positive and Negative AE on NOAA Dataset*

Autoencoder	Batch Threshold	Instance Threshold	Count Threshold
Positive AE	0.0064	0.0156	3
Negative AE	0.0073	0.0104	9

Normal 10% stream data is divided into batches of size 32 and passed to the drift detector. No warning level or drift point is detected by the drift detector. A sudden drift is added to attribute1 by adding an arbitrary value to the normalized values in normal 10 % stream data. Drift Detector generates warnings at batches 0, 1, and 2 and confirms the drift at batch 0 as three consecutive batches exceed thresholds. Student t-test also confirms the presence of drift in the batch stream.

Table 13. *Student t-Test on Layer 1 and Layer 2 Outputs for NOAA Data*

Normal vs Drifted	t-test Result (p values) at alpha=5%
Layer 1 Average Batch Recon. Error	0.00, There is a Drift
Layer 1 Instance Threshold Exceed Counts	0.00
Layer 2 Average Batch Recon. Error	0.00
Layer 2 Instance Threshold Exceed Counts	0.00

The presence of drift can be further validated by monitoring the distribution of batch reconstruction error and exceed counts at both layers of AE-DDM. The distribution of batch reconstruction error at layer 1 of AE-DDM is centered around 0.004 approximately in the case of normal non-drifted batch stream (see Figure 19a) which changes to 0.012 approximately in the case of drifted batch stream (Figure 19b). Similarly, the distribution of batch average reconstruction error at layer 2 changes from a unimodal distribution (Figure 19c) to a bimodal distribution (Figure 19d).

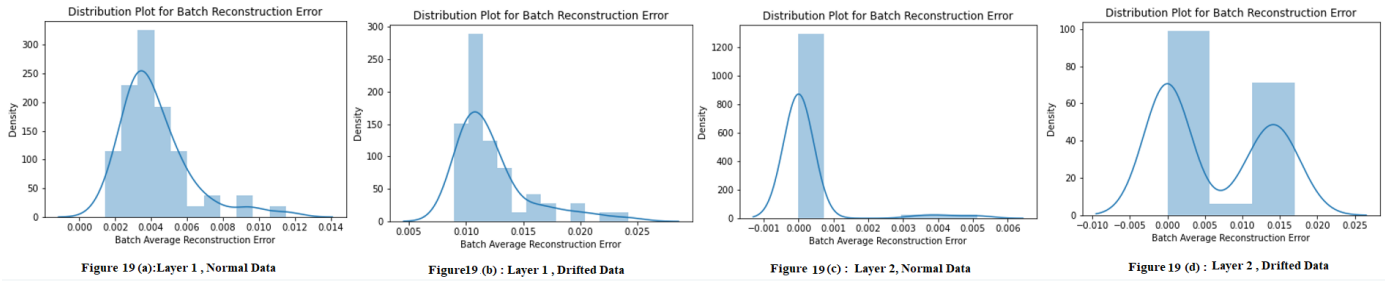


Figure 19. *Distribution Plots for Batch Reconstruction Error from Layer 1&2 AEs both for Normal as well as Drifted NOAA Data*

The changes in the distribution of exceed counts at both layers of AE-DDM also validate this drift. In the case of normal

data (1816 records, 57 batches of size 32), the distribution of exceeding counts at layer 1 is centered around 0, meaning that in most of the cases, no data point exceeds the layer 1 instance threshold, and in some cases, only one data point exceeds layer 1 instance threshold and in very rare cases number of data points exceeding instance threshold is between 2 to 8 (See Figure 20a). This pattern changes in the case of drifted batch stream where the number of data points exceeding layer 1 instance threshold range between 0 and 25 (Figure 20b). Similarly, a week bimodal distribution of layer 2 instance threshold exceeds counts at layer 2 for normal data (Figure 20c) changes to a clear bimodal distribution in case of drifted batch stream (Figure 20d).

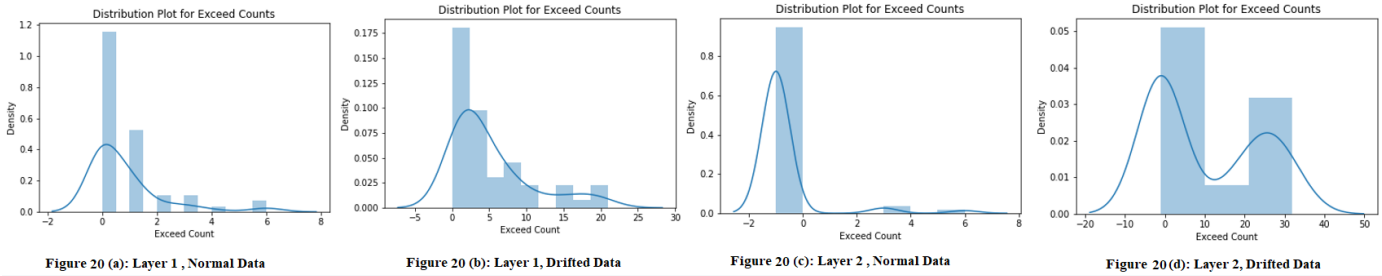


Figure 20. *Distribution Plots of Exceed Counts from Layer 1&2 AEs both for Normal as well as Drifted NOAA Data*

To check whether this drift is real or virtual, we trained 8 different classifiers namely Naïve Bayes, Logistic Regression, Decision Tree, K Nearest Neighbors, Support Vector Machine, Random Forest, XGBoost, and Multilayer Perceptron on NOAA dataset. Accuracy, TP, TN, FP, and FN on test data, normal data, and drifted data are given in Table 12. There is a considerable decrease in accuracy on drifted data as compared to test data and normal data stream. Absolute and percentage decreases in accuracy in Table 12 are computed as compared to test data. A considerable decrease in accuracy indicates that the drift present in the dataset is real as it impacts the classifier's performance.

Classifier	Test Data					Normal Data					Drifted Data					Decrease in Accuracy	
	TN	FP	FN	TP	Accuracy	TN	FP	FN	TP	Accuracy	TN	FP	FN	TP	Accuracy	Absolute	Percentage
NB	944	243	299	330	0.701	657	508	179	472	0.621	826	339	351	300	0.620	0.081	11.55%
LR	1086	101	317	312	0.769	870	295	225	426	0.713	166	999	6	645	0.446	0.323	42.00%
DT	1100	87	333	296	0.768	810	355	224	427	0.681	810	355	224	427	0.681	0.087	11.33%
KNN	1062	125	289	340	0.772	893	272	215	436	0.731	664	501	169	482	0.631	0.141	18.26%
SVM	1084	103	287	342	0.785	877	288	216	435	0.722	2	1163	0	651	0.359	0.426	54.27%
RF	1099	88	272	357	0.801	991	224	213	438	0.759	744	421	114	537	0.705	0.096	11.99%
XGB	1113	74	293	336	0.797	927	238	204	447	0.756	772	393	132	519	0.710	0.087	10.92%
MLP	1128	59	453	176	0.718	834	331	222	429	0.695	812	353	214	437	0.687	0.031	4.32%

Table 14. *Classifiers Performance on Test, Normal and Drifted Weather Data*

4.5. Discussion on Experimental Results

The proposed AE-DDM method for drift detection has been evaluated on four synthetic data sets (RBM, Hyperplane,

Stagger, and Gaussian) and one real-world NOAA dataset. AE-DDM detects drift based on algorithm 6 which uses two different thresholds (batch and count threshold) computed from reconstruction error values both from layer 1 and layer 2 autoencoders. The presence of drift is evaluated with the help of reconstruction error plots and distribution plots of reconstruction error and exceed count values. To check the significance of the differences in reconstruction error values and exceed counts in the case of normal and drifted batch streams and validate the results of AE-DDM, Student ‘s t test is also used. In the case of labelled datasets (Gaussian and NOAA), the impact of the drift is also evaluated by comparing the classification performance of 10 well-known supervised machine learning algorithms on drifted data as compared to non-drifted data.

Experiments on RBM (Sudden and Gradual Drift) which is an unlabeled dataset helped to lay the foundations for algorithm 7 which detects drift based on batch threshold only. Although, using batch threshold sudden drift can be detected with zero delay, but the generation of false warnings leads us to define another threshold called count threshold. When both thresholds are used in conjunction with algorithm 6, then the problem of this false warning/ positives is solved. Since RBM dataset is an unlabeled dataset, a single autoencoder has been trained on the entire data without considering the presence of different classes. Since our focus is on drift detection in a supervised machine learning scenario, we proposed a layered autoencoder-based drift detection method (AE-DDM) where two different autoencoders are trained; one on positive class data and the other on the negative class data. Autoencoder with low batch threshold is placed at layer 1 and the other at laye2 and drift detection is done based on Algorithm6 which uses both batch and count thresholds to generate warnings and then confirm the drift. Experiments on Hyperplane and Stagger datasets, where the decision boundaries and explicitly changed to induce drift and there is no change in the distribution of the data, proved to be not suitable to be used in drift detection methods that monitor the changes in the data distribution or the changes in the distribution of reconstruction loss like AE-DDM. This led us to make our own synthetic gaussian dataset.

Experimental results on the gaussian dataset which is a synthetic binary classification dataset with sudden drift show that AE-DDM can generate warnings and detect drift with zero delay. Since it is a synthetic dataset, the presence of two classes is not on the logical basis and is just based on a change in the data distribution, there is not much change in the classification performance of chosen machine learning algorithms in drifted data as compared to normal data. Experiments on NOAA datasets, which is a real-world dataset with sudden drift show that the drift detected by AE-DDM is real in nature as it impacts the classification performance of chosen machine learning algorithms.

5. Conclusion and Future Work

In this paper, an autoencoder based drift detection method (AE-DDM) is presented which can detect drift in classification-based (Binary) data streams. The proposed method uses two autoencoders; one trained on the positive class and the other trained on the negative class of non-drifted data. A validation set (non-drifted) is used to compute the batch threshold and count threshold for both classes. The autoencoder with a low batch threshold is placed at layer 1 and the other at layer 2. Incoming batch stream is passed to layer 1 autoencoder and reconstruction loss and exceed counts are computed. If a batch exceeds both thresholds of layer 1 autoencoder, then it is passed to layer 2 autoencoder and

reconstruction error and exceed counts are computed and compared with respective thresholds. If a batch exceeds both batch and count thresholds, a warning is generated and if three consecutive batches exceed both thresholds then drift is confirmed.

Experimental evaluation has been performed on 5 datasets (RBM, Hyperplane, Stagger, Gaussian, and NOAA). Results show that the proposed AE-DDM method for drift detection can effectively detect sudden drift with zero delay. Results on Gaussian and NOAA datasets show the effectiveness of this approach on labelled binary classification datasets. The impact of drift on the classification results of machine learning algorithms shows that the detected drift is real in nature. In case of gradual drift, the proposed method starts generating frequent warnings which can be further investigated. As we have used the simplest possible vanilla autoencoder architecture, there is a huge space for improvement and exploration by trying different types of autoencoders with deep architectures. As a next step, we aim to extend this approach with other autoencoder types and to experiment with a rich collection of datasets.

References

- Baena-García, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldà, R., & Morales-Bueno, R. (2006). Early Drift Detection Method. *4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams*, *6*, 77–86. https://doi.org/10.1.1.61.6101
- Barros, R. S. M., Cabral, D. R. L., Gonçalves, P. M., & Santos, S. G. T. C. (2017). RDDM: Reactive drift detection method. *Expert Systems with Applications*, *90*, 344–355. https://doi.org/10.1016/j.eswa.2017.08.023
- Bashir, S. A., Petrovski, A., & Doolan, D. (2017). A framework for unsupervised change detection in activity recognition. *International Journal of Pervasive Computing and Communications*, *13*(2), 157–175. https://doi.org/10.1108/IJPCC-03-2017-0027
- Bifet, A. (2017). Classifier concept drift detection and the illusion of progress.*Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10246 LNAI*, 715–725. https://doi.org/10.1007/978-3-319-59060-8_64
- Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing.*Proceedings of the 7th SIAM International Conference on Data Mining*, 443–448. https://doi.org/10.1137/1.9781611972771.42
- Brzeziński, D., & Stefanowski, J. (2011). Accuracy updated ensemble for data streams with concept drift.*International Conference on Hybrid Intelligent Systems*, *6679 LNAI*(PART 2), 155–163. https://doi.org/10.1007/978-3-642-21222-2_19
- Bu, L., Alippi, C., & Zhao, D. (2018). A pdf-Free Change Detection Test Based on Density Difference Estimation.*IEEE Transactions on Neural Networks and Learning Systems*, *29*(2), 324–334. https://doi.org/10.1109/TNNLS.2016.2619909
- Bu, L., Zhao, D., & Alippi, C. (2017). An Incremental Change Detection Test Based on Density Difference Estimation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *47*(10), 2714–2726.

https://doi.org/10.1109/TSMC.2017.2682502

- Cabral, D. R. de L., & Barros, R. S. M. de. (2018). Concept drift detection based on Fisher's Exact test. *Information Sciences*, *442–443*, 220–234. https://doi.org/10.1016/j.ins.2018.02.054
- Costa, A. F. J., Albuquerque, R. A. S., & Santos, E. M. Dos. (2018). A Drift Detection Method Based on Active Learning. *Proceedings of the International Joint Conference on Neural Networks*, *2018-July*. https://doi.org/10.1109/IJCNN.2018.8489364
- Dasu, T., Krishnan, S., Venkatasubramanian, S., & Yi, K. (2006). An information-theoretic approach to detecting changes in multi-dimensional data streams. *In Proc. Symp. on the Interface of Statistics, Computing Science, and Applications*.
- de Mello, R. F., Vaz, Y., Grossi, C. H., & Bifet, A. (2019). On learning guarantees to unsupervised concept drift detection on data streams. *Expert Systems with Applications*, *117*, 90–102. https://doi.org/10.1016/j.eswa.2018.08.054
- Ditzler, G., & Polikar, R. (2010a). An ensemble based incremental learning framework for concept drift and class imbalance. *Proceedings of the International Joint Conference on Neural Networks*. https://doi.org/10.1109/IJCNN.2010.5596764
- Ditzler, G., & Polikar, R. (2013a). Incremental Learning of Concept Drift from Streaming Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, *25*(10), 2283–2301. https://doi.org/10.1109/TKDE.2012.136
- Ditzler, G., & Polikar, R. (2010b). An ensemble based incremental learning framework for concept drift and class imbalance. *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 1–8. https://doi.org/10.1109/IJCNN.2010.5596764
- Ditzler, G., & Polikar, R. (2013b). Incremental learning of concept drift from streaming imbalanced dataDitzler, G., & Polikar, R. (2013). Incremental learning of concept drift from streaming imbalanced data. IEEE Transactions on Knowledge and Data Engineering, 25(10), 2283–2301. https://do. *IEEE Transactions on Knowledge and Data Engineering*, *25*(10), 2283–2301. https://doi.org/10.1109/TKDE.2012.136
- Ditzler, G., Roveri, M., Alippi, C., & Polikar, R. (2015). Learning in Nonstationary Environments: A Survey. *IEEE Computational Intelligence Magazine*, *10*(4), 12–25. https://doi.org/10.1109/MCI.2015.2471196
- Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. *Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 71–80. https://doi.org/10.1145/347090.347107
- Dos Reis, D., Flach, P., Matwin, S., & Batista, G. (2016a). Fast unsupervised online drift detection using incremental kolmogorov-smirnov test. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-August*, 1545–1554. https://doi.org/10.1145/2939672.2939836
- Dos Reis, D., Flach, P., Matwin, S., & Batista, G. (2016b). Fast unsupervised online drift detection using incremental kolmogorov-smirnov test. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-August*, 1545–1554. https://doi.org/10.1145/2939672.2939836
- Fan, W. (2004). Systematic data selection to mine concept-drifting data streams. *KDD-2004 - Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 128–137. https://doi.org/10.1145/1014052.1014069
- Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. *Lecture Notes in Computer*

Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), *3171*, 286–295. https://doi.org/10.1007/978-3-540-28645-5_29

- Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation.*ACM Computing Surveys*, *46*(4). https://doi.org/10.1145/2523813
- Gemaque, R. N., Costa, A. F. J., Giusti, R., & dos Santos, E. M. (2020). An overview of unsupervised drift detection methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *10*(6). https://doi.org/10.1002/widm.1381
- Goodfellow, Y. B. and A. (2016). *Deep Learning*. MIT Press.
- Gu, F., Zhang, G., Lu, J., & Lin, C. T. (2016). Concept drift detection based on equal density estimation.*Proceedings of the International Joint Conference on Neural Networks*, *2016-Octob*, 24–30. https://doi.org/10.1109/IJCNN.2016.7727176
- Haque, A., Khan, L., & Baron, M. (2016). SAND: Semi-supervised adaptive novel class detection and classification over data stream. *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, 1652–1658.
- Haque, A., Khan, L., Baron, M., Thuraisingham, B., & Aggarwal, C. (2016). Efficient handling of concept drift and concept evolution over Stream Data. *2016 IEEE 32nd International Conference on Data Engineering, ICDE 2016*, 481–492. https://doi.org/10.1109/ICDE.2016.7498264
- Hinton, G. E., & Zemel, R. S. (1994). Autoencoders, Minimum Description Length and Helmholtz free Energy. *Advances in Neural Information Processing Systems*, *6*, 3–10.
- Hu, H., Kantardzic, M., & Sethi, T. S. (2020). No Free Lunch Theorem for concept drift detection in streaming data classification: A review. In *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*(Vol. 10, Issue 2). Wiley-Blackwell. https://doi.org/10.1002/widm.1327
- Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams.*Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 97–106. https://doi.org/10.1145/502512.502529
- Iwashita, A. S., & Papa, J. P. (2019). An Overview on Concept Drift Learning.*IEEE Access*, *7*, 1532–1547. https://doi.org/10.1109/ACCESS.2018.2886026
- Jaworski, M., Duda, P., & Rutkowski, L. (2018). On applying the Restricted Boltzmann Machine to active concept drift detection. *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017 - Proceedings*, *2018-Janua*, 1–8. https://doi.org/10.1109/SSCI.2017.8285409
- Jaworski, M., Rutkowski, L., & Angelov, P. (2020). Concept Drift Detection Using Autoencoders in Data Streams Processing. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12415 LNAI*, 124–133. https://doi.org/10.1007/978-3-030-61401-0_12
- Jaworski, M., Rutkowski, L., Angelov, P., Artificial, P. A.-I. C. on, & 2020, undefined. (2020). Concept Drift Detection Using Autoencoders in Data Streams Processing. *Springer*, 124–133. https://doi.org/10.1007/978-3-030-61401-0_12
- Kelly, M. G., Hand, D. J., & Adams, N. M. (1999). The impact of changing populations on classifier performance.*KDD '99: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 367–371. https://doi.org/10.1145/312129.312285

- Khamassi, I., Sayed-Mouchaweh, M., Hammami, M., & Ghédira, K. (2018). Discussion and review on evolving data streams and concept drift adapting. *Evolving Systems*, *9*(1). https://doi.org/10.1007/s12530-016-9168-2
- Kifer, D., Ben-David, S., & Gehrke, J. (2004). Detecting Change in Data Streams.*Proceedings 2004 VLDB Conference*, 180–191. https://doi.org/10.1016/b978-012088469-8.50019-x
- Kim, Y., & Park, C. H. (2017). An efficient concept drift detection method for streaming data under limited labeling. *IEICE Transactions on Information and Systems*, *E100D*(10), 2537–2546. https://doi.org/10.1587/transinf.2017EDP7091
- Koh, Y. S. (2016a). CD-TDS: Change detection in transactional data streams for frequent pattern mining.*Proceedings of the International Joint Conference on Neural Networks*, *2016-Octob*, 1554–1561. https://doi.org/10.1109/IJCNN.2016.7727383
- Koh, Y. S. (2016b). CD-TDS: Change detection in transactional data streams for frequent pattern mining.*Proceedings of the International Joint Conference on Neural Networks*, *2016-Octob*, 1554–1561. https://doi.org/10.1109/IJCNN.2016.7727383
- Kolter, J. Zico, & Maloof, M. A. (2007). Dynamic weighted majority: An ensemble method for drifting concepts.*Journal of Machine Learning Research*, *8*, 2755–2790.
- Kolter, Jeremy Z., & Maloof, M. A. (2003). Dynamic weighted majority: A new ensemble method for tracking concept drift. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 123–130. https://doi.org/10.1109/icdm.2003.1250911
- Li, B., Wang, Y. jie, Yang, D. sheng, Li, Y. mou, & Ma, X. kong. (2019). FAAD: an unsupervised fast and accurate anomaly detection method for a multi-dimensional sequence over data stream. *Frontiers of Information Technology and Electronic Engineering*, *20*(3), 388–404. https://doi.org/10.1631/FITEE.1800038
- Liao, J., Zhang, J., & Ng, W. W. Y. (2016). Effects of different base classifiers to Learn++ family algorithms for concept drifting and imbalanced pattern classification problems. *Proceedings - International Conference on Machine Learning and Cybernetics*, *1*, 99–104. https://doi.org/10.1109/ICMLC.2016.7860884
- Liu, A., Lu, J., Liu, F., & Zhang, G. (2018). Accumulating regional density dissimilarity for concept drift detection in data streams. *Pattern Recognition*, *76*, 256–272. https://doi.org/10.1016/j.patcog.2017.11.009
- Liu, A., Song, Y., Zhang, G., & Lu, J. (2017). Regional concept drift detection and density synchronized drift adaptation. *IJCAI International Joint Conference on Artificial Intelligence*, *0*, 2280–2286. https://doi.org/10.24963/ijcai.2017/317
- Liu, G., Cheng, H. R., Qin, Z. G., Liu, Q., & Liu, C. X. (2013). E-CVFDT: An improving CVFDT method for concept drift data stream. *2013 International Conference on Communications, Circuits and Systems, ICCCAS 2013*, *1*, 315–318. https://doi.org/10.1109/ICCCAS.2013.6765241
- Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2019). Learning under Concept Drift: A Review.*IEEE Transactions on Knowledge and Data Engineering*, *31*(12), 2346–2363. https://doi.org/10.1109/TKDE.2018.2876857
- Lu, N., Lu, J., Zhang, G., & Lopez De Mantaras, R. (2016). A concept drift-tolerant case-base editing technique. *Artificial Intelligence*, *230*, 108–133. https://doi.org/10.1016/j.artint.2015.09.009
- Lu, N., Zhang, G., & Lu, J. (2014a). Concept drift detection via competence models.*Artificial Intelligence*, *209*(1), 11–

28. https://doi.org/10.1016/j.artint.2014.01.001

- Lu, N., Zhang, G., & Lu, J. (2014b). Concept drift detection via competence models.*Artificial Intelligence*, *209*(1), 11–28. https://doi.org/10.1016/j.artint.2014.01.001
- Lughofer, E., Weigl, E., Heidl, W., Eitzinger, C., & Radauer, T. (2016a). Recognizing input space and target concept drifts in data streams with scarcely labeled and unlabelled instances. *Information Sciences*, *355–356*, 127–151. https://doi.org/10.1016/j.ins.2016.03.034
- Lughofer, E., Weigl, E., Heidl, W., Eitzinger, C., & Radauer, T. (2016b). Recognizing input space and target concept drifts in data streams with scarcely labeled and unlabelled instances. *Information Sciences*, *355–356*, 127–151. https://doi.org/10.1016/j.ins.2016.03.034
- Maletzke, Andre G., Dos Reis, D. M., & Batista, G. E. A. P. A. (2017). Quantification in data streams: Initial results. *Proceedings - 2017 Brazilian Conference on Intelligent Systems, BRACIS 2017*, *2018-Janua*, 43–48. https://doi.org/10.1109/BRACIS.2017.74
- Maletzke, André G., dos Reis, D. M., & Batista, G. E. A. P. A. (2018). Combining instance selection and self-training to improve data stream quantification. *Journal of the Brazilian Computer Society*, *24*(1). https://doi.org/10.1186/s13173-018-0076-0
- Menon, A. G., & Gressel, G. (2021). Concept Drift Detection in Phishing Using Autoencoders.*Communications in Computer and Information Science*, *1366*, 208–220. https://doi.org/10.1007/978-981-16-0419-5_17
- Muhlbaier, M. D., & Polikar, R. (2007). Multiple classifiers based incremental learning algorithm for learning in nonstationary environments. *Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, ICMLC 2007*, *6*, 3618–3623. https://doi.org/10.1109/ICMLC.2007.4370774
- Muhlbaier, M. D., Topalis, A., & Polikar, R. (2009). Learn++ .NC: Combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes. *IEEE Transactions on Neural Networks*, *20*(1), 152–168. https://doi.org/10.1109/TNN.2008.2008326
- Muhlbaier, M., Topalis, A., & Polikar, R. (2004). Learn++.MT: A new approach to incremental learning.*Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3077*, 52–61. https://doi.org/10.1007/978-3-540-25966-4_5
- Mustafa, A. M., Ayoade, G., Al-Naami, K., Khan, L., Hamlen, K. W., Thuraisingham, B., & Araujo, F. (2017). Unsupervised deep embedding for novel class detection over data stream. *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, *2018-Janua*, 1830–1839. https://doi.org/10.1109/BigData.2017.8258127
- Nick Street, W., & Kim, Y. S. (2001). A streaming ensemble algorithm (SEA) for large-scale classification.*Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 377–382. https://doi.org/10.1145/502512.502568
- Nishida, K., & Yamauchi, K. (2007). Detecting concept drift using statistical testing. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4755 LNAI*, 264–269. https://doi.org/10.1007/978-3-540-75488-6_27
- Nishida, K., Yamauchi, K., & Omori, T. (2005). ACE: Adaptive classifiers-ensemble system for concept-drifting environments. *Lecture Notes in Computer Science*, *3541*, 176–185. https://doi.org/10.1007/11494683_18

- Oladele, S. (2021). *A Comprehensive Guide on How to Monitor Your Models in Production - neptune.ai* Página Oficial Neptune AI. https://neptune.ai/blog/how-to-monitor-your-models-in-production-guide
- *Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds* (2015).
- Page, E. S. (1954). Continuous Inspection Schemes. *Biometrika*, *41*(1/2), 100. https://doi.org/10.2307/2333009
- Pesaranghader, A., & Viktor, H. L. (2016). Fast hoeffding drift detection method for evolving data streams.*Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9852 LNAI*, 96–111. https://doi.org/10.1007/978-3-319-46227-1_7
- Pesaranghader, A., Viktor, H. L., & Paquet, E. (2018). McDiarmid Drift Detection Methods for Evolving Data Streams. *Proceedings of the International Joint Conference on Neural Networks*, *2018-July*. https://doi.org/10.1109/IJCNN.2018.8489260
- Pinagé, F., dos Santos, E. M., & Gama, J. (2020). A drift detection method based on dynamic classifier selection.*Data Mining and Knowledge Discovery*, *34*(1), 50–74. https://doi.org/10.1007/s10618-019-00656-w
- Polikar, R., Udpa, L., Udpa, S. S., & Honavar, V. (2001). Learn++: An incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *31*(4), 497–508. https://doi.org/10.1109/5326.983933
- Qahtan, A., Alharbi, B., Wang, S., & Zhang, X. (2015a). A PCA-based change detection framework for multidimensional data streams. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *2015-August*, 935–944. https://doi.org/10.1145/2783258.2783359
- Qahtan, A., Alharbi, B., Wang, S., & Zhang, X. (2015b). A PCA-based change detection framework for multidimensional data streams. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *2015-August*, 935–944. https://doi.org/10.1145/2783258.2783359
- Richard O. Duda, Peter E. Hart, D. G. S. (2000). *Pattern Classification* (2nd ed.). Wiley-Interscience.
- Schelter, S., Biessmann, F., Januschowski, T., Salinas, D., Seufert, S., & Szarvas, G. (2018). On Challenges in Machine Learning Model Management. In *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* (pp. 5–13). http://sites.computer.org/debull/A18dec/p5.pdf
- Schlimmer, J. C., & Granger, R. H. (1986). Incremental Learning from Noisy Data.*Machine Learning*, *1*(3), 317–354. https://doi.org/10.1023/A:1022810614389
- Schröder, T., & Schulz, M. (2022). Monitoring machine learning models: A categorization of challenges and methods. In *Data Science and Management*. https://doi.org/10.1016/j.dsm.2022.07.004
- Sethi, T. S., & Kantardzic, M. (2015). Don't pay for validation: Detecting drifts from unlabeled data using Margin Density. *Procedia Computer Science*, *53*(1), 103–112. https://doi.org/10.1016/j.procs.2015.07.284
- Sethi, T. S., & Kantardzic, M. (2017). On the reliable detection of concept drift from streaming unlabeled data.*ArXiv*.
- Sethia, T. S., & Kantardzic, M. (2018). Handling adversarial concept drift in streaming data.*ArXiv*.
- Shao, J., Ahmadi, Z., & Kramer, S. (2014). Prototype-based learning on concept-drifting data streams. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 412–421. https://doi.org/10.1145/2623330.2623609
- Song, X., Wu, M., Jermaine, C., & Ranka, S. (2007). Statistical change detection for multi-dimensional

data. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 667–676. https://doi.org/10.1145/1281192.1281264

- Soppin, S., Ramachandra, M., & Chandrashekar, B. N. (2021). *Essentials of Deep Learning and AI: Experience Unsupervised Learning, Autoencoders, Feature Engineering, and Time Series Analysis with TensorFlow, Keras, and scikit-learn (English Edition)*.
- Spinosa, E. J., De Carvalho, A. P. D. L. F., & Gama, J. (2007). OLINDDA: A cluster-based approach for detecting novelty and concept drift in data streams. *Proceedings of the ACM Symposium on Applied Computing*, 448–452. https://doi.org/10.1145/1244002.1244107
- Wald, A. (1973). *Sequential Analysis*. DOVER PUBLICATIONS, INC.
- Wang, Haixun, Fan, W., Yu, P. S., & Han, J. (2003a). *Mining concept-drifting data streams using ensemble classifiers* 226. https://doi.org/10.1145/956755.956778
- Wang, Haixun, Fan, W., Yu, P. S., & Han, J. (2003b). Mining concept-drifting data streams using ensemble classifiers. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 226–235. https://doi.org/10.1145/956750.956778
- Wang, Heng, & Abraham, Z. (2015). Concept drift detection for streaming data. *Proceedings of the International Joint Conference on Neural Networks*, *2015-Septe*. https://doi.org/10.1109/IJCNN.2015.7280398
- Wang, S., Minku, L. L., Ghezzi, D., Caltabiano, D., Tino, P., & Yao, X. (2013). Concept Drift Detection for Online Class Imbalance Learning. *Proceedings of the International Joint Conference on Neural Networks*. https://doi.org/10.1109/IJCNN.2013.6706768
- Wares, S., Isaacs, J., & Elyan, E. (2019). Data stream mining: methods and challenges for handling concept drift. *SN Applied Sciences*, *1*(11). https://doi.org/10.1007/s42452-019-1433-0
- Widmer, G. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*(1), 69–101. https://doi.org/10.1007/bf00116900
- Yong, B. X., Fathy, Y., & Brintrup, A. (2020a). Bayesian Autoencoders for Drift Detection in Industrial Environments. *2020 IEEE International Workshop on Metrology for Industry 4.0 and IoT, MetroInd 4.0 and IoT 2020 - Proceedings*, 627–631. https://doi.org/10.1109/MetroInd4.0IoT48571.2020.9138306
- Yong, B. X., Fathy, Y., & Brintrup, A. (2020b). Bayesian Autoencoders for Drift Detection in Industrial Environments. *2020 IEEE International Workshop on Metrology for Industry 4.0 and IoT, MetroInd 4.0 and IoT 2020 - Proceedings*, 627–631. https://doi.org/10.1109/MetroInd4.0IoT48571.2020.9138306
- Yu, S., & Abraham, Z. (2017). Concept drift detection with hierarchical hypothesis testing. *Proceedings of the 17th SIAM International Conference on Data Mining, SDM 2017*, 768–776. https://doi.org/10.1137/1.9781611974973.86
- Zliobaite, I., Bifet, A., Pfahringer, B., & Holmes, G. (2014). Active learning with drifting streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(1), 27–39. https://doi.org/10.1109/TNNLS.2012.2236570