Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology


Master's Thesis

# Evaluation of Drift Detection Techniques for Automated Machine Learning Pipelines

*Hammam Abdelwahab*

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Alexander Asteroth
Prof. Dr. Nico Hochgeschwender
MSc. Claudio Martens

January 2022

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

| | |
|---|---|
| Date | Hammam Abdelwahab |

# Abstract

Machine learning-based solutions are frequently adapted in several applications that require big data in operations. The performance of a model that is deployed into operations is subject to degradation due to unanticipated changes in the flow of input data. Hence, monitoring data drift becomes essential to maintain the model's desired performance. Based on the conducted review of the literature on drift detection, statistical hypothesis testing enables to investigate whether incoming data is drifting from training data. Because Maximum Mean Discrepancy (MMD) and Kolmogorov-Smirnov (KS) have shown to be reliable distance measures between multivariate distributions in the literature review, both were selected from several existing techniques for experimentation. For the scope of this work, the image classification use case was experimented with using the Stream-51 dataset. Based on the results from different drift experiments, both MMD and KS showed high Area Under Curve values. However, KS exhibited faster performance than MMD with fewer false positives. Furthermore, the results showed that using the pre-trained ResNet-18 for feature extraction maintained the high performance of the experimented drift detectors. Furthermore, the results showed that the performance of the drift detectors highly depends on the sample sizes of the reference (training) data and the test data that flow into the pipeline's monitor. Finally, the results also showed that if the test data is a mixture of drifting and non-drifting data, the performance of the drift detectors does not depend on how the drifting data are scattered with the non-drifting ones, but rather their amount in the test set.

# Acknowledgements

Throughout the writing of this thesis, I've been generously supported and guided to give the best results obtainable. As a start, I would like to thank Dr. Alexander Asteroth for the continuous guidance on how to conduct research properly and for the consistent help throughout my project period. Also, I would like to thank Prof. Nico Hochgeschwender for the fruitful feedbacks and help. Nevertheless, I would like to thank Mr. Claudio Martens for his consistent aid and helpful pieces of advice. Furthermore, I would like to thank Dr. Dennis Wegener, the head of the MLOps Team in Fraunhofer IAIS for his ceaseless availability for guidance and for facilitating the provision of all the tools to perform the necessary experiments. Special thanks to Hochschule Bonn-Rhein-Sieg and Fraunhofer IAIS for trusting me with this challenging yet interesting research topic. I wish the contributions of this work assist in providing more solutions related to the related area of research to this work. Finally, I would like to thank all my family members for being there for me through the ups and downs of this master's journey. I wouldn't be able to do it all without any of them.

# Contents


**1 Introduction** **1**

1.1 Motivation . . . 2
1.2 Challenges and Difficulties . . . 2
1.3 Monitoring Drifts as Part of MLOps . . . 4
1.4 Problem Statement . . . 6

**2 Related Work** **7**

2.1 Drift Detection in Research . . . 7
2.2 Dataset Drift . . . 7
2.2.1 Covariate Drift . . . 8
2.2.2 Prior Probability Drift . . . 8
2.2.3 Concept Drift . . . 8
2.3 Dataset Drift Detection . . . 8
2.4 Drift Detection in Data Mining . . . 8
2.5 Drift Detection in Deep Learning . . . 11
2.6 Concept Drift, Out-of-Distribution, and Anomaly Detection . . . 13
2.7 Which Algorithms to Select for Evaluation? . . . 17

**3 Methodology** **19**

3.1 Sampling Training Data . . . 19
3.2 Feature Extraction . . . 20
3.2.1 Principle Component Analysis . . . 21
3.2.2 Sparse Random Projection (SRP) . . . 21
3.2.3 Neural Networks as Feature Extractors . . . 22
3.3 Two-Sample Hypothesis Test with Maximum Mean Discrepancy . . . 23
3.4 Maximum Mean Discrepancy . . . 25
3.5 Kolmogorov-Smirnov Test . . . 27
3.6 Evaluation Methods . . . 30
3.7 Case Study for Drift Evaluation . . . 31
3.7.1 Image Classification . . . 31
3.8 Experiments' Setup . . . 32
3.8.1 The Monitoring Task . . . 32
3.8.2 Setup of Drift Data . . . 32
3.8.3 Experiment Phases . . . 34
3.9 Assumptions . . . 34

**4 Construction of The Experimental Pipeline** **37**
4.1 The Experimental Pipeline . . . 37
4.2 Feature Extractors . . . 38
4.3 Drift Detectors . . . 40
4.3.1 Detector's Monitoring Buffer . . . 40
4.4 Drift Alarm . . . 41
4.5 Tools . . . 41

**5 Results** **43**
5.1 Drift Detection for Image Classification . . . 43
5.1.1 Ideal Conditions for Drift Experiments . . . 44
5.1.2 Overcoming False-Positives with Sub-Sampling Reference Set . . . 47
5.1.3 Non-Ideal Conditions for Drift Experiments . . . 48
5.2 Performance Analysis of Drift Detectors . . . 51
5.2.1 Performance Analysis for MMD . . . 51
5.2.2 Performance Analysis for KS . . . 52
5.3 Comparative Analysis between Detectors . . . 52
5.3.1 Comparison of Time Performance for Detectors . . . 54
5.3.2 Comparative Analysis with Adding Dimension Reducers . . . 54
5.3.3 Comparative Analysis with Different Drift Ratios . . . 55

**6 Discussions** **59**
6.1 The Performance in Ideal Drift Cases . . . 59
6.1.1 Performance of MMD Detector in Ideal Drift Cases . . . 59
6.1.2 Performance of KS Detector in Ideal Drift Cases . . . 60
6.1.3 Overcoming False-Positives with Sub-Sampling . . . 60
6.2 The Performance in Non-Ideal Drift Cases . . . 60
6.2.1 Performance of MMD Detector in Non-ideal Drift Cases . . . 60
6.2.2 Performance of KS Detector in Non-ideal Drift Cases . . . 61
6.3 Effect of Feature Extraction on Drift Detection . . . 61
6.4 Effect of Sample Size on Drift Detection . . . 61
6.5 Effect of the Order of Drifts on Drift Detection . . . 61
6.6 Which Algorithm Performs Best? . . . 62

**7 Conclusions** **63**
7.1 Revisiting The Problem Statement . . . 63
7.2 Contributions . . . 64
7.3 Areas of Relevant Applications . . . 64
7.4 Future work . . . 65

**Appendix A Additional Analysis on P-values** **67**
A.1 Relationship between P-values and Drift Ratios . . . . . 67

**Appendix B Relationship Between P-values and KS Distances** **71**

**Appendix C Analysis of MMD with Bootstrapping** **73**

**Appendix D AUC of Drift Detectors with Feature Extractors and Dimension Reducers** **75**

**References** **79**

# List of Figures


1.1 A simplified machine learning components in production [18, p.4]. . . . . 1
1.2 Difference between monitoring in a regular software system and a machine learning-based system [14, p.2]. . . . . 3
1.3 Manual ML steps to serve the model as a prediction service with CD automation [44]. . . 4
1.4 ML pipeline automation for continuous training in production (MLOps level 1) [44]. . . . 5
1.5 ML pipeline automation for continuous training in production with CI/CD Automation (MLOps level 2) [44]. . . . . 6

2.1 Example of a drifting data in three phases: (a)Before drift with two classes. (b)Drift phase with a new class in a new data region. (c)Drift phase with a new class in an existing data region [42, p. 4]. . . . . 10
2.2 Categorization of drifts based on speed [30, p. 6]. . . . . 10
2.3 Categorization of concept drifts types to (a)fixed space drift. (b) non-fixed space drift [42, p. 6]. . . . . 11
2.4 Demonstration of ODIN. Based on the drift signal, a specializer determines which model ensemble fits the input best by clustering the drifting input [79, p. 3]. . . . . 13
2.5 Model construction in quality characteristic prediction of WIP products [94, p. 9]. . . . . 14
2.6 The components involved with anomaly detection techniques [17, p. 4]. . . . . 15
2.7 Projection of data points in one,two, and three dimensions respectively. [83, p. 8] . . . . . 16
2.8 Projection of data points in one,two, and three dimensions respectively [66, p. 2]. . . . . 17
2.9 Difference between data drifts and outliers [74]. . . . . 18

3.1 A General schema for designing monitoring system for drift detection. . . . . 20
3.2 Demonstration of how training data were utilized for drift detection experiments. . . . . 20
3.3 Relationship between number of components and distance errors (By Scikit-learn [65]). . . 22
3.4 Example of a neural network as a dimension reducer [89, p .2]. . . . . 23
3.5 Demonstration of how two-sample test was implemented to detect drifts. . . . . 25
3.6 Example of the grid representation for Hodges's method [41, p. 3]. . . . . 29
3.7 Example of the implementation of the inner method [90, p. 1]. . . . . 30
3.8 Examples of labeled images from Stream-51 dataset [71, p. 4]. . . . . 31
3.9 Demonstration of the three drift types to be tested. . . . . 32
3.10 Examples of test data including known labels and unknown labels. . . . . 33
3.11 Example of a known class with different levels of Gaussian noise. . . . . 33
3.12 Structure of the experiments' three phases. . . . . 35
3.13 A minimal set of tasks to train and deploy a model as part of a machine learning pipeline. 36

4.1 UML of the Automated Machine Learning Pipeline for Drift experiments. . . . . 38
4.2 ResNet Architecture for Image classification [39, p. 773]. . . . . 39
4.3 Expected behaviors of a drift detector during no drifts and drift phases. . . . . 41
4.4 Drift monitoring task as part of a machine learning pipeline. . . . . 42

5.1 Demonstration of the monitoring buffer during experiments in ideal conditions. . . . . 44
5.2 Demonstration of the monitoring buffer during initial experiments. . . . . 45
5.3 Distances, p-values of MMD and KS detector in ideal experimental conditions with Subsampling. . . . . 49
5.4 Distances, p-values of MMD and KS detector in non ideal experimental conditions. . . . . 50
5.5 Relationship between p-values and distances based on the inner method for 100 sample sizes. 54
5.6 ROC for MMD and KS in ideal drift conditions. . . . . 55
5.7 ROC Curves for drift detectors under non-ideal conditions. . . . . 56
5.8 Relationship between p-values and drift ratios for buffer size = 100 for MMD. . . . . 57
5.9 Relationship between p-values and Drifts for MMD with size 10. . . . . 57
5.10 Relationship between p-values and drift ratios for buffer size = 100 for KS. . . . . 58
5.11 Relationship between p-values and Drifts for KS with size 10. . . . . 58

A.1 Relationship between p-values and Drifts for MMD with size 10. . . . . 67
A.2 Relationship between p-values and Drifts for MMD with size 100. . . . . 68
A.3 Relationship between p-values and Drifts for MMD with size 1000. . . . . 68
A.4 Relationship between p-values and Drifts for KS with size 10. . . . . 69
A.5 Relationship between p-values and Drifts for KS with size 100. . . . . 69
A.6 Relationship between p-values and Drifts for KS with size 1000. . . . . 70

B.1 Relationship between p-values and distances based on the inner method for 10 sample sizes. 71
B.2 Relationship between p-values and distances based on the inner method for 1000 sample sizes. 72

C.1 Relationship between p-values and distances based on the inner method for 100 sample sizes. 73
C.2 Relationship between p-values and distances based on the inner method for 100 sample sizes. 74

D.1 ROC for MMD and KS detectors without a dimension reducer. . . . . 75
D.2 ROC for MMD and KS detectors with PCA. . . . . 76
D.3 ROC for MMD and KS detectors with SRP. . . . . 77
D.4 ROC for MMD and KS detectors with UAE. . . . . 78

# List of Tables


2.1 A summary of drift types based on types of criteria [42, p. 5]. . . . . 10
2.2 Categorization of drift detection techniques with examples for each category [42]. . . . . 12
2.3 Categorization of anomaly detection techniques with examples for each category [17]. . . . 14

4.1 Number of dimensions for the output features for each extractor. . . . . 40

5.1 The initial configuration settings for drift testing. . . . . 44
5.2 Results from the MMD drift detector under testing in ideal experimental conditions. . . . 46
5.3 Results from the KS drift detector under testing in ideal experimental conditions. . . . . 47
5.4 Results from the MMD drift detector in ideal conditions using Sub-sampling. . . . . 48
5.5 Results from the KS drift detector in ideal conditions using Sub-sampling. . . . . 48
5.6 Results from the MMD drift detector under testing in non ideal experimental conditions . 51
5.7 Results from the KS drift detector under testing in non-ideal experimental conditions. . . 52
5.8 Analysis of p-value calculation using Bootsrapping for MMD Detector. . . . . 53
5.9 Average AUC for MMD under different drift conditions. . . . . 53
5.10 Time Performance for MMD and KS. . . . . 54
5.11 Average AUC for MMD drift detector with dimension reducers. . . . . 55
5.12 Average AUC for KS drift detector with dimension reducers . . . . . 56

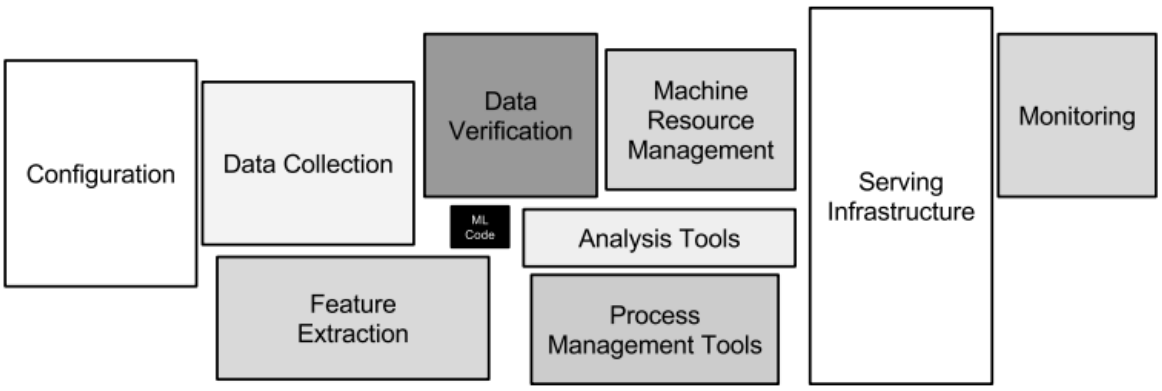
6.1 A summarized (general) comparison between drift detectors based on the obtained results. 62

# 1

# Introduction

Machine learning communities are enormously growing every day, especially in the research field. It's believed that several real-world problems can be solved using machine learning as continual researches are ongoing for this purpose. Given the proper training data, a machine learning model can understand a problem's dynamics and provide an estimated solution within an estimated range of certainty. Several fields such as computer vision, natural language understanding, and time-series analysis have adopted machine learning as a primary technology for data-driven business solutions. Still, most of the ongoing development of solutions lies mainly within an academic context with fewer details about how to automate deploying, monitoring, or maintaining the model's performance [24]. Once the trained model is deployed into the real world, different new challenges appear. Hence, the model's reliability will be subject to degradation [77]. After building, training, evaluating, and testing a machine learning-based solution, the next point of focus is to handle it such that the developed models are staged and later deployed into the real world. To do so, there exist many challenges that need to be considered before allowing the developed model to take accountability in any problem. This includes the components of reliable software design that regards configuration issues, data dependency, hidden feedback loops, and handling the changes in the external world [18]. Figure 1.1 illustrates some of the required components to develop an accountable machine learning-based solution as software.

Figure 1.1: A simplified machine learning components in production [18, p.4].



From Figure 1.1, it is shown that obtaining a highly performing and efficient machine learning solution with proper implementation resembles a small component among the rest of the components. It should be noted that all these components form a machine learning lifecycle and are usually integrated to build a pipeline for the purpose of automation, and smooth operations handling. In such pipelines, the deployed model is contained within a complex ecosystem of interconnected tasks [18]. Managing machine learning pipelines in an efficient and standardized manner is a challenge on which that ongoing research is being conducted [13]. Nevertheless, new tools are being frequently deployed to facilitate handling machine learning operations. To handle machine learning pipelines for a single model or a set of models, a growing field of research has been acknowledged under the name of MLOps that stands for Machine Learning Operations. The definition of MLOps has now become "The standardization of Machine Learning Life Cycle management" [86]. While each component of the machine learning pipeline has its challenges to efficiently contribute to the lifecycle, performance monitoring defines unique issues. When events that impact the models' performances occur, alarms should be immediately raised [24]. Furthermore, while traditional monitoring methods include logging performance service metrics such as the request latency, storage, and serving frequency, machine learning applications require special metrics to monitor their performances given the interchangeability of their functionality along with different environments. These metrics include model performance, data-related metrics, drifts metrics, and explainability metrics [47]. One motivation to have these metrics is to mitigate the degradation of the model's performance unknowingly. For example, for a model that is deployed into the real world is classifying images, it is not possible to know if the model's accuracy remained high until the true labels are collected. While in some applications, collecting the true labels consume time, proper drift metrics can indicate changes in the flow of incoming input or predictions which aids in making decisions whether to consider the model's prediction or not [47]. Additionally, in environments where data evolves, there is an ongoing requirement for adapting the used models to the new data such that the high performance remains consistent. This adaptation requires an informant that triggers the action to start adapting and learning continuously [24].

## 1.1 Motivation

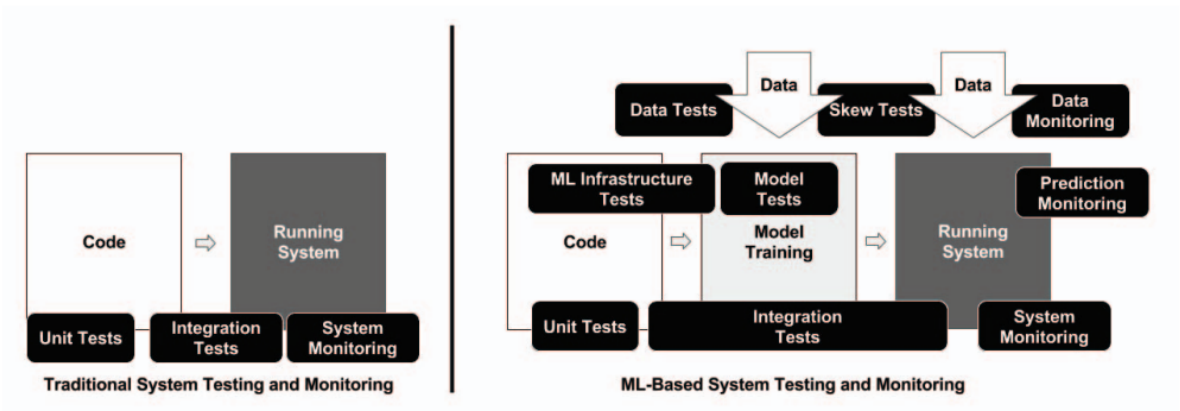
Monitoring a model's performance is an essential component in the deployment ecosystem as seen in Figure 1.1. As previously discussed, it is important to mitigate the degradation of the model's performance unknowingly [47]. Nevertheless, the frequently changing environment requires instant change identification and requires fast reactions to adapt to it. Additionally, developing trust in machine learning models through a transparent prediction process is needed such that further decision-making steps would be based on solid results from the models. While this poses an issue with black box-based models, monitoring metrics can provide an indicator to whether the predictions can be reliable or not [14] [47].

## 1.2 Challenges and Difficulties

A classical aspect of model monitoring consists of observing the training step's accuracy, receiver operating characteristic curve (ROC), Area under the ROC curve (AUC), Precision, and Recall. After deployment of the trained model, the monitoring component keeps track of software performance measures

such as requests per second, latency rate, utilization of CPU and memory, and any custom metric that aids in ensuring high-quality production services. Because the prediction's ground truth might not be instantly available, monitoring the model's performance using the same training metrics is timely expensive, specifically, if the deployed model is serving in a dynamic environment [47]. With regard to this challenge, many research papers redirected the focus from monitoring the model's training metrics to monitoring data statistics as an indirect indicator of the model's performance [3][25][47][60][66]. Due to the monotonic nature of regular software systems, building pipelines for testing and maintenance encompass software's source code only. Yet, for software systems based on machine learning, the situation varies based on the nature of the incoming data into the used models [66]. One of the main challenges facing deployed models is the constant drifts of data and problem characteristics over time. This problem requires taking constant monitoring to enable taking actions such as parameter tuning or model updating algorithms [47]. In a production environment where one model or several models are in an operation mode, identifying statistical metrics for monitoring is challenging. Furthermore, the manual tracking of models' performances is costly if the real-world environment has a fast dynamic behavior. Therefore, there is a need to automate model monitoring such that drifts are identified [66][47]. Figure 1.2 demonstrates the difference between a traditional monitoring system and a machine learning-based monitoring system. Based on this figure, the traditional monitoring system ensures the source code is running as desired by using unit tests. Also, the traditional system ensures that the source code can be integrated into other components of a running system by conducting integration tests. On the other side, Figure 1.2 shows how machine learning-based monitoring systems add model testing with model training and prediction monitoring. Finally, this monitoring system monitors incoming data, runs data tests, and checks if the incoming data is skewing (drifting) or not [14]. This reflects how challenging it can be to develop a monitoring system for a machine learning-based software and investigation on how to develop the monitoring task is important for such systems.

Figure 1.2: Difference between monitoring in a regular software system and a machine learning-based system [14, p.2].

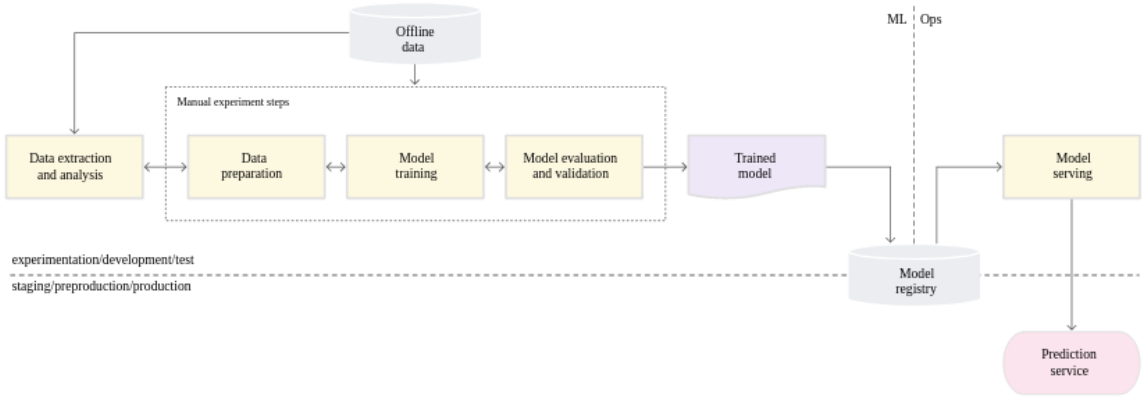


## 1.3 Monitoring Drifts as Part of MLOps

In both the research and in practice fields, developing tools that support handling Machine Learning Operations and developing automated pipelines is receiving growing interests [80]. Tools such as Apache Airflow and Kubeflow aim to break down jobs into tasks in various cloud platforms such as Microsoft Azure, Amazon Web Services (AWS), or Google Cloud. According to Google Cloud's article entitled "MLOps: Continuous delivery and automation in machine learning pipelines" [44], MLOps practices can be divided into three levels based on the maturity of the machine learning pipeline's architecture. MLOps defines machine learning architectures based on their maturity to three levels starting from zero to two. In MLOps level 0, the process of training, deploying, and retraining the model is manual. Figure 1.3 explains in detail the components of an MLOps architecture from preparing the training dataset until serving the model. From the figure, it is shown that MLOps level 0's architecture is divided into ML part on the left and Ops part on the right. On the left part, the processes of data preparation, model training, evaluation, and validation are done manually. The model is then stored in a model registry. On the right part, the model is obtained from the registry and deployed to start serving in operation mode. In the figure, the processes are also divided by a horizontal line such that the upper part includes the tasks implemented during development and experimentation while the lower part (includes prediction service) is the operational part which operates in staging, preproduction, or in production.

For MLOps level 1, instead of deploying a model into production, the source code for the whole pipeline

Figure 1.3: Manual ML steps to serve the model as a prediction service with CD automation [44].

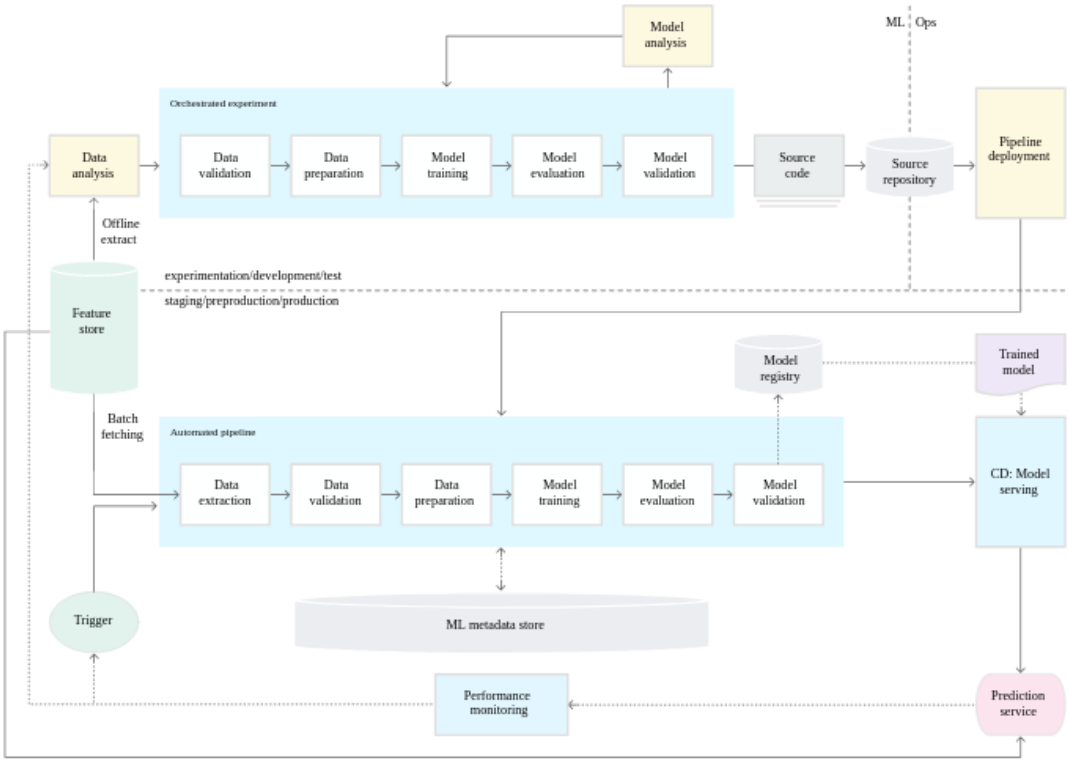



(MLOps level 0's pipeline) is deployed into a source repository during development (experimentation/development/test Parts) as shown in Figure 1.4. According to this figure, the model can be retrained in production and be deployed again based on the desired training configurations in the ML metadata store. The ability to retrain the model and redeploy it ensures continuous delivery or (CD) for model serving in production. Furthermore, the retraining is initiated an incoming trigger using the deployed pipeline. Also from the figure, the trigger is fired as an outcome of the performance monitoring which works by taking

prediction results and data from the feature store as inputs. In this architecture, it is shown how the performance monitoring task plays a crucial role in updating the model in case it is in an undesired state.

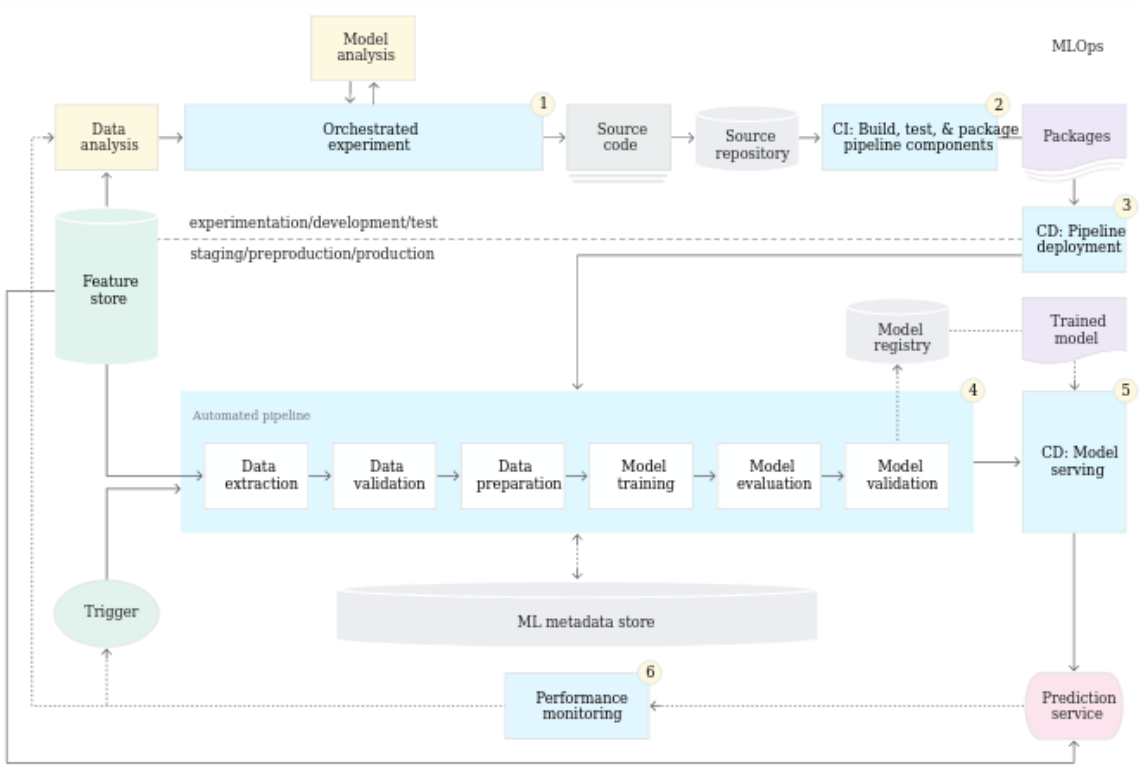
For MLOps level 2, Figure 1.5 demonstrates its architecture. From the figure, it is shown that the task

Figure 1.4: ML pipeline automation for continuous training in production (MLOps level 1) [44].




of orchestrated experiments is added to the MLOps level 1 pipeline such that the system is able to run experiments to analyze its performance and to select which model to use in which scenario. The results of these orchestrated (automated) experiments enable to implement continuous integration or (CI) to the deployed pipelines such that the serving model (or models) is able to continuously serve incoming requests by providing predictions with the highest performances possible [44]. While this architecture resembles full automation of a machine learning pipeline, the focus of this project is on MLOps level 1 because it is simpler, it includes most of the components in Figure 1.1, and it enables to focus on the performance monitoring task without the need to automate the entire pipeline as in MLOps level 2.

Figure 1.5: ML pipeline automation for continuous training in production with CI/CD Automation (MLOps level 2) [44].




## 1.4 Problem Statement

To maintain the performance of a software system that uses a machine learning model in a machine learning pipeline, a monitoring task that can detect data drifts should be developed. However, selecting the appropriate detection technique for model monitoring requires the evaluation of the existing ones first. Therefore, a literature review of the state-of-the-art drift detection techniques for model monitoring is to be conducted. In addition, a selection set of drift detection algorithms from the literature review is to be experimented, analyzed, and evaluated. Finally, the technical requirements to develop a data monitoring task and integrate it into the pipeline are to be addressed.

# 2

# Related Work

In this chapter, a literature review of the existing techniques for drift detection is conducted. The term drift is to be defined, and related work in the context of anomaly detection, out-of-distribution detection, and outlier detection is discussed.

## 2.1 Drift Detection in Research

When a machine learning model is deployed into the real world, the model handles received input data from the world's environment such that a prediction is estimated. However, the input data is constantly subject to change, and the model's performance might consequently deteriorate. Therefore, the model should be able to adapt to these changes accordingly. In this context, the change is defined as concept drift [10]. For a set of input features $x$ which are also defined as covariates, the target $y$ is generated by the model. The concept drift is defined as the change in the joint distribution $p(x, y)$ [60]. Generally, there are two problems from which we can define concept drift mathematically. The first problem is when the targets $Y$ are determined by the covariates $X$. We describe this problem as $X \rightarrow Y$, and the joint distribution is defined as follows.

$$p(x, y) = p(y|x)p(x) \tag{2.1}$$

The second problem is when the targets $Y$ determine the covariates $X$. For example, in medical diagnosis, detecting a disease from an image determines the symptoms represented as covariates. The problem is then defined as $Y \rightarrow X$, and the joint distribution becomes as follows.

$$p(x, y) = p(x|y)p(y) \tag{2.2}$$

The term "concept drift" has different definitions and is used in several contexts. The definition provided in this work is considered as a unified definition [60]. It should be mentioned that the concept drift term is also known as the dataset drift [60]. It should also be noted that the terms concept drift and drift are to be used interchangeably, yet both indicate the same meaning.

## 2.2 Dataset Drift

Based on the given definitions of dataset drift, drifts can be categorized into three parts: the covariate drifts, prior probability drift, and concept drift [60].

### 2.2.1 Covariate Drift

The covariate drift is defined as the change in the joint distribution caused by the change in the covariate's prior distributions. Specifically, the covariate drift is the change in $p(x, y)$ due to the change in $p(x)$ from the training distribution $p_{tr}(x)$ while the conditional distributions are fixed.

### 2.2.2 Prior Probability Drift

When a machine learning problem is defined as X to Y ($X \rightarrow Y$), then the change in the prior distribution is the change in $p(x)$ caused by a covariate drift. However, if the problem is from Y to X ($Y \rightarrow X$), then the change in the prior distribution becomes the change in $p(y)$ while the conditional distributions are fixed.

### 2.2.3 Concept Drift

The concept drift is defined as the change in the joint distribution $p(x, y)$ caused by the mutation in the conditional probability distribution ($p(x|y)$ in $Y \rightarrow X$ problems or $p(y|x)$ in $X \rightarrow Y$ problems) while the prior distribution remains constant.

## 2.3 Dataset Drift Detection

The dataset drift problem is being researched in several fields in different contexts. Although different terms exist to identify drifts (concept drift or dataset shift), all these terms defined the problem either as the change in the distribution or defined as the change in the data generating process [84]. In the field of domain adaptation, the Importance Reweighting algorithm and the Kernel Mean Matching approaches successfully demonstrated overcoming concept drifts [98]. However, a comprehensible detection method that can separate out- of-distribution samples rather than implementing required corrections techniques is needed.

## 2.4 Drift Detection in Data Mining

Before using the term drift for machine learning pipelines or machine learning operations (MLOps), the problem of detecting drifts has been widely and explicitly discussed in the field of data mining. Data mining processes include several tasks such as data streaming, data storing, and feature selection to extract efficient information set for machine learning model training or decision making. The term data stream is used to describe the sequences of data flowing through time. An example of streaming data could be measurement readings coming from a distributed set of sensors, or network traffic [10].

According to Barddal et al. and Hu et al. when it comes to streaming, the most common learning problem is classification. In a classification problem, a learner learns from features extracted from the data streams [10] [42]. The drift detection is then defined by what is called "feature drift". Also, Barddal et al. defined feature drift using the term relevance as a generic term for similarity measure [10]. The

higher the relevance value, the less likely to have a feature drift case. The feature drift is equivalent to the covariate drift in which the distribution of features changes over time [61]. One way of detecting drifts is through ensembling. The ensembling approach makes use of the feature selector as part of the model. The idea is to create multiple models and train them separately to produce the training targets. For drifting features, the models react differently. This is reflected either by the model's information gain or the entropy of the classification predictions [69].

In the context of data mining, feature drift testing depends on the preceding feature selection task. Therefore, selecting the proper features is a necessary preliminary step to facilitate efficient features drift. However, selecting the right features in itself is not a trivial step. According to Barddal et al., feature selection could be either manually done by experts as in the Very Fast Decision Tree (VFDT), Facil, and Very Fast Decision Rules (VFDR) for data classification, or implicitly implemented by the learning model as in Concept-adapting Very Fast Decision Tree (CVFDT), Heterogeneous Ensemble for Data Stream (HEFT-Stream), and Hoeffding Adaptive Tree (HAT) for data classification. These algorithms for feature selection were experimented in order to evaluate their ability to implicitly overcome drifts. Hence, there were no explicit definitions of feature drift detection tasks embedded within them nor there was an explicit metric to measure drifts [10].

Furthermore, the feature drift issue was discussed more comprehensively by Hu et al. such that, novel classes contain novel features. In an example of data streaming, with two classes "triangle" and "circle", a new square class (the novel class) is served as an input to the classifier. This new class's features could either remain the decision boundary within the region of the decision boundary or exist in a new region as shown in Figure 2.1 [42].

Additionally, the definition of drifts has been categorized by Gama et al. into two types. The first type is called single drifts. It looks for a current drift in the flow of features. The second type is called drift sequence in which all historical drifts in the feature flow are taken into consideration. For the single drifts, the criterion of drift detection is based on the speed of the drifts, and the distribution of the changing features. Based on the drift's speed, it could be defined as abrupt, sudden, gradual, or incremental drifts. Figure 2.2 demonstrates these types of drifts based on speed [30].

Also, based on the distribution of changes, the drift is defined either as a fixed space drift, or a non-fixed space. In fixed space drifts, there exist a reference distribution to compare with the changing features' distribution. The assumption here is that this distribution is fixed and doesn't change with drifts, unlike in the non-fixed space where the distribution changes along with the individual changes of classes distributions as drifts occur. A summary of the categorization is shown in Figure 2.3 [42] .

After defining the drift type, the task of detecting drifts can be broken down into performance-based detection, and distribution-based detection. The performance-based detection monitors a model's performance metrics such as accuracy, precision, and F-score. This category requires obtaining the prediction's ground truth to fire a signal in case a drift occurs. Hence, it is defined as a supervised approach. In contrast, distribution-based detection is defined as an unsupervised approach in which the change in data distribution is monitored by tracking the distribution location, density, or range through time. For performance-based detection, there exist several algorithms which use performance metrics.

Figure 2.1: Example of a drifting data in three phases: (a)Before drift with two classes. (b)Drift phase with a new class in a new data region. (c)Drift phase with a new class in an existing data region [42, p. 4].

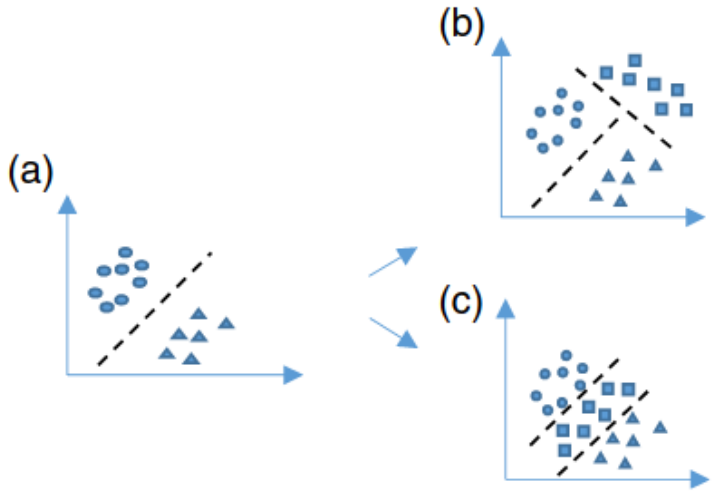



Figure 2.2: Categorization of drifts based on speed [30, p. 6].

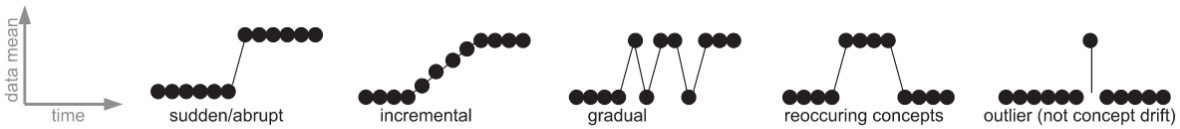
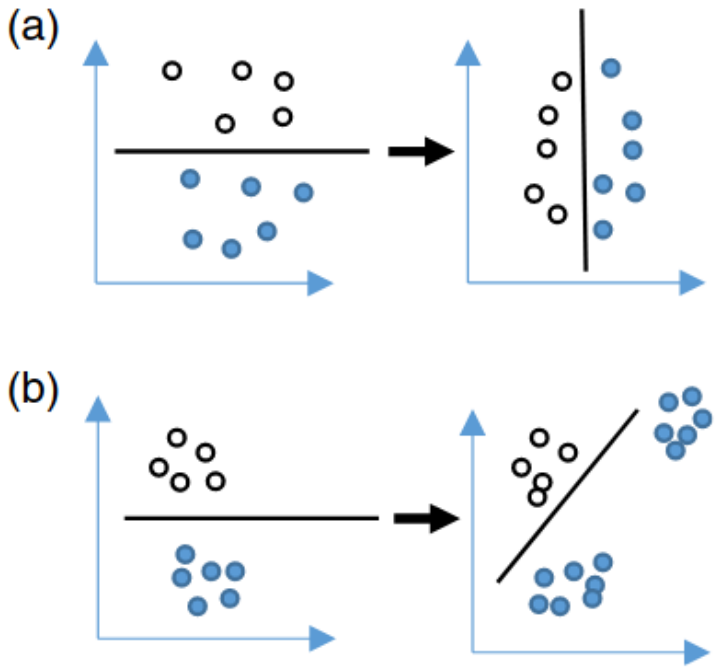



Table 2.1: A summary of drift types based on types of criteria [42, p. 5].

| **Detection Scope** | **Criteria** | **Type** |
|---|---|---|
| Single drift | Speed | Sudden |
| | | Gradual |
| | | Incremental |
| | Distribution | Fixed Space |
| | | Non-Fixed Space |
| | Recurrence | Recurrent |
| | | Non-recurrent |
| Drift sequence | Time Interval | Periodic |
| | | Irregular |

Figure 2.3: Categorization of concept drifts types to (a)fixed space drift. (b) non-fixed space drift [42, p. 6].




Examples of each category are listed in Table 2.2. Out of all these algorithms, KS has the advantage of drift detection for features with high dimensions using univariate ensembling [42][33]. Other drift detection methods for high-dimensional data and are distribution-based are the parametric detector (Hotelling), the non-parametric detector (Kullback-Leibler), and Semi-Parametric LogLikelihood detector (SPLL) [28]. However, the ensembling KS has been shown to outperform these mentioned algorithms [28]. Finally, there exist more distribution-based techniques that can identify suspicious samples such as enabling the model to have a label for "Unknown" classes. This however requires incorporating out-of-distribution samples into the training process [28].

## 2.5 Drift Detection in Deep Learning

Automatic drift detection for neural network-based learners slightly differs from classical machine learning learners. While training a machine learning model requires efficient feature pre-processing steps, deep learning models do these steps implicitly which makes them very powerful tools for high dimensional data such as images and texts. [60]

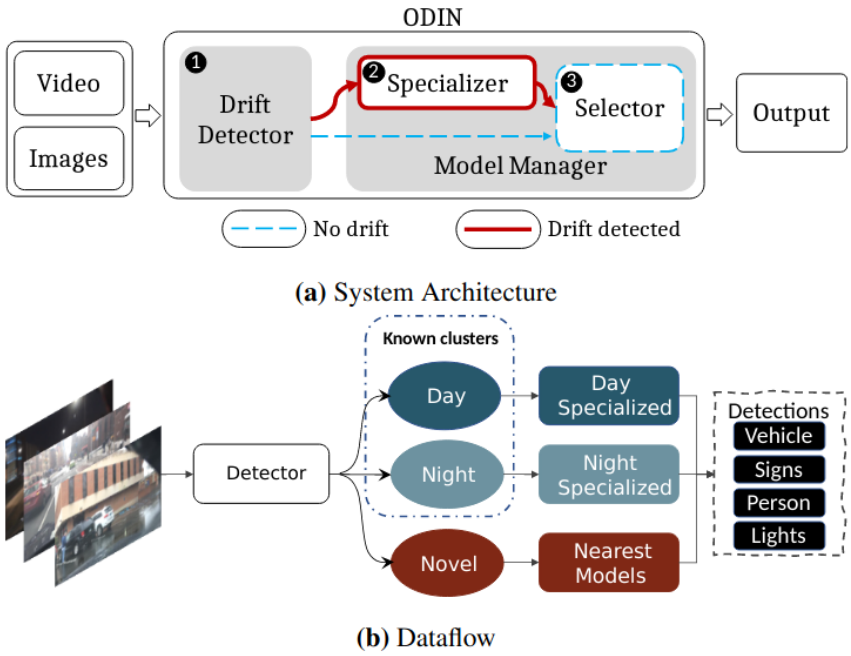
State-of-the-art approaches make use of the model-extracted features. Utilizing the extracted features for deep learning models has pushed recent advances in different fields such as computer vision, natural language processing and understanding, and reinforcement learning. However, the problem of drift

Table 2.2: Categorization of drift detection techniques with examples for each category [42].

| **Category** | **Algorithms** | **Reference** |
|---|---|---|
| Performance-based | Early Drift Detection Method (EDDM) | [7] |
| | Statistical Test of Equal Proportions (STEPD) | [62] |
| | Ensemble Classifier with Drift Detection (ECDD) | [55] |
| | Fisher's Exact Test | [87] |
| Ensembling-based | Adaptive-Size Hoeffding Tree | [12] |
| | Diversity for Dealing with Drifts (DDD) | [57] |
| | Knowledge Maximized Ensemble (KME) | [68] |
| | Restricted Boltzmann Machine (RBM) | [99] |
| | Online Histogram-Based Naïve Bayes Classifier (OHNBC) | [6] |
| Distribution-based | Kolmogorov–Smirnov (KS)\cite{razali2011power} | [37] |
| | Semi-supervised Adaptive Novel class detection (SAND) | [38] |
| | OnLine Novelty and Drift Detection Algorithm (OLINDDA) | [78] |
| | 1-norm Support Vector Machine (SVM) | [100] |
| | One Class Classifier | [100] |
| | Self Organizing Map (SOM) | [45] |

occurrences was inevitable in all these fields. Hence, ongoing research is being conducted to detect and overcome the drifts in testing phases and real-world cases such that the models mitigate the dropping of their performances. The underlining concept in drift detection for deep learning-based solutions is that the closed-world assumption does not apply. In addition, the training data does not entirely cover the latent distribution of the test data or real-world data. The focus of the state-of-the-art research is to embed the ability to overcome drifts within the model such that the performance remains within acceptable ranges. For example, the ODIN detector focuses on detecting drifts caused by changing weather conditions (e.g day, night) in which the known object would have unfamiliar features that cause to model's performance deterioration [79]. Using the ensembling of Generative Adversarial Networks (GAN), input models are clustered based on the produced features. If the produced features don't belong to the familiar clusters, then drift is detected. Figure 2.4 demonstrates the use of ODIN to detect drifts from the high dimensional images. Furthermore, Wang et al. have used Auto-Encoders to investigate prediction quality by using the encoder part to regenerate features that are passed to a multilayer perceptron for prediction [94]. The output is then passed to a decoder to forecast the quality of the prediction. Figure 2.5 shows the architecture of the model embedded with the Auto-Encoder. In addition to the mentioned solutions, uncertainty-based techniques have been investigated to detect concept drift under the condition of the true label being unavailable or scarce. According to Baier et al., Monte-Carlo Dropout is used such that the model can provide different prediction distributions in case of drifts [8]. Similarly, Monte-Carlo DropConnect is used to identify unknown inputs [93]. The difference between it and the Monte-Carlo Dropout is the architecture of the used neural network. In the Dropout case, the model's neurons are randomly turned off during predictions. However, in the DropConnect case, the weights are turned off during predictions. Finally, Deep Ensembling is also used technique to investigate concept drifts [48]. It is worth noting that in the training phase (or retraining in case an update is needed), it requires all ensemble models to be retrained as well.

Figure 2.4: Demonstration of ODIN. Based on the drift signal, a specializer determines which model ensemble fits the input best by clustering the drifting input [79, p. 3].




## 2.6 Concept Drift, Out-of-Distribution, and Anomaly Detection

While the challenge of drift detection is still an ongoing issue, it is being frequently studied. Several research studies cover a wide range of techniques that attempt to dominate solutions to identify drifting data. These studies approach the same problem yet with identifying the problem of detecting drifts within several research areas. In 2009, Chandola et al. [17] conveyed a survey to cover different techniques to approach solutions for anomaly detection. The survey also categorized the techniques based on the underlying principles with which each technique functions. According to Chandola et al. [17], the anomalies are defined as patterns in data that do not follow a defined normal behavior. From there, the survey demonstrated different contexts in which the problem of anomaly detection is investigated. As shown in Figure 2.6, anomaly detection has been also frequently studied in the field of machine learning, data mining, statistics, information theory, and more. Furthermore, different contexts characterize this problem such as the nature of data, the availability of labels, the type of anomaly, and how the output anomaly is described. In addition, the anomaly detection techniques can be categorized into classification-based, distance-based, Density-based, Clustering-based, and statistical-based techniques. Examples of each category are listed in Table 2.3. Moreover, with the motivation to find solutions for anomaly detection with high dimensional data and with less computational requirements, a survey was conducted by Thudumu et al. [83]. This survey considers problem characteristics similar to what was discussed by Chandola et al. [17] as shown in Figure 2.6. Nevertheless, it adds two more characteristics which are high-volume and high-velocity of

Figure 2.5: Model construction in quality characteristic prediction of WIP products [94, p. 9].

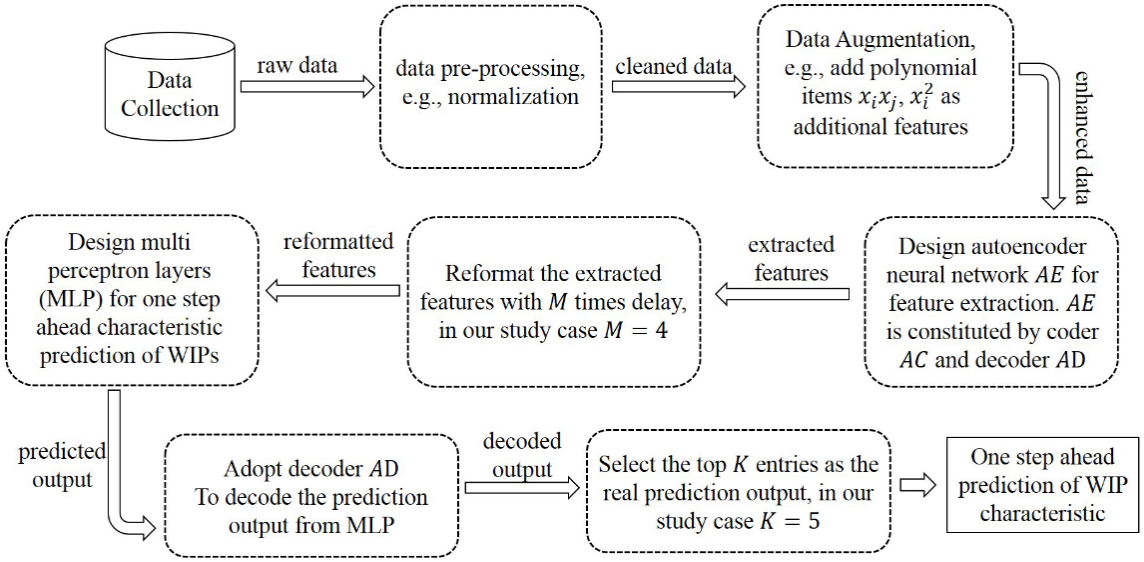
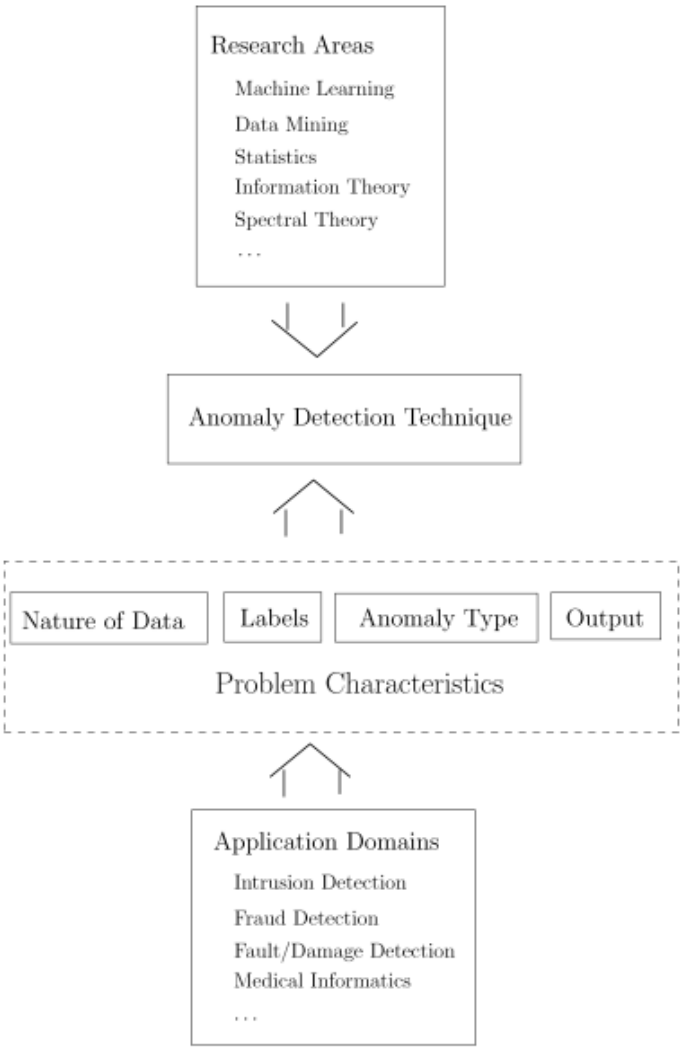



Table 2.3: Categorization of anomaly detection techniques with examples for each category [17].

| Category | Algorithms | References |
|---|---|---|
| Classifier-based | Support Vector Machines (SVM) | [88] |
| | One-class SVM | [75] |
| | One-Class Kernel Fisher Discriminants | [72] |
| | Rule-based Classifiers | [26], [40], [46], [73], [19] |
| | Neural Networks | [22], [56], [95] |
| | Bayesian Neural Networks | [9], [76], [23] |
| Distance-based | K-Nearest Neighbour | [16] |
| | Local Outlier Factor (LOF) | [15] |
| Clustering-based | K-means | [37] |
| | Expectation-Maximization (EM) | [59] |
| | DBSCAN | [27] |
| | Shared Nearest Neighnour (SNN) | [49] |
| | RObust Clustering using linKs (ROCK) | [36] |
| Statistical-based | Grubb's Test | [35] [5] |

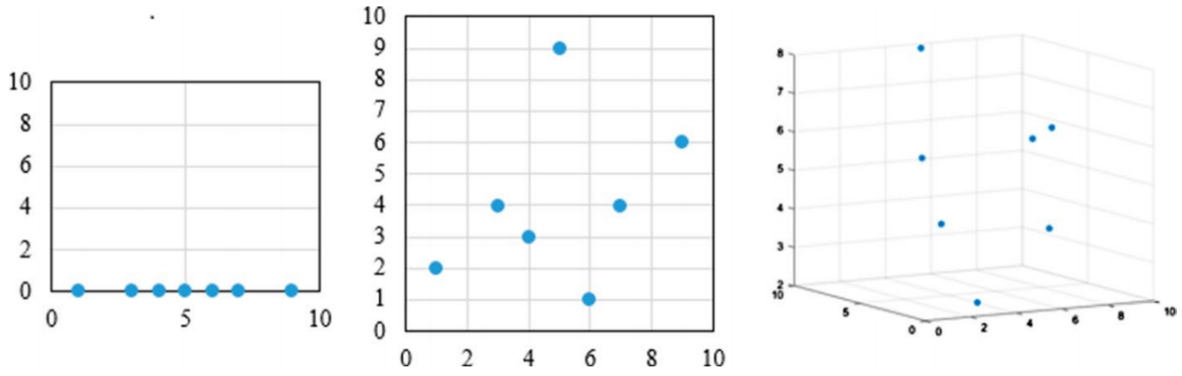
big data. According to [83], the high-volume refers to the amount of accumulated data, namely the size of the data. In addition, the high-velocity refers to the speed at which data are generated or added to the operation. [97] These two characteristics add constraints to the desired complexity of the anomaly detectors. This is due to the challenge imposed on the anomaly detection problem by the increasing dimensionality of data in the context of big data. Furthermore, a demonstration of the effect of increasing

Figure 2.6: The components involved with anomaly detection techniques [17, p. 4].



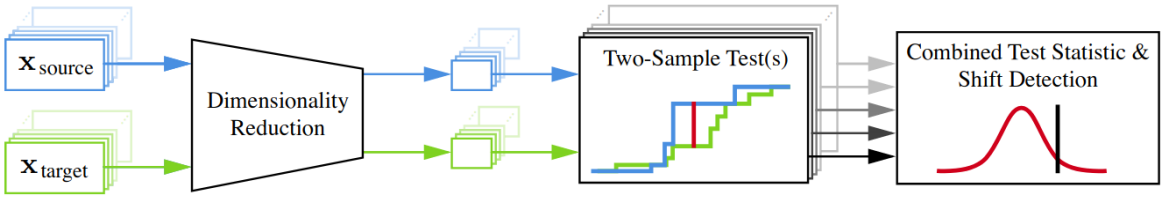

dimensionality on anomaly detection techniques is shown in Figure 2.7. This figure illustrates how the increasing dimensionality of the data causes sparsity of points when projected into one, two, and three dimensions. In addition, this sparsity causes a substantial number of false positives [83][4]. Another survey covered the same problem using the term out-of-distribution. According to Shafae et al. [77], the problem of identifying inputs that are not familiar to the training dataset is defined as out-of-distribution detection. In the survey conducted by Shafaei et al., the motivation for surveying out-of-distribution techniques was the concept that outliers come from a statistical distribution that is different from the training data's distribution [77]. Furthermore, it was stated that training a model to separate between

Figure 2.7: Projection of data points in one,two, and three dimensions respectively. [83, p. 8]




inliers and outliers can return optimistic results. Nevertheless, due to the absence of a standard definition of outliers, the use of a trained model for such purpose can be unreliable [4]. In their survey, they added uncertainty-based techniques including Monte Carlo Dropout and Deep Ensembling which were discussed in the previous section. Additionally, abstention-based techniques were added. In the abstention-based techniques, a function defined as a reject function is used. According to [11][20][21], this function could be a threshold for a prediction's magnitude or a threshold from a hypothesis test. While proposing their solution, it appeared that the Out-of-distribution Image Detection in Neural Network (ODIN) has outperformed their solution. Besides, ODIN has shown to be a powerful state-of-art technique that is based on modifying the magnitude of predictions such that it is below a pre-estimated threshold for out-of-distribution results [77][51]. Interestingly, the ODIN approach was compared against Maximum Mean Discrepancy (MMD) as a baseline for out-of-distribution detection. MMD has been also considered frequently in the field of domain adaptation and was used as a loss function [31]. Nonetheless, it has been included in a survey for drift detection for monitoring machine learning systems [66]. In this paper, the terms drift detection, anomaly detection, and out-of-distribution were used interchangeably. Besides, several solutions were covered in the paper as possible solutions to detect drifts. These algorithms were Maximum Mean Discrepancy (MMD), Kolmogorov-Smirnov (KS) Test, Chi-Squared Test, and Binomial Testing using a domain classifier. Also, it was shown that KS and MMD tests were powerful statistical techniques for high-dimensional data. Finally, the paper also discusses the effects of feature handling and dimension reduction before testing. Figure 2.8 shows the suggested pipeline for detecting drifts (anomalies). In the showed pipeline, two sets of samples are flown as input to the pipeline. The first input is the source set, while the second input is the target set. Both sets flow through the dimensionality reduction task. The lower-dimensional inputs then pass through a Two-Sample Test to implement hypothesis testing. From there, the combined test statistics and drift detection determine whether a drift is detected or not [66]. Distinguishing between drift detection, anomaly detection, and outlier detection in terms of definition seems to be problematic since the three terms have been studied widely in different fields with similar

Figure 2.8: Projection of data points in one,two, and three dimensions respectively [66, p. 2].

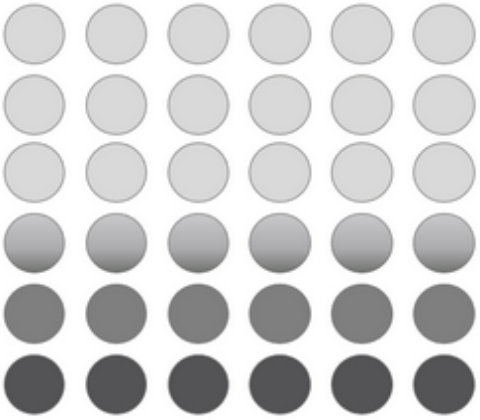



solutions as shown in [83][77][66]. However, there is a discussion that drift detection is different from anomaly detection, out-of-distribution detection, and outlier detection. It was discussed that while drift detection aims at detecting changes in the data distributions, outlier detection specifies whether a specific point lies far from a target distribution [74]. Figure 2.9 demonstrates the stated difference between drift detection and outlier detection. In this figure, the case of outliers is represented with the red background while the drift case was represented with white background. The example of the outliers shows them with two grey colors and they should be identified. In the case of drifts, the samples tend to get different values corresponding to having a changing distribution that needs to be detected. So the rising question will be, are drift detection and outlier detection contradicting? According to Huang et al. [43], they don't contradict each other. In particular, the drift detection methods perform as a high-level anomaly detector followed by outlier detection methods.

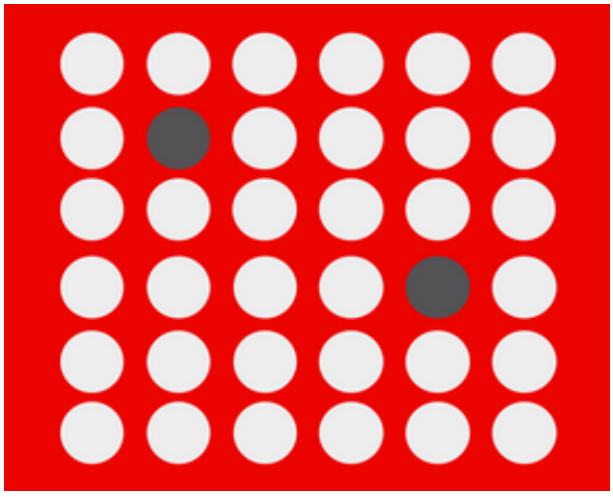
## 2.7 Which Algorithms to Select for Evaluation?

Following the proposed solutions by Rabanser et al. [66], the Kernel-based Maximum Mean Discrepancy and the Kolmogorov-Smirnov are selected for comparative analysis and evaluation on drift detection. Both algorithms have been shown to be able to detect drifts for high-dimensional features which enables to test them on different drift scenarios as to be shown in Section 3.8.2. Furthermore, implementing statistical hypothesis testing as in Figure 2.8 enables to provide justifiable answers. Namely, how the drift is detected. In addition, statistical hypothesis testing enables to test whether the test data and the reference data are independent and identically distributed (i.i.d) as usually assumed in training the desired model. Finally, the selected algorithms are to be run independently from the machine learning model in the pipeline which enables to analyze their performances separately from any model that can be used in the operational pipeline.

Figure 2.9: Difference between data drifts and outliers [74].



(a) Visualization of Drift occurrence for a set of samples.



(b) Visualization of outliers in a set of samples.

# 3

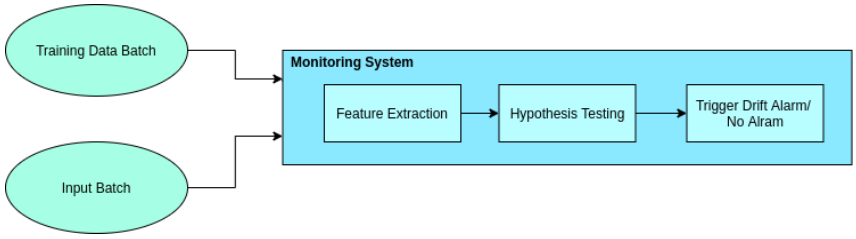
# Methodology

In this chapter, the methodological approach to evaluate drift detectors was defined. The defined approach intended to give a comprehensive explanation of how experiments were conducted. In particular, the techniques to select the elements of a drift detector were described. In addition, it was shown which kind of data was selected and why. Also, how training data were sampled and utilized for drift monitoring. Furthermore, the application of statistical hypothesis testing to detect drifts was demonstrated. Finally, the tools to develop the drift monitoring task for the experiments were specified.
As discussed in Chapter 2, there has been dense researches conducted on drift detection. These previous researches were implemented in a different context such as in data mining and machine learning. For machine learning and deep learning specifically, drift detection was investigated using different terms such as anomaly detection, and outlier detection. One of the similarities between the studied detection approaches was selecting or extracting the meaningful features from the input data, then implementing a detection algorithm for further decision making. The reason to select or extract features was to parse these features as inputs for the specified detection algorithm. As a first step for designing a monitoring task with a drift detector, an architecture to experiment with the pre-determined drift detection algorithms is demonstrated in Figure 3.1. The presented schema breaks down a monitoring process in a machine learning pipeline into three tasks. The three tasks were feature extraction, hypothesis testing, and drift alarm firing. The latter task was simplified as it mainly sends a message to other processes for decision-making regarding the modification of the machine learning model in the pipeline. The inputs in this schema are the training batch from the training dataset and a test input batch for drift evaluation. It should be noted that this schema was inspired by the proposed schema by Rabanser et al. as shown in Figure 2.8 [66]. The following sections explain the purpose of the proposed tasks in this schema and how they were utilized.

## 3.1 Sampling Training Data

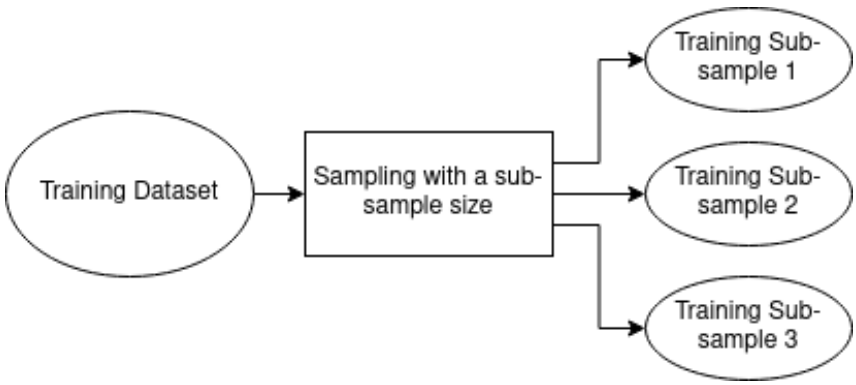
For drift detection, training data plays an essential role as reference data to determine whether the tested inputs are drifting or not in the statistical hypothesis testing step. Therefore, it is important to use the training data as a reference for drift monitoring. For the scope of this project, multiple sub-samples were randomly sampled and used iteratively as references. Hence, all drift detection experiments were frequently conducted for proper analysis. The reason to use sub-samples instead of the entire data-set was due to the effect of sample sizes in the performance of the hypothesis testing step as to be shown in the

Figure 3.1: A General schema for designing monitoring system for drift detection.




results. As a start, the sample size to conduct the experiments was 100. Then, an empirical evaluation was conducted to analyze the drift detectors' performances for each size. Figure 3.2 demonstrates how the training sub-samples were randomly sampled and utilized for drift detection experiments. The figure shows an example in which the training dataset is segmented into three sub-samples by randomly picking samples from the original dataset. Each sub-samples can be considered as a reference set or a test for drift evaluation based on the simulated scenario and the conducted experiment.

Figure 3.2: Demonstration of how training data were utilized for drift detection experiments.




## 3.2 Feature Extraction

Feature extraction is considered the first step in the proposed monitoring task. It has several benefits starting with reducing the high dimensions of raw input data. Also, feature extraction facilitates removing the redundant information in the data and using only the important information within the data [67]. The essence of feature extraction shows significantly in image, audio, and natural language processing [63]. There is a wide range of options from which feature extraction techniques can be selected. For example, one can select a filter-based method, an ensembler-based method, a linear method, or an embedding-based method [61]. Following the suggested approaches by Rabanser et al. in [66], the linear methods and embedding methods were selected for evaluating drift detectors. Namely the followings:

- Principle Component Analysis (PCA).

- Sparse Random Projection (SRP).
- Neural Networks as feature extractors.

The performance of the drift detectors was then compared based on the used feature extractor. It is worth noting that a combination of feature extraction techniques was also experimented to analyze the performances under different feature extraction techniques as will be seen later in the results. In the following, the used techniques are described.

### 3.2.1 Principle Component Analysis

The Principle Component Analysis (PCA) is a dimension reduction technique that is frequently used for processing high-dimensional data. PCA aims to extract the important information from the observed data. This is done by decomposing the data as a set of orthogonal variables also defined as the principal components [1]. The reduced dimensional data is represented as projections on these principle components [1][96]. The calculation of the orthogonal variables is made by using the Singular Value Decomposition (SVD). Given a set of high-dimensional data defined by the matrix $X$, the steps to implement PCA with SVD are as follows.

- Normalize the data by finding the mean of the matrix per each dimension, then subtract the mean from the matrix $X$.
- Decompose the normalized matrix into three matrices $(U, \Sigma, V^T)$. Namely, the normalized set $X$ becomes:
$$X = U\Sigma V^T \tag{3.1}$$
- The principle components correspond to columns of $V^T$. Therefore, select the number of components as required.

These selected components are the principal components. To obtain the desired reduced data, the data is projected on the selected principal components. It should be noted that the number of components determines the desired reduced number of dimensions. Also, to determine the desired number of components, the variance ratio should be investigated such that it remains high and acceptable. This corresponds to the case that the new representation doesn't cause the important information in the lower dimensional space to be lost [1][32][96].

### 3.2.2 Sparse Random Projection (SRP)

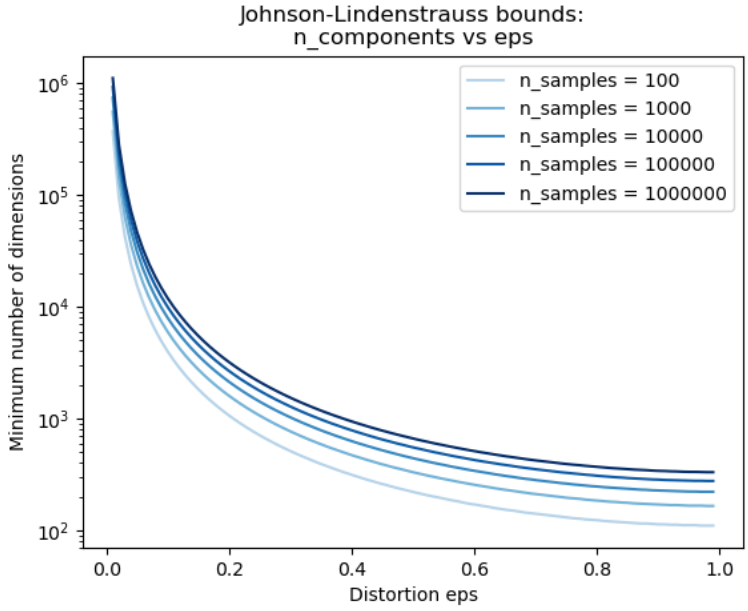
The Sparse Random Projection (SRP) is a dimension reduction technique that is based on the known Johnson-Lindenstrauss lemma [2]. Based on the Johnson-Lindenstrauss lemma, the high-dimensional data can be projected into lower dimensions in a random manner such that the distances between points are

preserved. It should be mentioned that the number of components $n$ is related to the error of distance perseverance $\epsilon$ given the number of dimensions in the original space $m$ as follows.[2]

$$n \geq \frac{4log(m)}{\epsilon^2/2 - \epsilon^3/3} \tag{3.2}$$

The Sparse Random Projection technique is favored when memory efficiency is required. This is due to the sparse projection matrix that is a concatenation of unit vectors in the $n$-dimensional space. [2] It is worth noting that the number of samples affects the relationship between the error and the number of components, such that the lower the number of samples, the higher the number of components needed to meet the desired error. Hence, before selecting the desired error, an investigation of the effect of the number of samples is needed. Figure 3.3 illustrates this relationship for different data sizes as experimented using the Scikit-learn's implementation of SRP [65]. From Figure 3.3, it can be seen that for a sample size of 100 (as selected in Section 3.1) a low error range from 0.0 to 0.4 correspond to the number of components from 100 to approximately 500. When evaluating drifts in the experiments, an error of 0.4 was arbitrarily selected.

Figure 3.3: Relationship between number of components and distance errors (By Scikit-learn [65]).

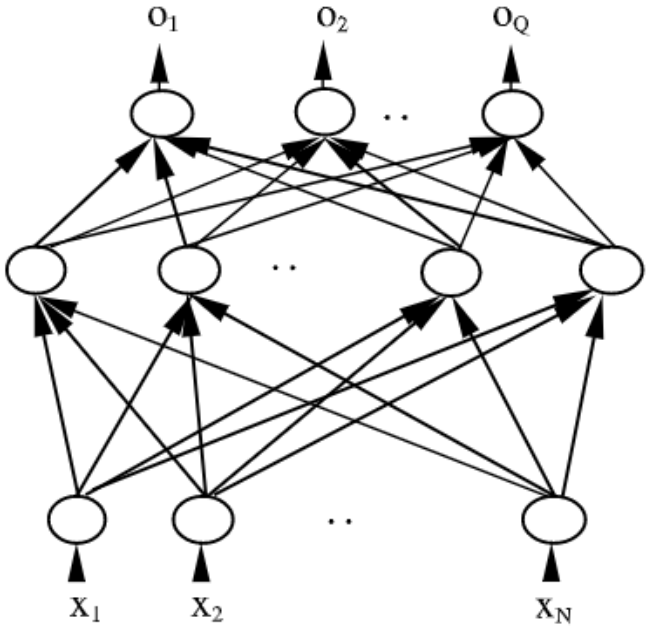



### 3.2.3 Neural Networks as Feature Extractors

The use of the neural network as feature extractors and dimension reducers has become a classical technique in machine learning models. As shown in Figure 3.4, the input flows into the neural network

via feedforward, and feature vectors are generated as outputs. Usually, these feature vectors are flowing into other hidden layers or into one last layer for label prediction. The design of the network depends on the application of the developed model. As discussed in Chapter 2, different drift detection techniques utilized the outcomes of the model's layers as the measured quantity for drift analysis.

Figure 3.4: Example of a neural network as a dimension reducer [89, p .2].




## 3.3 Two-Sample Hypothesis Test with Maximum Mean Discrepancy

Now that the techniques to extract features and reduce dimensions were explained. The next step is to discuss the hypothesis testing task in which the features are parsed as inputs and a decision whether there is a drift is determined as an output. The type of hypothesis testing to be conducted here is the Two-Sample test. A two-sample test is a statistical hypothesis testing mechanism that mainly focuses on assessing the similarity of the distributions from which the samples are observed [50]. This powerful technique has been adapted frequently in various applications. The most relevant application of the two-sample test is GANs as discussed in Chapter 2. In a Two-sample test, as in a standard hypothesis testing, two hypothesizes are defined. The first hypothesis is the Null hypothesis while the second one is the Alternative hypothesis. For the context of Two-sample tests, the Null hypothesis states that the observed two samples are drawn from the same probability distributions. In particular, these two samples are drawn from equal distributions. Let $P$ be the distribution from which the first sample is drawn while $Q$ is the distribution of the second sample. Then the Null hypothesis states that $P = Q$. In contrast, the Alternative hypothesis states that the observed two samples are drawn from different distributions. Hence,

$P \neq Q$. The decision of whether these two samples are drawn from the same distribution or not relies on rejecting the Null hypothesis. Namely, if the Null hypothesis is rejected, then the two samples are not coming from the same distribution and therefore, their distributions are different. Furthermore, to test if the Null hypothesis is rejected or not, a test statistic is needed to measure the difference between distributions being evaluated. The test statistic will then be the quantitative measure that needs to be selected such that a small value is resulted from measuring similar distributions and vice-versa. For the context of drift detection, the Two-sample test can be defined as follows. Given in the problem statement a subset of samples from training data set defined as $\{x_1, x_2, ..., x_n\} \sim p$ where $x$ is a sample from the subset, $p$ is the distribution of the training set, and $n$ is the number of data samples in the subset. Also given a subset of samples from incoming data set for prediction. This subset is defined as $\{x^{'}_1, x^{'}_2, ..., x^{'}_m\} \sim q$ where $x^{'}$ is a sample from this subset, $q$ is the distribution of the prediction set, and $m$ is the number of samples in this subset. In this context, the Null Hypothesis is defined as there is no drift between the training set and the prediction set [50]. Mathematically, it is taken as:

$$H_0 : p(x) = q(x^{'}) \tag{3.3}$$

Consequently, the alternative hypothesis is defined as follows.

$$H_1 : p(x) \neq q(x^{'}) \tag{3.4}$$

Now that the hypothesizes are defined, the problem of drift detection can be revisited and defined as follows. Given a model denoted by $f_t^M$, a training set $\{x_1, x_2, ..., x_n\} \sim p$ where $n$ is the number of data samples in the set. For an incoming data sets in production $\{x^{'}_1, x^{'}_2, ..., x^{'}_n\} \sim q$, the objective of the drift monitoring task is to enable the continually trained classifier to reject the Null Hypothesis $H_0 : p(x) = q(x^{'})$ when a covariate shift occurs after deployment. To be able to reject the Null hypothesis, the quantitative decision as a statistical test will become a function of the two sample sets.

$$T(X, X^{'}) : \{x_1, x_2, ..., x_n\} \times \{x^{'}_1, x^{'}_2, ..., x^{'}_m\} \longrightarrow R \tag{3.5}$$

To evaluate the hypothesis test, given that tests run on finite samples, it is essential to measure the statistical test results given true hypotheses. For this purpose, Type $I$ error and Type $II$ error are calculated. The Type $I$ error is defined as the probability of rejecting the null hypothesis given that it's true, namely the false-positive case. In contrast, Type $II$ error is the probability of accepting the null hypothesis given that it's not true or false negative. The p-value will be used to estimate the probability of correctness of the test statistics. In this context, the p-value is expressed as the probability of multiple test statistics measures taken at different situations ($T$) being greater or equal to a calculated threshold measure ($\widehat{T}$) as demonstrated in Equation 3.6. To calculate the measures $T$ and $\widehat{T}$, and estimate the p-value, the following steps are implemented.

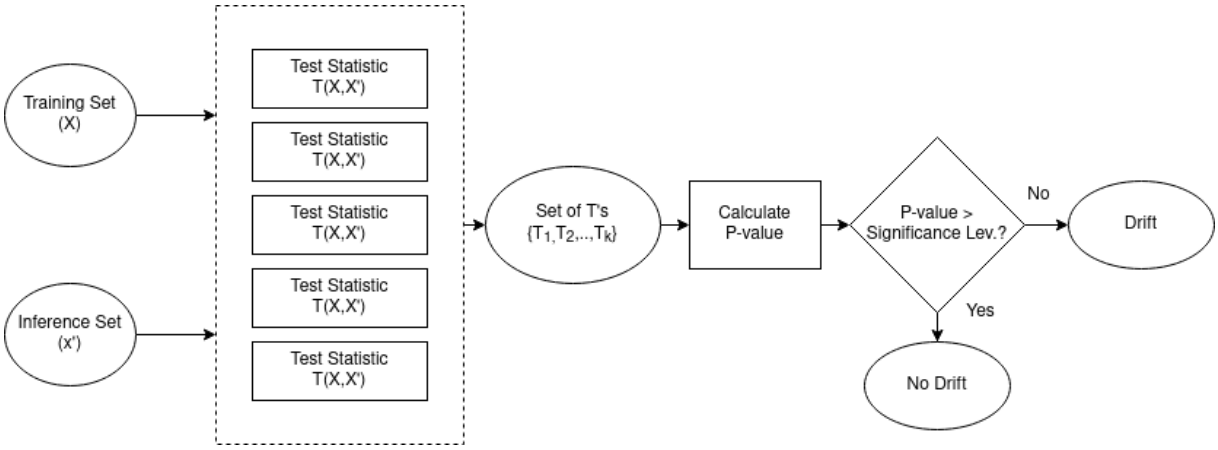
- The measure $T$ is estimated as the initial calculation taken by the test statistics.
- The set of measures $T$ will then be taken after manipulating the sample sets.

- After that, a probability of compliance with the null hypothesis is estimated, that is the p-value.
- This p-value is then compared to the significance level $\alpha$. The significance level determines whether the Null hypothesis is rejected or not.

$$p\text{-}value = P(T \geq \widehat{T}) \geq \alpha \tag{3.6}$$

Figure 3.5 summarizes the implementation of two-sample test to detect drifts [52][50][34]. Because of the relevance of this methodology to the problem of drift detection, the p-value was used as an indicator for evaluating this technique's ability to detect drifts. The next step was to investigate how Maximum Mean Discrepancy can be used as a test statistic.

Figure 3.5: Demonstration of how two-sample test was implemented to detect drifts.




## 3.4 Maximum Mean Discrepancy

The Maximum Mean Discrepancy (MMD) is a statistical distance metric that has been frequently used in different applications in machine learning as discussed in Chapter 2. As explained by Gretton et al. (2012) in [34], MMD measures similarity between distributions. Before understanding how MMD works, there are a few concepts that need to be briefly explained as a prerequisite. These concepts are as follows [34].

- A class of functions $\mathcal{F}$ exists such that $f : X \to \mathbb{R}$ and the expected values for $x$ and $y$ can be defined as $E_x[f(x)]$ and $E_y[f(y)]$ respectively.
- The empirical estimation of the expected value $E_x[f(x)] = \frac{1}{m}\sum_{i=1}^{m} f(x_i)$ where $m$ is the number of samples of $x$.
- The Reproducible Kernel Hilbert Space (RKHS) is a special class of functions $\mathcal{F}$ from which a function substitute the inner product $\langle f, \phi(x)\rangle$ where $\phi(x)$ is a feature mapping function.

- The mean embedding of a distribution $p$ defined as $\mu_p$ in RKHS corresponds to the expectation of functions $E_x f$. Namely, $E_x f = \langle f, \mu_p \rangle_{\mathcal{H}}$ for all $f \in \mathcal{H}$.

The next step is to properly define MMD. Given observations $X := \{x_1, x_2, ..., x_n\}$ and $Y := \{y_1, y_2, ..., y_n\}$ drawn from the distributions $p, q$ respectively, then a small distance is measured if the two distributions are equal and vice versa. In Lemma 3.4.1, which is discussed with it's proof in [34], this value is described as follows .

**Lemma 3.4.1** *Let ($\mathcal{X}$,d) be a metric space, and let $p, q$ be two Borel probability measures defined on X. Then $p = q$ if and only if $E_x(f(x)) = E_y(f(y))$ for all $f \in C(\mathcal{X})$, where C(X) is the space of bounded continuous functions on X.*

Based on Lemma 3.4.1, and because obtaining $p = q$ in practice is problematic when finite samples are used for testing, MMD is practically defined as in Equation 3.7 [34].

$$MMD[\mathcal{F}, p, q] := sup_{f \in \mathcal{F}}(E_{x \sim p}[f(x)] - E_{y \sim q}[f(y)]) \tag{3.7}$$

Furthermore, because the true distribution is unknown, empirical expectations are calculated such that Equation 3.7 becomes.

$$\text{MMD}[\mathcal{F}, p, q] := sup_{f \in \mathcal{F}}(\frac{1}{m}\sum_{i=1}^{m} f(x_i) - \frac{1}{n}\sum_{i=1}^{n} f(y_i)) \tag{3.8}$$

The next step is to define the class of functions that uniquely describes MMD when $p = q$. According to [34], the choice of the class F was the unit ball in a reproducing Hilbert kernel space $\mathcal{H}$. The reason this space is chosen is that the RKHS enables to generate continuous functions such that the calculated differences in the Hilbert space linearly reflect the differences in the original space $\mathcal{X}$. While this is useful to calculate the difference between two points in the Hilbert space, it is possible to implement the mapping from the original space to other spaces such as Banach space [34]. However, this will not be the point of focus in this project and the rest of the steps will be implemented in the Hilbert space following the study and implementation of Gretton et al. (2012) [34]. In RKHS, kernel functions are used to map the points from X to $H$ in probability distribution forms. Namely, the finite samples will be mapped into the Hilbert space as probability distributions. In particular, Gaussian kernels and Laplace kernels have proofed to suit best in this context due to their ability to generate dense functions in the Hilbert space and they are continuous. Namely, they have shown to be universal kernels [34]. For simplicity, only Gaussian kernels are to be used here. using kernel functions, it is possible to represent MMD now as follows. Where $\mu[p]$ and $\mu[q]$ are the mean embeddings of the distributions $p$ and $q$ in the Hilbert space $\mathcal{H}$ [34].

$$\text{MMD}[F, p, q] = ||\mu[p] - \mu[q]||_H \tag{3.9}$$

Now it is possible to calculate the maximum discrepancy by implementing the following Equation 3.10. The derivation of this equation is provided by Gretton et al. in [34]. For the scope of this work, this

equation was adapted as the test statistics for drift detection.

$$\text{MMD}[F, X, Y] = [\frac{1}{n}\sum_{i,j=1}^{n} k(x_i, x_j) - \frac{2}{nm}\sum_{i,j=1}^{n,m} k(x_i, y_j) + \frac{1}{m^2}\sum_{i,j=1}^{m} k(y_i, y_j)]^{\frac{1}{2}} \tag{3.10}$$

## 3.5 Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov (KS) test is a famous nonparametric test for continuous and discrete distributions. In this section, the concept of implementing KS for one-dimensional is firstly explained, then the implementation of KS for multivariate KS test is discussed. The idea for the one-dimensional KS test is as follows:

- First, sort the one-dimensional observations.
- Calculate the cumulative distribution function (c.d.f) of the ordered observations.
- Find the empirical distribution function of the ordered observation.
- Find the supremum of the difference between the empirical distribution function and the cumulative distribution function as shown in Equation 3.11.

$$D = \sup_x |F_n(X) - F(X)| \tag{3.11}$$

In Equation 3.11, $n$ is the number of the observations, $F_n(X)$ is the empirical distribution function, $F(X)$ is the cumulative distribution function. The supremum of the differences between these functions is called a distance or a discrepancy and is defined as $D$. When the observations are high multidimensional, then the multivariate Kolmogorov-Smirnov is implemented. For the scope of detecting drifts in a monitoring task, the reference set and the test set are high dimensional features with distributions that are unknown. Therefore, the needed test for drift should be a distribution-free test. For this reason, a solution that was based on Hodges's method is proposed [41]. Given a reference set of $m$ samples and a test set of $n$ samples, the implementation of the solution is as follows [41][90][82].

- This solution starts with combining the reference set and the test set as one set.
- Then, each element in this set is represented by its corresponding rank. Namely, it's order in the set. In python, this is implemented by the function $argsort()$.
- Sort the combined set based on the rank of the elements of the reference set and the test set.
- It should be mentioned that even though the two sets are combined, the origin of each element is known. Namely, it is known which set each element came from.
- For the values from the reference set, estimate an empirical distribution function $F_m(x)$ where $x$ is an element in the combined set. Each element in the set that has a rank below $m$ is assigned to a probability of $1/m$. Then the cumulative sum is found. This is implemented column-wise.

- Similarly for the elements from the test set, estimate an empirical distribution function $G_n(x)$ where $x$ is an element in the combined set. Each element in the set that has a rank below $n$ is assigned to a probability of $1/n$. Then the cumulative sum is found. This is also implemented column-wise.

- The distance $D$ is then calculated by finding the maximum distance between $F_m(x)$ and $G_n(x)$ as per Equation 3.12.

$$D = \sup_x |F_m(x) - G_n(x)| \tag{3.12}$$

The underlying concept in this approach is that, if the Null hypothesis is true, then the difference between the empirical distribution functions should be small. In particular, if $m = n$, then similar data points would have similar ranks, which in return result in having similar empirical distribution functions. From similar distribution functions, the resulting distance should be small. This approach has shown to be effective for drift detection and has been tested in different contexts in which anomalous events occur [41] [90][82]. After the distance is measured, the next step is to calculate the Type $II$ error, namely the p-value. It should be mentioned that Smirnov provided an equation to directly estimate the p-value (which was named $P_2$ in the original paper). The equation 3.13 estimates the p-value ($P_2$) for large $m$ and $n$.

$$P_2 = p(\sqrt{\frac{mn}{m+n}}D \geq x) \rightarrow 2\sum_{k=1}^{\infty}(-1)^{k-1}exp(-2k^2x^2) \tag{3.13}$$

Later, this equation was found to be inaccurate for large sizes of sets [41]. For this reason, an alternative solution was proposed. This solution utilizes a two-dimensional grid in which the x-axis represents the samples from the reference set and the y-axis represents the samples from the test set. On this grid, each point is represented by $(i/m, j/n)$ where $i, j$ are the measured ranks of elements for the reference set and test set earlier. The next step is to draw a path on this grid. The path starts from $(0, 0)$ then moves forward on the grid based on the sequence of the combined set up to $(m, n)$. For example, suppose that a sample set x has 6 observations ($m = 6$), and a sample set y that has 4 observations ($n = 4$). If the sequence of the combined set is $xyxyxxyyxx$, then the path starts from $(0, 0)$ then moves in the following direction: right, up, right, up, right, right, up, up, right, right. This is demonstrated in Figure 3.6. It is worth mentioning that the step sizes are $1/6$ for the $x$ and $1/4$ for $y$. According to [41], for each observation, namely for each step taken, the distance can be calculated representing the following equation.

$$F_m(x) - G_n(x) = i/m - j/n \tag{3.14}$$

After that, find the points that are the furthest from the diagonal ($Q^+$ below the diagonal and $Q$ above the diagonal). In this example, the distance for both $Q^+$ values were $1/6 - 0 = 1/6$ and $4/6 - 2/4 = 1/6$ above the diagonal. For $Q$ the distance is $4/6 - 1 = -1/3$ below the diagonal. This concept is adapted by [41] to calculate the p-value by using what's called the inner method. For an observed path, the following steps are implemented to calculate the p-value.

- Construct two boundaries by displacing the diagonal by the maximum distance calculated earlier as shown in Figure 3.7.

- Find the number of paths to go from $(0, 0)$ to $(m, n)$ while staying within the drawn boundaries. For each point on the grid, the number of paths to reach that point is expressed by $A(i, j)$ and is found by:

$$A(i, j) = A(i - 1, j) + A(i, j - 1) \tag{3.15}$$

- The starting points $A(0, y)$ and $A(x, 0) = 1$.

- The p-values becomes then the number of ways to go from $(0, 0)$ to $(m, n)$ but outside the diagonals. This is calculated as follows:

$$P_2 = 1 - A(m, n)/\binom{m+n}{n} \tag{3.16}$$

While this method seems quite complex, it has shown accuracy and efficiency in computation since this is used in Python's SciPy package [92]. Further improvements have been proposed by Viehmann to enable faster and stabler performance by normalizing the observations and reducing the need to implement random sequence generation which boosts the time performance in Equation 3.15 [90]. And the conditional value $C_{i,j}$ in a $m \times n$ grid becomes the desired p-value.

$$C_{i,j} = \begin{cases} 1 & \text{if} |i/m - j/n| \geq d, \\ 0 & \text{if} |i/m - j/n| \leq d \text{ and (i=0 or j=0)}, \\ C_{i-1,j}\frac{i}{i+j} + C_{i,j-1}\frac{j}{i+j} & \text{otherwise}, \end{cases} \tag{3.17}$$

Figure 3.6: Example of the grid representation for Hodges's method [41, p. 3].

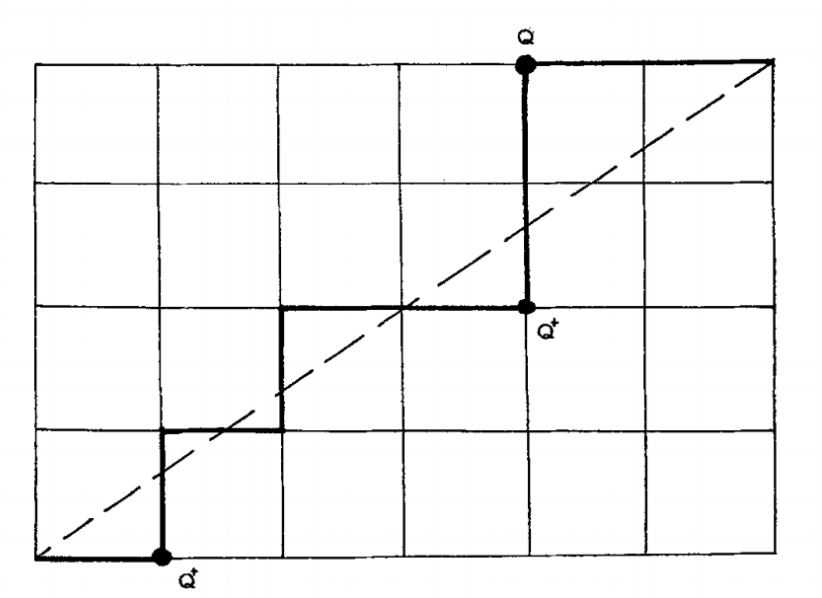
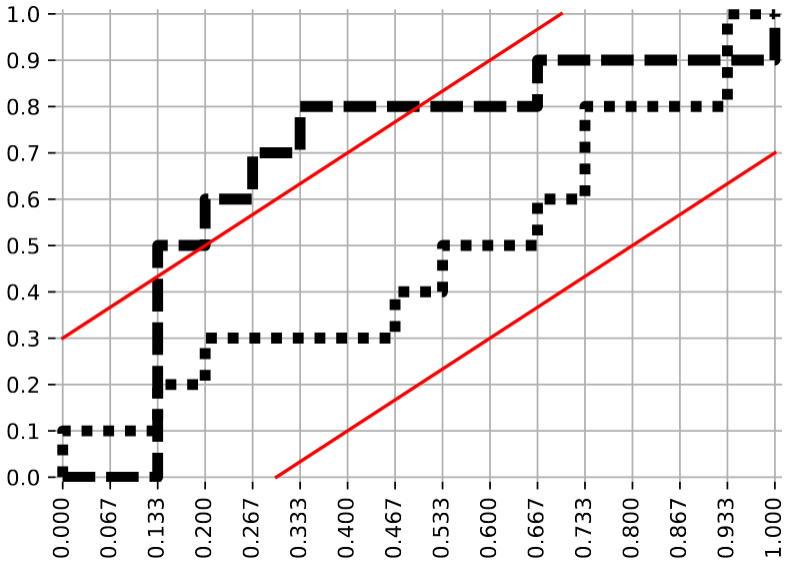


Figure 3.7: Example of the implementation of the inner method [90, p. 1].




## 3.6 Evaluation Methods

In order to carry out drift detection comparisons, a metric that is independent of the model's performance is needed. However, it should be mentioned that a correlation between the model's accuracy and the drift detectors' p-values was visualized and investigated. Furthermore, the Receiver Operator Characteristic (ROC) was used to analyze the performances of the drift detectors. ROC demonstrates a relation between the false-positive rates and the true-positive rates which are obtained from a model or a drift detector as in this case [29]. Here, the observations were collected from the drift detectors. The relation between the true-positive rates and the false-positive rates is visualized in a graph form. After running multiple tests with the prepared drifting data, the true and false-positive rates were calculated as follows.

$$True\ positive\ rate = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{3.18}$$

$$False\ positive\ rate = \frac{False\ Positives}{False\ Positives + True\ Negatives} \tag{3.19}$$

Finally, the Area Under the ROC Curve (AUC) was used as a single scalar representative of the detectors' performances. The reason to use AUC for comparison is its statistical property to reflect the probability that a drift detector will randomly detect drifting instances than randomly detect non-drifting instances. This is analogous to using AUC for classifiers as suggested by [29].

## 3.7 Case Study for Drift Evaluation

For the scope of this project, the evaluation of drift detection methods was done on image classification. In the following sections, a detailed explanation of the case studies is provided.

### 3.7.1 Image Classification

The classification task assigns labels to the input data based on a learned criterion. For the scope of high dimensional data, image datasets were selected as input types. In particular, Stream-51 dataset was used. Stream-51 is a large-scale image data-set that contains 51 labels. The images were sampled from videos of temporally correlated frames. The temporal correlation of images mimics how a machine learning model would perceive inputs in real life. This is useful if the model learns from these perceived inputs in an online or streaming fashion. Figure 3.8 shows examples of the labeled images along with temporal correlation of frames from which these images were sampled. Finally, this dataset is structured in a way that shows how to preprocess the images (normalizing and resizing) and how to create drift scenarios that facilitate constructing novel detection experiments [71]. This supports significantly for implementing the desired drift detection experiments.

Figure 3.8: Examples of labeled images from Stream-51 dataset [71, p. 4].

## 3.8 Experiments' Setup

The first step to run experiments on drift detectors was to build the experiment's environment. This environment was composed of the machine learning pipeline's monitoring task, drift data setup, and the experiment's phases.

### 3.8.1 The Monitoring Task

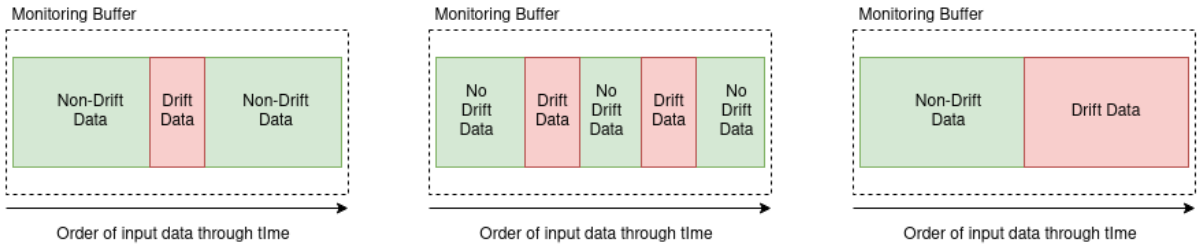
The experimental machine learning pipeline was minimal in design with the aim to focus on drift detection. As part of the pipeline, the monitoring task was designed in a pipeline form and consisted of the following inter-dependent components.

- Feature extractor
- Machine Learning model
- Drift detector

It should be noted that the inter-dependency of these components on each other can be utilized by the user based on the use-case under experimentation.

### 3.8.2 Setup of Drift Data

Simulating drift data for testing is challenging because it doesn't include all possible drifts in a real-world scenario. Here, drift data were prepared to meet the categorization of drifts in terms of occurrence through time into sudden, recurrent, and gradual. Figure 3.9 demonstrate how batches of images were prepared to meet these drifts cases. It should be mentioned that the ratio between drifting images and non-drifting images in the drift test is considered. The reason was to explore the detectors' performance given different ratios of drifting data in a drift test. Furthermore, the images were randomly



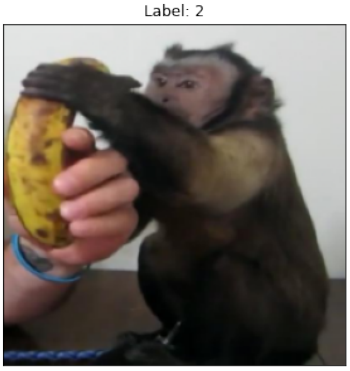

(a) Sudden Drift (b) Recurrent Drift (c) Gradual Drift

Figure 3.9: Demonstration of the three drift types to be tested.

selected based on their labels. Only a portion of the Stream-51 dataset was used here to include 5 classes as training data and 1 class as drifting data. Figure 3.10 shows examples of the classes used for the drift experiments. Finally, artificial environmental changes were included as drifts because they can affect a


(a) Known label 0 (train)

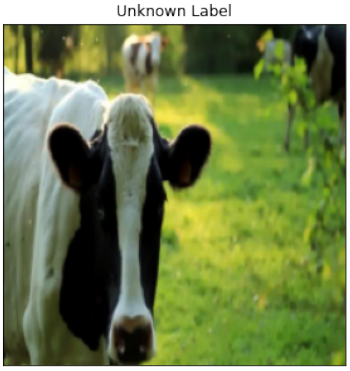

(b) Known label 1(elephant)




(c) Known label 2 (monkey)


(d) Known label 3 (horse)


(e) Known label 4 (zebra)




(f) **Unknown label (cow)**

Figure 3.10: Examples of test data including known labels and unknown labels.

model's performance negatively. Therefore, noisy images were tested for drift detection as well. Figure 3.11 shows noisy images with two severity levels.


(a) No Noise


(b) White Noise


(c) **Severe Blur**

Figure 3.11: Example of a known class with different levels of Gaussian noise.

### 3.8.3 Experiment Phases

After developing the machine learning pipeline and preparing drift test data, the experiments were implemented in three phases to facilitate the evaluation of drift detectors. Figure 3.12 demonstrates the three phases and the steps implemented towards proper evaluations.

- Initialization phase in which all the machine learning pipeline components were established.
- Drift detection testing phase in which all drifting batches of data flow into the pipeline to test drifts.
- Experimental evaluation phase in which the results were visualized and performances of the detectors were compared.

In the testing phase, the drift detectors were tested in two conditions, the ideal condition and the non-ideal condition. In the ideal condition, the monitoring buffer (see Section 4.3.1) would be filled either with drifting data or non-drifting data. In the non-ideal condition, the monitoring buffer would be filled with a mix of drifting and non-drifting data. The goal was to test the detectors' abilities to detect drifts that are embedded within non-drifting data.

## 3.9 Assumptions

It is worth mentioning that some assumptions were taken into consideration while running the predefined experiments. The following points demonstrate the assumptions.

- It was assumed that models used for image classification use-case were pre-trained. For the scope of this project, a model that was based on ResNet-18 is used due to its frequent usage for image classification and its easy availability [81].
- It was assumed that the used pre-trained model was robustly designed and trained such that in the case of no drifts, the model's accuracy remains high and drops only when drifting data are passed to it.
- The expected performance of the running machine learning model in the pipeline was to drop in a drift occurrence.
- This experimental pipeline focused mainly on the drift monitoring and detecting tasks. Hence, the structure of the pipeline regarding training and deploying the model is intended to be minimalist as shown in Figure 3.13.
- It is assumed that the ground truth labels were collected after the experimental model and the drift detectors function during an experiment, so it was not possible to investigate the model's accuracy to investigate drifts occurrences.

Figure 3.12: Structure of the experiments' three phases.

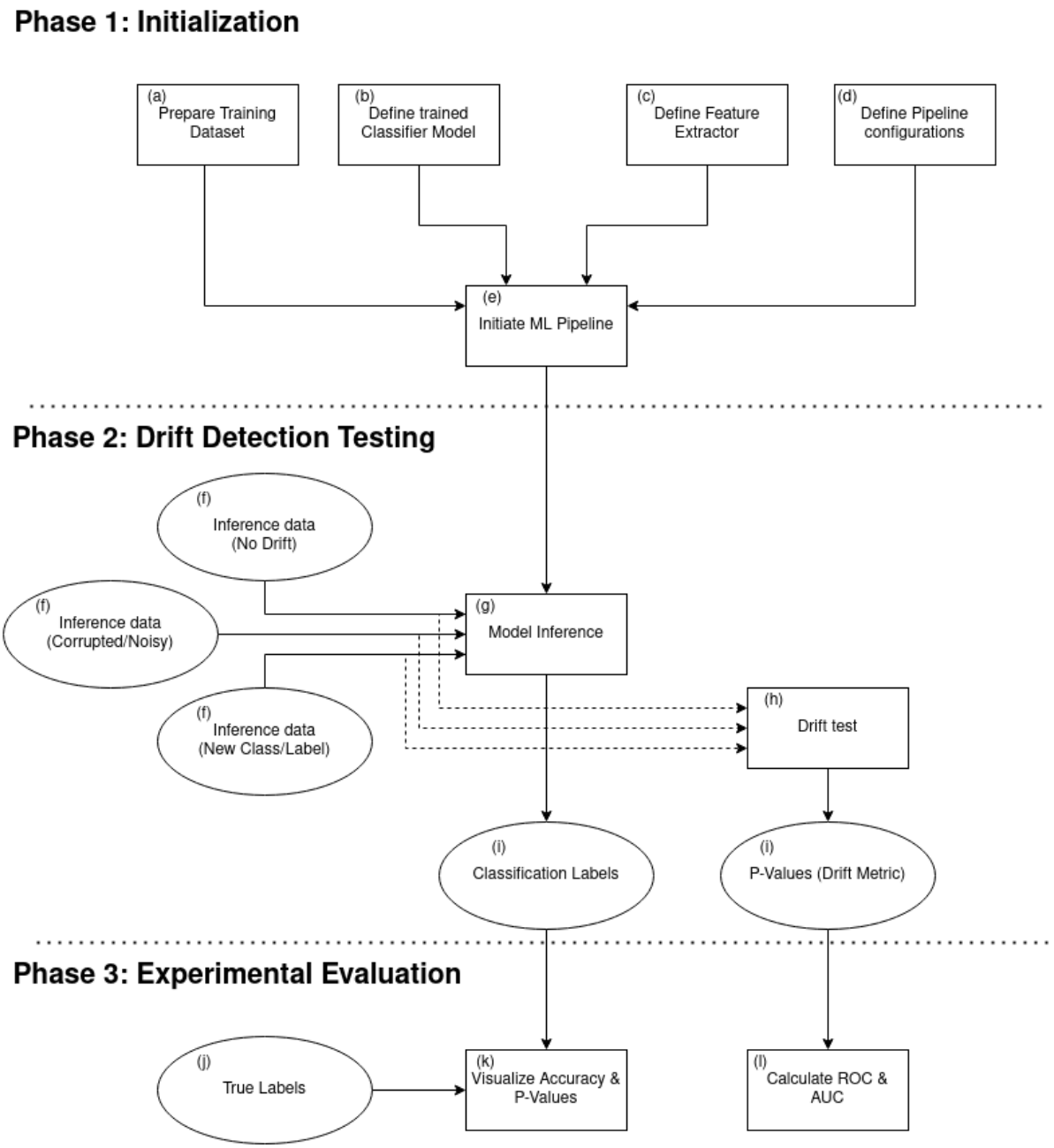
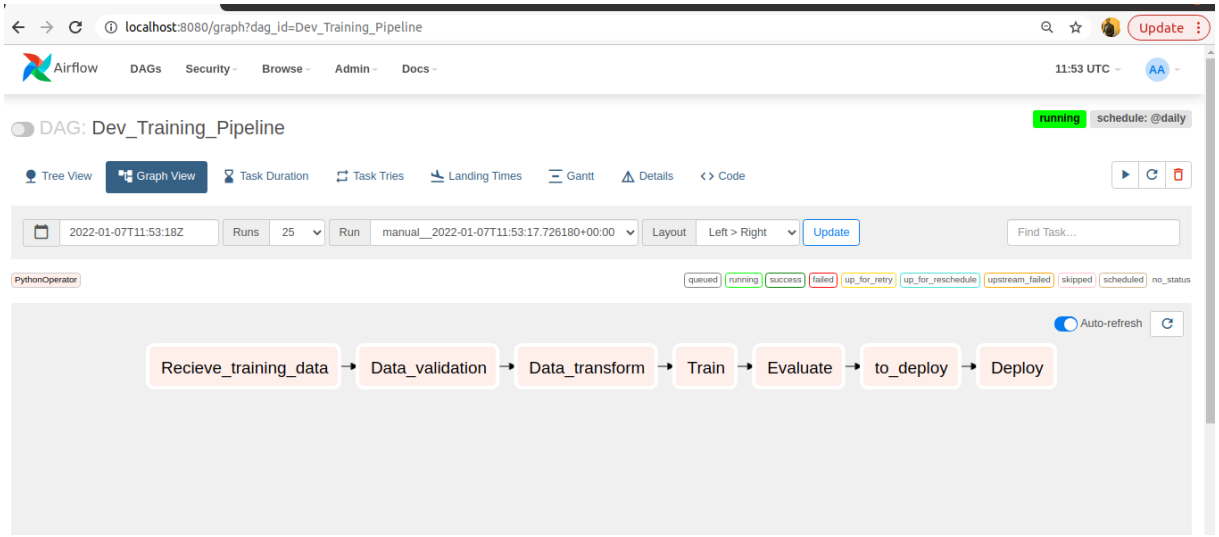


Figure 3.13: A minimal set of tasks to train and deploy a model as part of a machine learning pipeline.



# 4

# Construction of The Experimental Pipeline

In this chapter, the construction of the experimental pipeline, the drift detectors, and the evaluation tools are illustrated in-depth. It should be noted that the programming language used was Python 3.8 in Anaconda-based environment. Data processing, feature extractors, and drift detectors were built with Pytorch 1.8.1 framework.

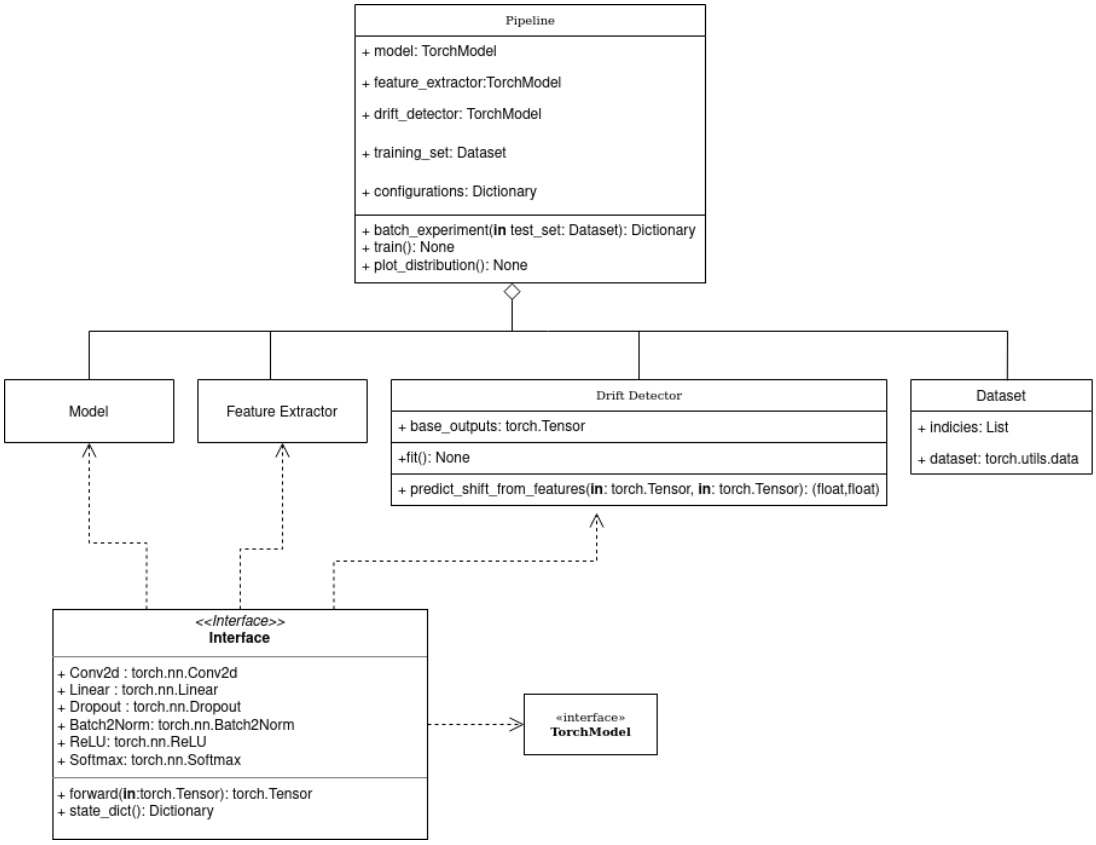
## 4.1 The Experimental Pipeline

The experimental pipeline was developed as a module that wraps the aggregated machine learning module along with the feature extractor and the drift detector, as shown in Figure 3.12. Figure 4.1 demonstrates the experimental pipeline architecture using Unified Modelling Language (UML). The model used in the experiments was a Resnet-18 that inherits from the Pytorch-based model architecture, hence the TorchModel type definition. Similarly, the feature extractor inherits from the Pytorch-based model architecture. It should be noted that the model, feature extractor, and detector were separately developed and later composed to form the pipeline. Although the feature extractor was considered as a separate entity in this pipeline architecture, the defined feature extractor consisted of the model's layers themselves. Furthermore, the configurations attribute was defined such that it included all the parameters needed to establish the pipeline properly. These defined parameters are the following.

- The batch size for training data for training.
- The batch size for test data for model inference.
- The monitoring buffer size in which the training and test data distributions were compared.
- The training model hyperparameters (learning rate, weight decay, number of epochs).
- Paths to store or load the model, feature extractor, and drift detector.

The pipeline class also included the *batchexperiment* method which was intended to run a loop for each batch of inputs. Three steps were run in each loop. The first step was the inference step which was done by passing the input batch into the model. Because this was an experimental pipeline, the ground truth was stored and used later for evaluation. The next step was to pass the same input batch into the feature extractor to generate feature vectors for each sample in the batch. The last step was to pass the stacked

Figure 4.1: UML of the Automated Machine Learning Pipeline for Drift experiments.

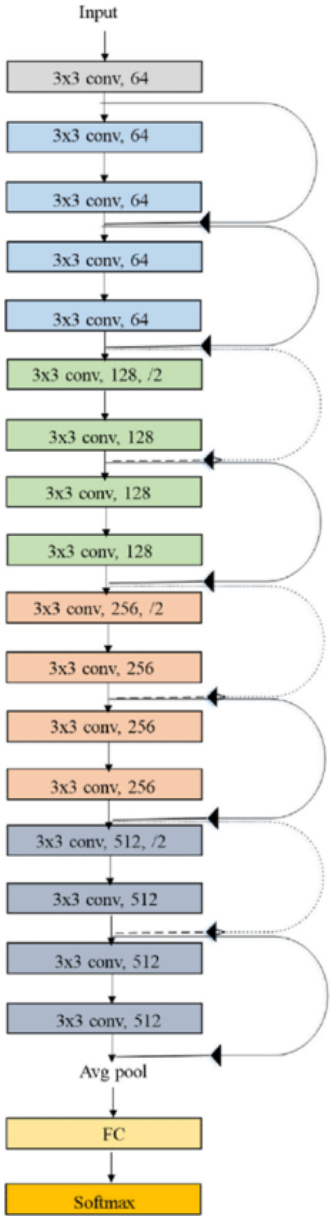



feature vectors into the drift detector to generate the drift distances and the p-values. The used model for the experiments was the ResNet model due to its extensive usage for image classification. The architecture of the ResNet model makes use of the residual blocks which boosts the model's performance. Figure 4.2 shows the architecture of the used ResNet model [39]. To enable this model to classify 51 labels from Stream-51, the FC layer of the ResNet model was substituted with a Linear layer of 51 units or neurons. Then, a Softmax activation function was used to generate a range of values between zero and one that was observed as probability values. Finally, the model was trained through transfer learning by applying backpropagation only on this last added layer. Not only this makes the training process fast, but also makes use of the weights obtained by training ResNet on the famous ImageNet dataset. This enables the model to be used also as a feature extractor as will be seen later in the results [85][64].

## 4.2 Feature Extractors

Multiple feature extractors were used for drift testing. As explained in Section 3.2, a feature extractor takes the raw high-dimensional images and extracts lower-dimensional feature vectors which are passed as inputs to the drift detectors. For the neural network feature extractors, the pre-trained ResNet model

Figure 4.2: ResNet Architecture for Image classification [39, p. 773].




was also used here. Furthermore, in later drift tests, the Untrained Auto-Encoder (UAE) was used. The architecture of this UAE consisted of convolutional layers followed by a multilayer perceptron. The idea was to investigate whether any nonlinear neural network would be able to function as a feature extractor or not. It should be noted that the number of dimensions for the generated features varied based on the type of the used extractor. Table 4.1 shows the number of dimensions of features per feature extractor. It

should be noted that the number of dimensions was 512 as per the architecture of the ResNet-18 model. For the SRP reducer, the resulted dimension size was due to the arbitrarily selected error (0.4) as discussed in Section 3.2.2. For PCA, the selection of 100 dimensions was due to the limitation that the number of samples was 100. Therefore, it was not possible to have a higher number of dimensions. Finally, for the UAE, the random design aimed to produce a low number of dimensions (32).

Table 4.1: Number of dimensions for the output features for each extractor.

| **Feature Extractor** | **Number of Dimensions** |
|---|---|
| ResNet-18 | 512 |
| ResNet-18 + SRP | 313 |
| ResNet-18 + PCA | 100 |
| UAE | 32 |

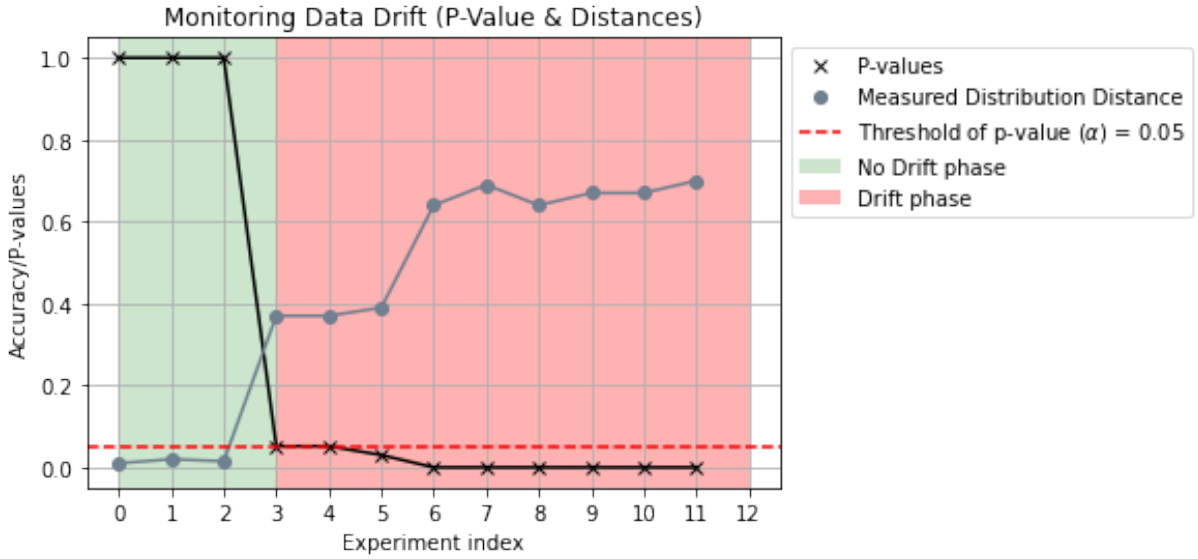
## 4.3 Drift Detectors

The drift detectors were designed and developed independently, then integrated into the experimental pipeline as seen in Figure 4.1. The structure of the drift detector is similar to the detectors developed in TorchDrift [82]. The reason to use TorchDrift's detectors was their ease of development for the developed torch-based machine learning pipeline, in addition to their ability to be integrated with a torch-based model by using hooks. This means that whenever an input passes through the model for inference, the input can automatically pass through the drift detector to calculate the p-values as a drift indicator. This can be seen in phase 2 in Figure 3.12. The drift detector consisted of the *base outputs* in which the processed feature vectors of the training dataset are stored as a tensor. This tensor was then used as a reference distribution for later comparison with the distribution of the test data. The method $fit()$ was the one responsible for collecting the training samples, processing them into feature vectors, stacking them into one tensor, and storing them in the predefined *base outputs* attributes. Finally, the outputs of the drift detector were the algorithm's distances metric (from MMD or KS), and the p-value as the drift indicator. For analyzing the performance of the drift detectors in the experimental pipeline, the output results are visualized in a way similar to Figure 4.3. This figure demonstrates the behavior of the drift detector in case no drifting data is flowing into the pipeline's model, and also in case drifting data such as in Figure 2.1 or Figure 3.11. It should be noted that the values of the distances and the p-values shown in Figure 4.3 are arbitrary and intend to give an expectation of how the result might look like the case of no drifts (with green background) and the cases of drifts (with red background).

### 4.3.1 Detector's Monitoring Buffer

It should be mentioned that before passing the test data for the drift detection test, they were accumulated in the memory buffer that was implicitly part of the monitoring task. In this buffer, the condition of the test data was set for testing, namely whether it was an ideal condition test or non-ideal condition test including recurrent, drift, and sudden drifts as explained in Section 3.8.3.

Figure 4.3: Expected behaviors of a drift detector during no drifts and drift phases.




## 4.4 Drift Alarm

The drift alarm is the sub-task that is activated once the drift detector identifies a drift. The alarm could be designed in any form that the alarm message would be delivered. For the purpose of simplicity, the drift alarm was designed such that once a drift is detected, it is logged in a report that is generated by the experimental pipeline. Furthermore, based on the logged alarm, the background of the visual graphs during drift analysis would be red in case of a drift alarm or green in case no alarm is logged. An example is illustrated in Figure 4.3.

## 4.5 Tools

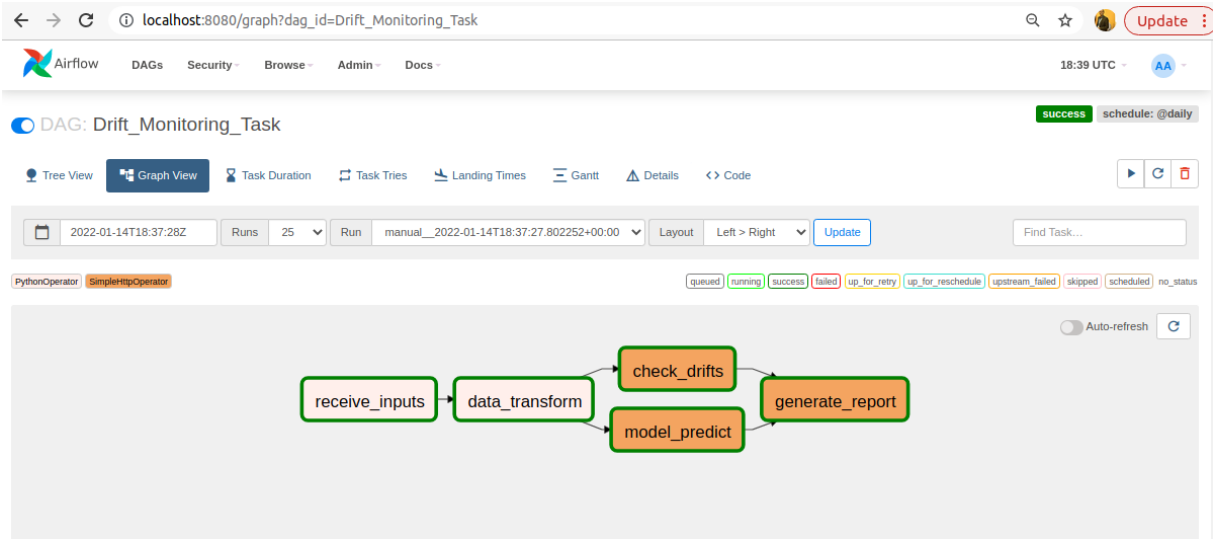
In this section, the tools with which experiments were conducted are listed as follows.

- This project was developed in Linux OS in an Anaconda environment.
- The programming language used was Python 3.8.
- The machine learning framework used was Pytorch 1.8.1. with the availability of a GPU and CUDA for fast model training.
- The development of the drift detectors was significantly facilitated by using Torchdrift [82].
- Docker was used for environment isolation and application shipping which enabled the code to be executable on any machine that has Docker installed in it.

- In addition, Apache Airflow was used to simulate a monitoring task as part of a machine learning pipeline in testing or production as shown in Figure 4.4.
- Finally, Jupyter Notebook was used for the purpose of performance analysis and evaluation.

It should be noted that some of the used tools were optional and had no effect on the performance of the drift detectors such as Jupyter Notebook, Airflow, and Docker. The Jupyter Notebook was used for visualization while Airflow and Docker were used to develop a simulation of a running monitoring task as part of a machine learning pipeline.

Figure 4.4: Drift monitoring task as part of a machine learning pipeline.



# 5

# Results

In this chapter, the outputs of the developed experimental pipelines are illustrated. The objective was to study the ability of the drift detectors to detect drifts, to understand the behavior of the detectors when drifts occur against when no drifts occur, and to compare the drift detectors with different feature extractors. The results of drift detection experiments were shown from the image classification use case by using Stream-51 dataset as explained in Section 3.7. The experiments were conducted as described in Section 3.8.3. The explanation of the results is further discussed in Chapter 6. It should be mentioned that for the image classification use case, the configurations in Table 5.1 were fixed for simplifying the experiments and making the analysis of the results feasible in the scope of this work. This table breaks down the configurations into 3 categories: Model, Training or Reference Set, and Drift Detector. The configurations for the model were intended to be minimalist and only included the learning rate, the number of epochs, and whether this model is trainable or not. The configurations for the training set were the ones responsible for generating drift phases or scenarios. In this set of configurations, the Stream-51 dataset was split based on the number of classes it contains (51 classes) , into 3 groups. The first group contained 5 classes, the second group contained 1 class, and the rest of the classes were included in the last group. The idea was that each group simulates a scenario such that if the first group of classes was used for training, then the second and third groups would be considered as drifting classes or scenarios and the drift detector should identify them as drifting data. Therefore, the first group was selected for training as indicated by the (Training classes index) in Table 5.1. Furthermore, the set of the configuration for the drift detector included the type of detector to be tested (e.g MMD), the number of reference samples used as a reference distribution, and the number of samples from inference data for drift testing.

## 5.1 Drift Detection for Image Classification

For the predefined Image Classification use case, the experimental machine learning pipeline was initiated such that non-drifting images (images from validation set) were flown through the pipeline. While the model's accuracy was expected to remain high, the drift detectors were tested against this kind of input flow. In the next step, the input images were corrupted with white noise and flown through the pipeline. The model's accuracy was expected to remain high (assuming robustness of the model in Section 3.9), and the detectors' performances were observed again. The same procedure was followed twice, once with corrupting the images with severe noise as in Figure 3.11c, and once again with images

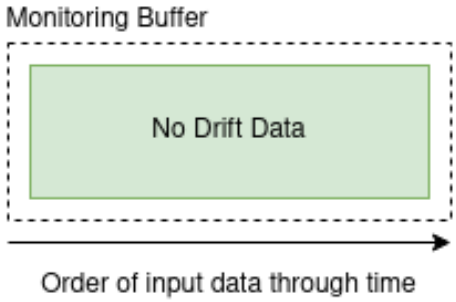
Table 5.1: The initial configuration settings for drift testing.

| **Configuration Tree (Dictionary)** | **Key** | **Value** |
|---|---|---|
| Model | Learning Rate | 0.001 |
| | Epochs | 5 |
| | Train | True |
| Training (Reference) Set | Dataset | Stream-51 |
| | Number of Classes | 51 |
| | Distribute classes | [5, 1, 45] |
| | Training classes index | 0 |
| Drift Detector | Reference Sample Size | 100 |
| | Monitor Buffer Size | 100 |

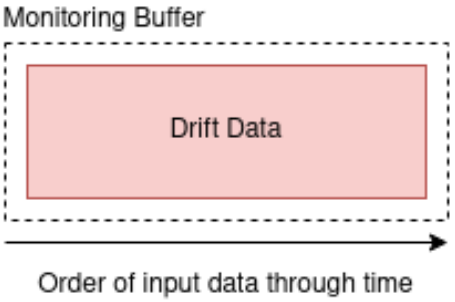
that contained novel classes as in Figure 3.10f as shown in Chapter 3. Furthermore, the experiments were conducted one time in ideal conditions, and another time in non-ideal conditions. In the following, these conditions along with their results are shown.

### 5.1.1 Ideal Conditions for Drift Experiments

In this set of experiments, the monitor buffer in the developed pipeline was occupied with either a fully non-drifting batch in the phases of no drifts, or a fully drifting batch in the phases of experiments where drifting data flows into the buffer. This is shown in Figure 5.1 and it is defined as an ideal experimental condition.




(a) Monitoring Buffer filled with no drift samples only.

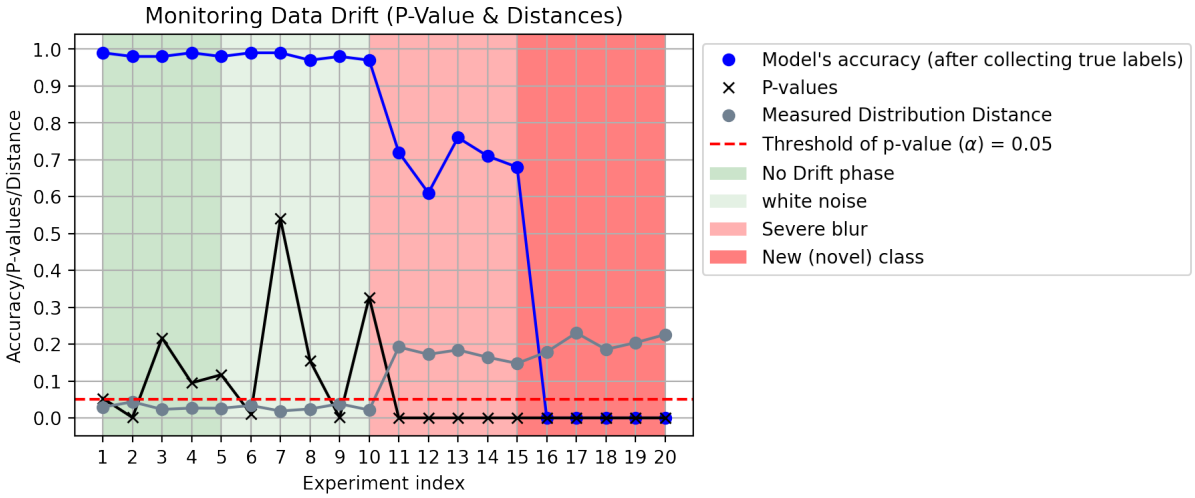



(b) Monitoring Buffer filled with drift samples only.

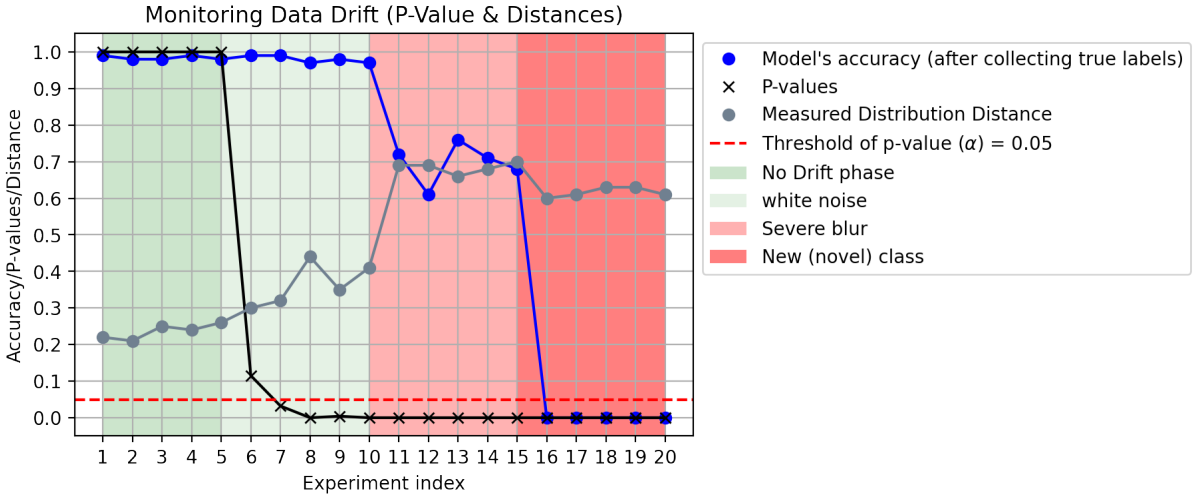
Figure 5.1: Demonstration of the monitoring buffer during experiments in ideal conditions.

In Figure 5.2, the performances of the model and the drift detectors are illustrated. For the MMD and KS drift detectors, Figures 5.2a and 5.2b demonstrate the model's accuracy along with the measured distances and the p-values estimated by the detectors. The x-axis shows the index of the conducted experiment, and the y-axis shows the values of the model's accuracy, the detector's distances, and p-values. The experiment indices were categorized into 4 phases based on the type of input data. The first phase

(with a green background) consisted of 5 experiments, each with different non-drifting data. The second phase (with a light green background) included 5 experiments, each with non-drifting images but with white noise. The third phase contained 5 experiments, each with images corrupted by severe blur. The last phase covered 5 experiments, each set of input images contained a novel class. It should be mentioned that the reference set remained without changes.




(a) Distances, p-values of MMD detector in ideal experimental conditions.




(b) Distances, p-values of KS detector in ideal experimental conditions.

Figure 5.2: Demonstration of the monitoring buffer during initial experiments.

Starting with the MMD detector in the No Drift phase, The results are demonstrated in Table 5.2. From the table, the scored distances along the 5 experiments were (0.029, 0.044, 0.023, 0.027, 0.026), the scored p-values were (0.053, 0.001, 0.216, 0.094, 0.116), and the accuracy of the model for each experiment

was (0.980, 0.980, 0.990, 0.980, 0.980). There, the p-values below the significance level are shown in red. Thus, it is shown that experiment 2 was considered a drift case by the MMD detector given that this was the No Drift phase. In the second phase, namely the white noise phase from index 6 to 10, the observed distances were (0.033, 0.019, 0.024, 0.038, 0.021). The observed p-values were (0.011, 0.540, 0.740, 0.154, 0.017), and the model accuracies were (1.000, 0.980, 0.155, 0.001, 0.327). From Table 5.2, it is shown that experiments 2 and 6 were considered drifts by the MMD detector. In the third phase, namely the drift phase with severely blurred images, the measured distances were (0.192, 0.172, 0.184, 0.164, 0.148) while the p-values dropped for each experiment to become (0.000, 0.000, 0.000, 0.000, 0.000). It should be noted that the model's accuracy during the 5 experiments has dropped for each experiment as follows (0.689, 0.680, 0.740, 0.670, 0.660). Table 5.2 shows that all experiments from 11 to 15 were considered drift cases since the p-values were all below the significance threshold. For the last phase, namely the novel class phase from 16 to 20, the measured distances were (0.179, 0.230, 0.186, 0.204, 0.226). The p-values and the model's accuracy has drastically dropped for each experiment to become (0.000, 0.000, 0.000, 0.000, 0.000).

Table 5.2: Results from the MMD drift detector under testing in ideal experimental conditions.

| **Experiment Index** | **MMD Distance** | **P-value** | **Model's Accuracy** |
|---|---|---|---|
| 1 | 0.029 | 0.053 | 0.98 |
| 2 | 0.044 | 0.001 | 0.98 |
| 3 | 0.023 | 0.217 | 0.98 |
| 4 | 0.027 | 0.095 | 0.98 |
| 5 | 0.026 | 0.117 | 0.98 |
| 6 | 0.033 | 0.011 | 1.00 |
| 7 | 0.019 | 0.540 | 0.98 |
| 8 | 0.024 | 0.155 | 0.98 |
| 9 | 0.038 | 0.001 | 0.98 |
| 10 | 0.021 | 0.327 | 0.98 |
| 11 | 0.192 | 0.000 | 0.689 |
| 12 | 0.172 | 0.000 | 0.680 |
| 13 | 0.184 | 0.000 | 0.740 |
| 14 | 0.164 | 0.000 | 0.670 |
| 15 | 0.148 | 0.000 | 0.660 |
| 16 | 0.179 | 0.000 | 0.000 |
| 17 | 0.231 | 0.000 | 0.000 |
| 18 | 0.186 | 0.000 | 0.000 |
| 19 | 0.204 | 0.000 | 0.000 |
| 20 | 0.226 | 0.000 | 0.000 |

The same 20 experiments were implemented against the KS detector. The results of these experiments are shown in Figure 5.2b and Table 5.3. For the first phase (No Drift), the table shows that the distances and p-values were (0.220, 0.210, 0.250, 0.240, 0.260) and (1.000, 1.000, 1.000, 1.000, 1.000) respectively. It is worth noting that no drift case alarm was fired here as indicated by the table in red. For the second

phase with white noise, the distances have increased for each experiment to be (0.300, 0.312, 0.440, 0.350, 0.410) causing the p-values to drop to (0.115, 0.032, 0.000, 0.004, 0.000). It should be mentioned that the experiments 7, 8, 9, and 10 were considered drifts by the detector as shown in Table 5.3. For the third phase with severe blur, the resulted distances and p-values were (0.689, 0.689, 0.740, 0.670, 0.660) and (0.000, 0.000, 0.000, 0.000, 0.000) respectively. Here, all the experiments from 11 to 15 were considered drift cases as shown in Table 5.3. For the last phase, the distances resulted to be (0.600, 0.610, 0.630, 0.630, 0.610) and the p-values have drastically dropped to (0.000, 0.000, 0.000, 0.000, 0.000).

Table 5.3: Results from the KS drift detector under testing in ideal experimental conditions.

| **Experiment Index** | **KS Distance** | **P-value** | **Model's Accuracy** |
|---|---|---|---|
| 1 | 0.220 | 1.000 | 0.98 |
| 2 | 0.210 | 1.000 | 0.98 |
| 3 | 0.250 | 1.000 | 0.98 |
| 4 | 0.240 | 1.000 | 0.98 |
| 5 | 0.260 | 1.000 | 0.98 |
| 6 | 0.300 | 0.115 | 1.00 |
| 7 | 0.312 | 0.032 | 0.98 |
| 8 | 0.440 | 0.000 | 0.98 |
| 9 | 0.350 | 0.004 | 0.98 |
| 10 | 0.410 | 0.000 | 0.98 |
| 11 | 0.690 | 0.000 | 0.689 |
| 12 | 0.690 | 0.000 | 0.680 |
| 13 | 0.660 | 0.000 | 0.740 |
| 14 | 0.680 | 0.000 | 0.670 |
| 15 | 0.700 | 0.000 | 0.660 |
| 16 | 0.600 | 0.000 | 0.000 |
| 17 | 0.610 | 0.000 | 0.000 |
| 18 | 0.630 | 0.000 | 0.000 |
| 19 | 0.630 | 0.000 | 0.000 |
| 20 | 0.610 | 0.000 | 0.000 |

### 5.1.2 Overcoming False-Positives with Sub-Sampling Reference Set

An additional experiment was added to mitigate false alarms that occurred during the No Drift phase and the White Noise phase. An attempt was implemented by dividing the 200 samples into 2 sub-sets each with 100 samples. Hypothesis tests were implemented independently and the final distances and p-values were averaged. This attempt is made on experiments 1 to 10. Figure 5.3a and Table 5.4 show the results of the MMD detector for experiments 1 to 5 (No Drift phase). From the table, the scored p-values were (0.376, 0.257, 0.650, 0.517, 0.412) corresponding to the distances (0.042, 0.051, 0.036, 0.039, 0.041). Then for experiments 6 to 10 (White Noise), the p-values were (0.525, 0.294, 0.333, 0.446, 0.461) and the distances were (0.041, 0.045, 0.053, 0.042, 0.047). Similarly, the experiments were conducted using KS detector. Figure 5.3b and Table 5.5 show the results of the KS detector. From the table, the p-values and

distances for the no drift phase were (1.000, 1.000, 1.000, 1.000, 1.000, 1.000) and (0.260, 0.300, 0.260, 0.300, 0.310) respectively. Finally, for the white noise phase, the p-values and distances were (0.673, 0.300, 1.000, 1.000, 0.126) and (0.370, 0.400, 0.320, 0.330, 0.420) respectively.

Table 5.4: Results from the MMD drift detector in ideal conditions using Sub-sampling.

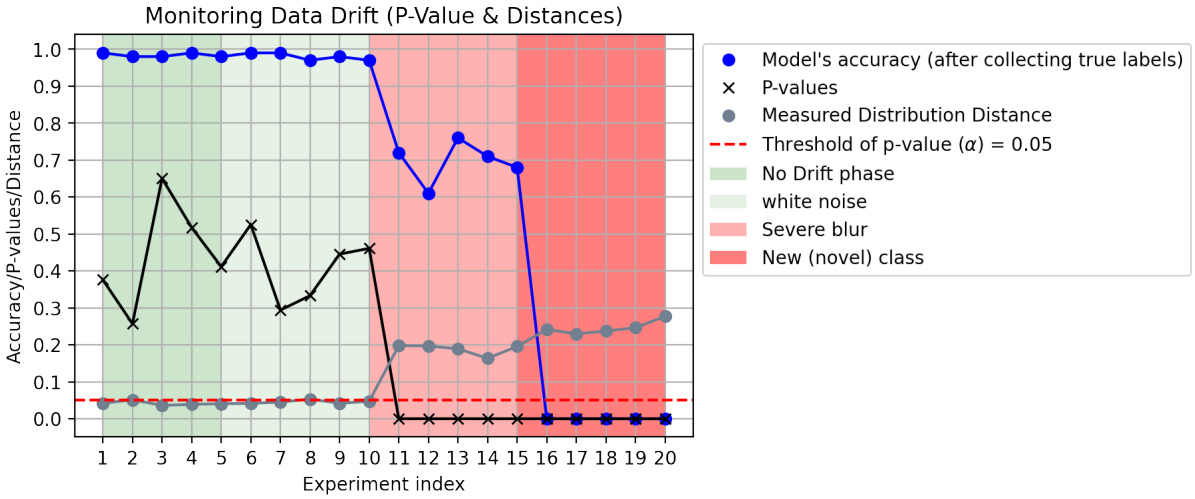
| **Experiment Index** | **MMD Distance** | **P-value** | **Model's Accuracy** |
|---|---|---|---|
| 1 | 0.042 | 0.376 | 0.98 |
| 2 | 0.051 | 0.257 | 0.98 |
| 3 | 0.036 | 0.650 | 0.98 |
| 4 | 0.039 | 0.517 | 0.98 |
| 5 | 0.041 | 0.412 | 0.98 |
| 6 | 0.041 | 0.525 | 1.00 |
| 7 | 0.045 | 0.294 | 0.98 |
| 8 | 0.053 | 0.333 | 0.98 |
| 9 | 0.042 | 0.446 | 0.98 |
| 10 | 0.047 | 0.461 | 0.98 |

Table 5.5: Results from the KS drift detector in ideal conditions using Sub-sampling.

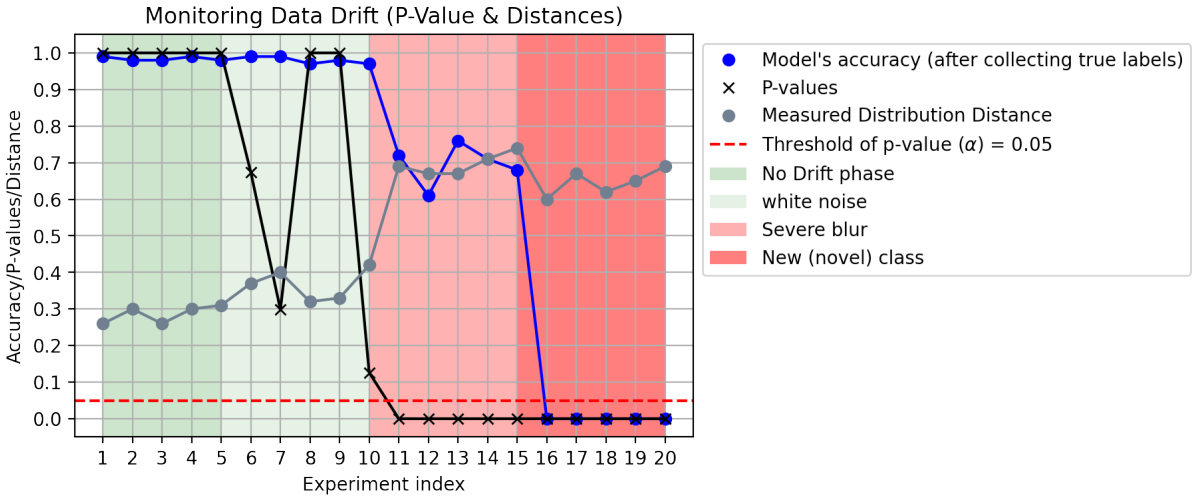
| **Experiment Index** | **KS Distance** | **P-value** | **Model's Accuracy** |
|---|---|---|---|
| 1 | 0.260 | 1.000 | 0.98 |
| 2 | 0.300 | 1.000 | 0.98 |
| 3 | 0.260 | 1.000 | 0.98 |
| 4 | 0.300 | 1.000 | 0.98 |
| 5 | 0.310 | 1.000 | 0.98 |
| 6 | 0.370 | 0.673 | 1.00 |
| 7 | 0.400 | 0.300 | 0.98 |
| 8 | 0.320 | 1.000 | 0.98 |
| 9 | 0.330 | 1.000 | 0.98 |
| 10 | 0.420 | 0.126 | 0.98 |

### 5.1.3 Non-Ideal Conditions for Drift Experiments

In this set of experiments, the monitoring buffer consists of a mixture of drifting and non-drifting images for each experiment. The mixtures were in recurrent order as shown in Figure 3.9b, a gradual drift as in Figure 3.9c, or a sudden drift as illustrated in Figure 3.9a. The performances of the detectors along with the model's accuracy per each experiment are demonstrated in Figure 5.4. The experiment setup for this case was similar to the case of the ideal conditions, except that three phases were added to the set of experiments instead of four. The reason for that was the focus was intended on drift detection with different positions of drift data among non-drifting data. The first phase was when drifts occur in a recurrent order. The second phase was when gradual drifts occur after a flow of non-drifting data. Finally,




(a) Distances, p-values of MMD detector in ideal experimental conditions with Sub-sampling.

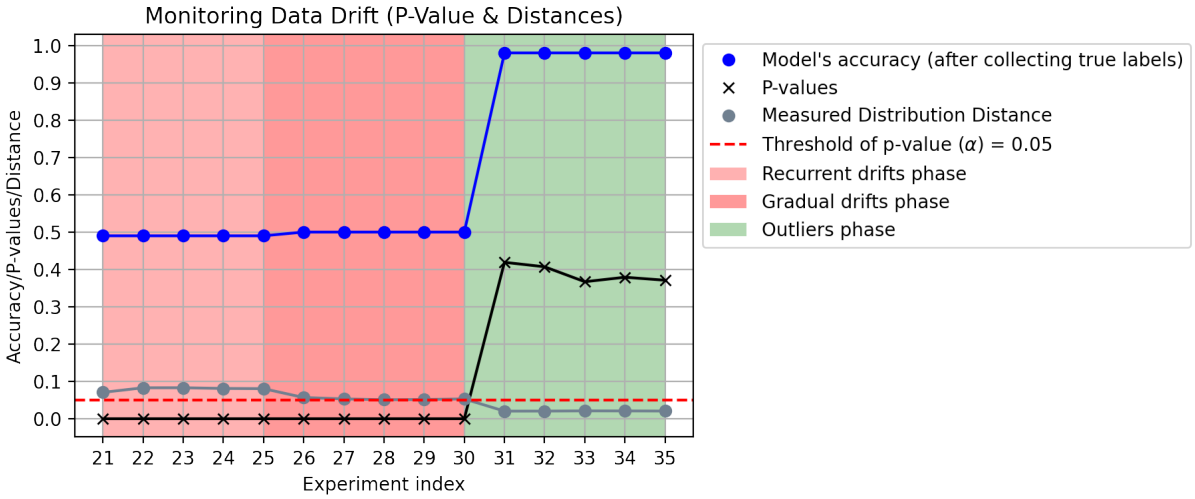



(b) Distances, p-values of KS detector in ideal experimental conditions with Sub-sampling.

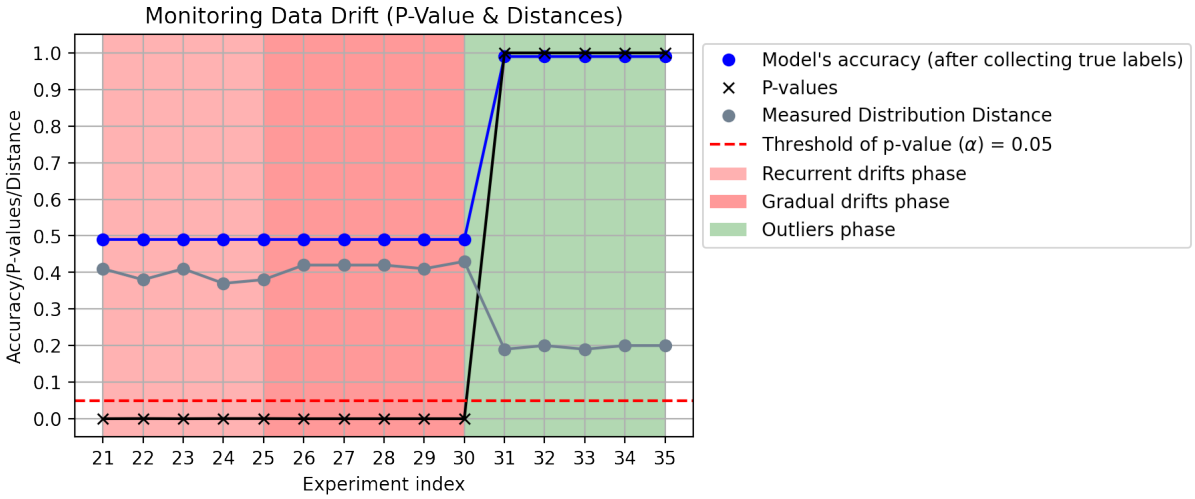
Figure 5.3: Distances, p-values of MMD and KS detector in ideal experimental conditions with Sub-sampling.

the case of sudden drifts. It should be mentioned that the results for the recurrent and gradual drifts were obtained by injecting 50 drifting images (with a novel class) into the monitor with size 100. Namely, 50% of the input flow was drifting. While for the sudden drift 10% of drifting data were injected within the non-drifting data. The results were obtained in Figure5.4a and Table 5.6 visualizes each experiment index on the x-axis against the model's accuracy, the detector's p-values, and the detector's distances. For Table 5.6, it starts with the MMD detector in the first phase (recurrent drifts), the measured distances were (0.071, 0.083, 0.083, 0.081, 0.081) as shown in Figure 5.4a for experiments 21 to 25. The p-values scored (0.000, 0.000, 0.000, 0.000, 0.000) while the model accuracy dropped to be (0.500, 0.500, 0.500,

0.500, 0.500). For the second phase, the measured distances for each experiment (26 to 30 in Figure 5.4a) were (0.057, 0.053, 0.051, 0.052, 0.053) while the p-values scored (0.000, 0.000, 0.000, 0.000, 0.000). It should be noted that the model accuracy remained the same with values (0.500, 0.500, 0.500, 0.500, 0.500). Finally, for the last phase with sudden drifts, the measured distances were (0.020, 0.024, 0.021, 0.020, 0.021). The p-values scored (0.419, 0.407, 0.367, 0.379, 0.371). The accuracy values of the model were (0.980, 0.980, 0.980, 0.980, 0.980). Next, the same procedure was followed to observe the KS performance




(a) Distances, p-values of MMD detector in non ideal experimental conditions.




(b) Distances, p-values of KS detector in non ideal experimental conditions.

Figure 5.4: Distances, p-values of MMD and KS detector in non ideal experimental conditions.

in the three defined non ideal drift phases. The results are shown in Figure 5.4b and Table 5.7. For the recurrent phase, the measured distances were (0.410, 0.380, 0.410, 0.370, 0.380). The p-values were

Table 5.6: Results from the MMD drift detector under testing in non ideal experimental conditions

| Experiment Index | MMD Distance | P-value | Model's Accuracy |
|---|---|---|---|
| 21 | 0.071 | 0.000 | 0.500 |
| 22 | 0.083 | 0.000 | 0.500 |
| 23 | 0.083 | 0.000 | 0.500 |
| 24 | 0.081 | 0.000 | 0.500 |
| 25 | 0.081 | 0.000 | 0.500 |
| 26 | 0.057 | 0.000 | 0.500 |
| 27 | 0.053 | 0.000 | 0.500 |
| 28 | 0.051 | 0.000 | 0.500 |
| 29 | 0.052 | 0.000 | 0.500 |
| 30 | 0.053 | 0.000 | 0.500 |
| 31 | 0.020 | 0.419 | 0.990 |
| 32 | 0.024 | 0.407 | 0.980 |
| 33 | 0.021 | 0.367 | 0.980 |
| 34 | 0.020 | 0.379 | 0.980 |
| 35 | 0.021 | 0.371 | 0.980 |

(0.000, 0.000, 0.000, 0.000, 0.000) and the model's accuracy through each experiment were (0.500, 0.500, 0.500, 0.500, 0.500). For the gradual drift phase, the measured distances were (0.420, 0.420, 0.420, 0.410, 0.430). The p-values scored (0.000, 0.000, 0.000, 0.000, 0.000) while the model's accuracies were (0.500, 0.500, 0.500, 0.500, 0.500). Finally, the sudden drift phases resulted the detector to score the following measured distances (0.190, 0.199, 0.190, 0.200, 0.200). The estimated p-values here were (1.000, 1.000, 1.000, 1.000, 1.000) while the model accuracies went back up to become (0.99, 0.99, 0.99, 0.99, 0.99). Figure 5.4b demonstrates the flow of the model's accuracy, the detector's p-values and distances across all the conducted experiments.

## 5.2 Performance Analysis of Drift Detectors

In this section, visualizations of the underlying processes of MMD and KS detectors are represented. First, the performance of the MMD detector is visualized for each phase of the conducted experiments. Then, the performance of the KS detector is similarly illustrated for each phase.

### 5.2.1 Performance Analysis for MMD

For the MMD Detector, Table 5.8 shows the total count of the calculated MMD distances after each permutation step for each experiment and how the p-values were estimated. A set of histograms for all the experiment is shown in the Appendix in Figure C.1 and Figure C.2. It can be seen from Table 5.8 that the p-values are the result of dividing the total counts of the distances higher than the distance calculated before permutations as explained in Section 3.3 by the number of permutations (1000).

Table 5.7: Results from the KS drift detector under testing in non-ideal experimental conditions.

| Experiment Index | KS Distance | P-value | Model's Accuracy |
|---|---|---|---|
| 21 | 0.410 | 0.000 | 0.500 |
| 22 | 0.380 | 0.000 | 0.500 |
| 23 | 0.410 | 0.000 | 0.500 |
| 24 | 0.370 | 0.000 | 0.500 |
| 25 | 0.380 | 0.000 | 0.500 |
| 26 | 0.420 | 0.000 | 0.500 |
| 27 | 0.420 | 0.000 | 0.500 |
| 28 | 0.420 | 0.000 | 0.500 |
| 29 | 0.410 | 0.000 | 0.500 |
| 30 | 0.430 | 0.000 | 0.980 |
| 31 | 0.190 | 1.000 | 0.980 |
| 32 | 0.199 | 1.000 | 0.980 |
| 33 | 0.190 | 1.000 | 0.980 |
| 34 | 0.200 | 1.000 | 0.980 |
| 35 | 0.200 | 1.000 | 0.980 |

### 5.2.2 Performance Analysis for KS

For the KS detector, the measured distance and sizes of the samples significantly affect the calculation of the corresponding p-value due to the inner method used as discussed in Section 3.5. Figure 5.5 shows the relationship between the measured distances and the corresponding p-values for fixed sample sizes of 100 for reference and test data. According to Figure 5.5, a drift alarm is fired (as p-value goes below 0.05) if the measured distances is at least 0.320. Table 5.7 demonstrates how the p-values were calculated based on this the measured distance.

## 5.3 Comparative Analysis between Detectors

Now that the results of both MMD and KS were illustrated and analyzed. The next step was to compare them using first, ROC and AUC. The concept of ROC and AUC were explained in Section 3.6. The comparison was based on their ability to detect drifts by comparing 100 samples from reference data against 100 samples from test data as implemented during the experiments. Then, a performance comparison in terms of time was implemented to investigate which method performed faster. The performance time is the time taken to calculate the p-value per experiment. Table 5.9 shows the average AUC results for each detector. The AUC values were calculated by outlining the average ROC curve for the detectors under different experimental phases. Namely, the no drift phase, the white noise phase, the severe blur phase, the severe blur phase, and the novel class phase. The table shows the high AUC values under all drift conditions except the sudden drifts for both detectors. For the MMD detector, the average AUC values scored (1.000, 0.989, 0.995, 0.509) while for KS the average AUC scored (1.000, 0.997, 0.985, 0.530). From the table, it is shown that both detectors perform poorly in case of sudden drifts.

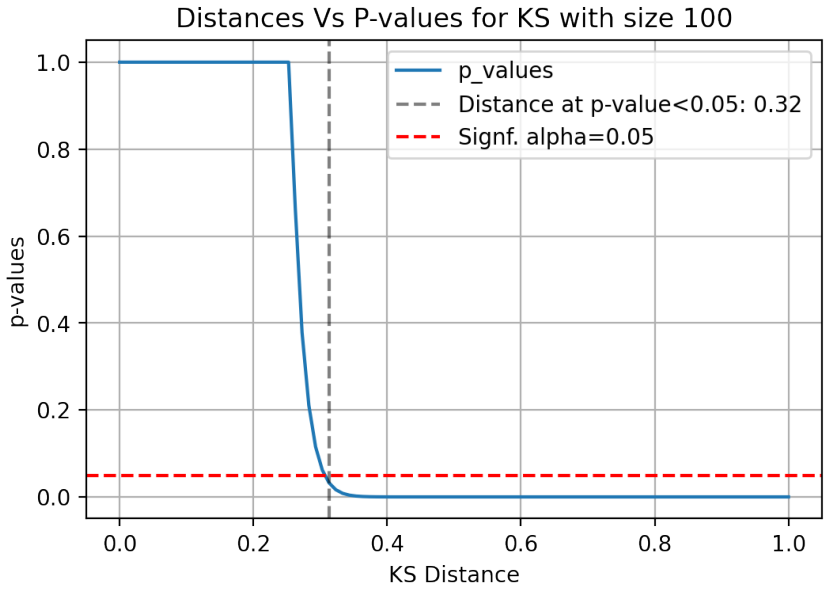
Table 5.8: Analysis of p-value calculation using Bootsrapping for MMD Detector.

| **Phase** | **Experiment Index** | **Original Distance (Before Permutation)** | **Total distances > Original Distance** | **P-value** |
|---|---|---|---|---|
| No Drift | 1 | 0.029 | 53 | 0.053 |
| | 2 | 0.044 | 1 | 0.001 |
| | 3 | 0.023 | 217 | 0.217 |
| | 4 | 0.027 | 95 | 0.095 |
| | 5 | 0.026 | 117 | 0.117 |
| White Noise | 6 | 0.034 | 11 | 0.011 |
| | 7 | 0.019 | 540 | 0.540 |
| | 8 | 0.024 | 155 | 0.155 |
| | 9 | 0.038 | 1 | 0.001 |
| | 10 | 0.021 | 327 | 0.327 |
| Severe Blur | 11 | 0.192 | 0 | 0.000 |
| | 12 | 0.172 | 0 | 0.000 |
| | 13 | 0.184 | 0 | 0.000 |
| | 14 | 0.164 | 0 | 0.000 |
| | 15 | 0.148 | 0 | 0.000 |
| Novel Class | 16 | 0.179 | 0 | 0.000 |
| | 17 | 0.231 | 0 | 0.000 |
| | 18 | 0.186 | 0 | 0.000 |
| | 19 | 0.204 | 0 | 0.000 |
| | 20 | 0.226 | 0 | 0.000 |
| Novel Class (Recurrent Drift) | 21 | 0.071 | 0 | 0.000 |
| | 22 | 0.083 | 0 | 0.000 |
| | 23 | 0.083 | 0 | 0.000 |
| | 24 | 0.081 | 0 | 0.000 |
| | 25 | 0.081 | 0 | 0.000 |
| Novel Class (Gradual Drift) | 26 | 0.057 | 0 | 0.000 |
| | 27 | 0.053 | 0 | 0.000 |
| | 28 | 0.051 | 0 | 0.000 |
| | 29 | 0.052 | 0 | 0.000 |
| | 30 | 0.053 | 0 | 0.000 |
| Novel Class (Sudden Drift) | 31 | 0.020 | 419 | 0.419 |
| | 32 | 0.024 | 407 | 0.407 |
| | 33 | 0.021 | 367 | 0.367 |
| | 34 | 0.020 | 379 | 0.379 |
| | 35 | 0.021 | 371 | 0.371 |

Table 5.9: Average AUC for MMD under different drift conditions.

| **Detector** | **Average AUC** | | | |
|---|---|---|---|---|
| | **Ideal conditions** | **Recurrent Drift** | **Gradual Drift** | **Sudden Drift** |
| MMD | 1.000 | 0.989 | 0.995 | 0.509 |
| KS | 1.000 | 0.997 | 0.985 | 0.530 |

Figure 5.5: Relationship between p-values and distances based on the inner method for 100 sample sizes.




### 5.3.1 Comparison of Time Performance for Detectors

Further comparison was implemented based on performance time shown in Table 5.10. From the table, it is shown that the MMD detector takes 0.267 seconds to generate and calculate a p-value, while KS takes 0.012 seconds.

Table 5.10: Time Performance for MMD and KS.

| **Detector** | **Average Time (Seconds)** |
|---|---|
| MMD | 0.267 |
| KS | **0.012** |

### 5.3.2 Comparative Analysis with Adding Dimension Reducers

Additional experiments were implemented in which different dimension reduction algorithms were added to the used feature extractor. The idea was to investigate how reducing the dimensions of features would affect the overall performance of the drift detectors. Furthermore, an untrained Auto-Encoder was used with random weight initialization to perform feature extraction and dimension reduction. The results are shown in Table 5.11 and Table 5.12. Starting with MMD Detector, the results show that using Principle Component Analysis, the performance remained high with average AUC values (0.800, 0.980, 0.980, 0.530). Using the Sparse Random Projection reducer has produced average AUC values of (0.000,

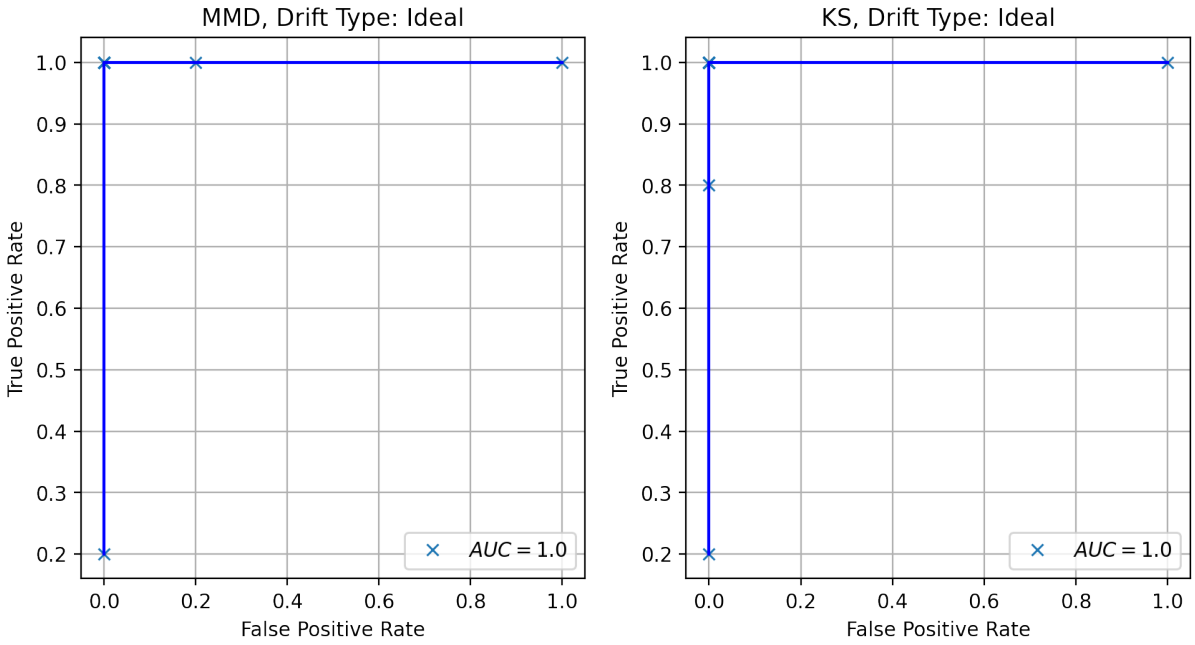



Figure 5.6: ROC for MMD and KS in ideal drift conditions.

0.000, 0.000, 0.000). Finally, the average AUC values resulting from using Auto-Encoder were (0.960, 0.492, 0.468, 0.528). In Table 5.12, the average AUC for each dimension reducer was calculated. The results show that without using a dimension reducer, the AUC values were (1.000, 0.989, 0.995, 0.509). After using PCA, the average AUC results scored (1.000, 0.980, 0.998, 0.530). For the SRP reducer, the average AUC has become (0.000, 0.000, 0.000, 0.000). Finally, using the UAE as a feature extractor and a dimension reducer scored (0.960, 0.492, 0.468, 0.526). Detailed performance of each dimension reducer in non-ideal drifts is shown in Appendix D.

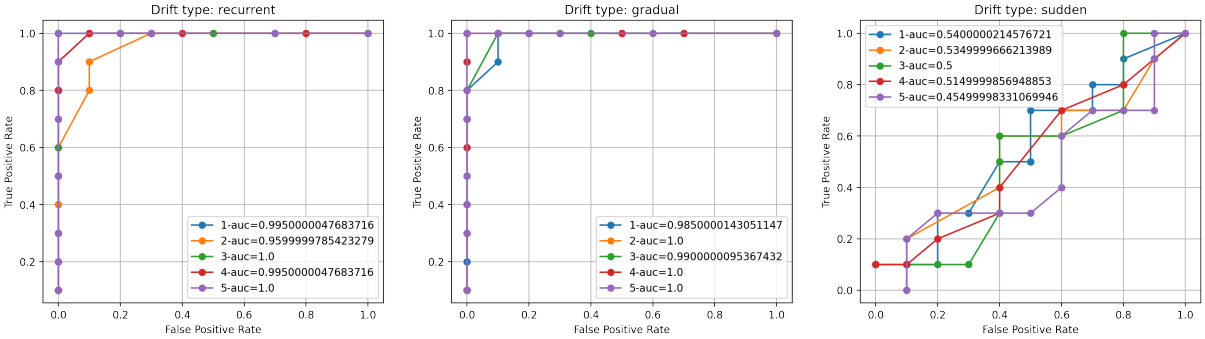
Table 5.11: Average AUC for MMD drift detector with dimension reducers.

| **Detector** | **Drift Type** | **Dimension Reducer** | | | |
|---|---|---|---|---|---|
| | | **No Reducer** | **PCA** | **SRP** | **UAE** |
| MMD | Ideal | 1.000 | 0.800 | 0.000 | 0.960 |
| | Recurrent | 0.989 | 0.980 | 0.000 | 0.492 |
| | Gradual | 0.995 | 0.998 | 0.000 | 0.468 |
| | Sudden | 0.509 | 0.530 | 0.000 | 0.526 |

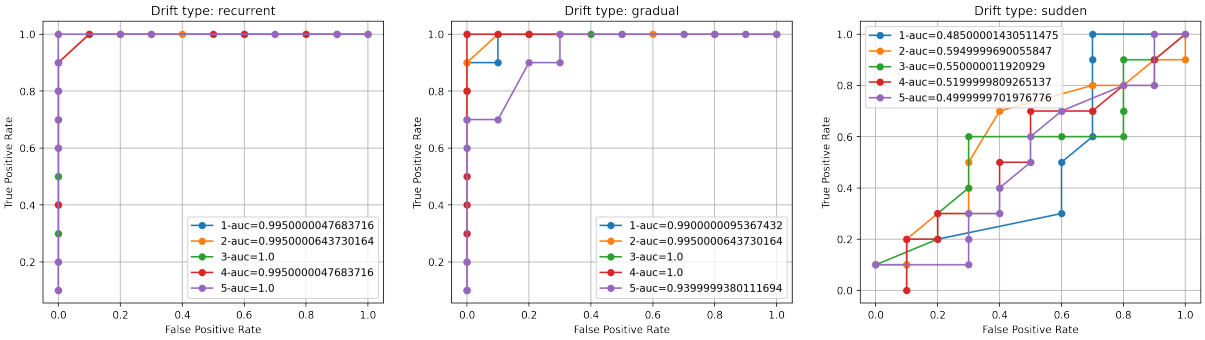
### 5.3.3 Comparative Analysis with Different Drift Ratios

In the previous experiments, specifically in the non-ideal conditions, recurrent drifts and gradual drifts were experimented with 50% drift ration. Namely, the experiments were conducted by mixing 50 samples

Figure 5.7: ROC Curves for drift detectors under non-ideal conditions.




(a) ROC for MMD detector for sudden, recurrent, and gradual drifts.




(b) ROC for KS detector for sudden, recurrent, and gradual drifts.

Table 5.12: Average AUC for KS drift detector with dimension reducers

| **Detector** | **Drift Type** | **Dimension Reducer** | | | |
|---|---|---|---|---|---|
| | | **No Reducer** | **PCA** | **SRP** | **UAE** |
| KS | Ideal | 1.000 | 0.981 | 0.000 | 0.975 |
| | Recurrent | 0.997 | 0.966 | 0.000 | 0.411 |
| | Gradual | 0.985 | 0.975 | 0.000 | 0.447 |
| | Sudden | 0.530 | 0.530 | 0.000 | 0.498 |

of drifting data with 50 samples of non-drifting data. Here, different ratios from 0% up to ideal drift conditions (100%) were set for testing. The goal is to study the detectors' ability to detect drifts based on the amount of drifting data in the monitoring buffer. Figure A.2 and Figure A.5 show the p-values result in response to different drift ratios. From Figure A.2, it is shown that the MMD detector is able to detect drifts (p-values go below 0.05) if there exist at least 34 drifting samples in a buffer filled with 100 test samples. For the KS detector, there should be at least 45 drifting samples as shown in Figure A.5.

Figure 5.8: Relationship between p-values and drift ratios for buffer size = 100 for MMD.

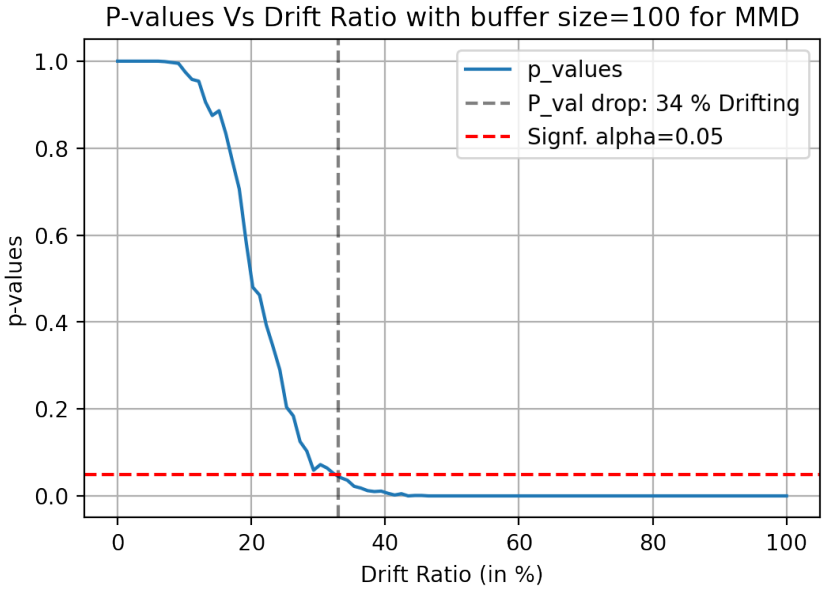



Figure 5.9: Relationship between p-values and Drifts for MMD with size 10.

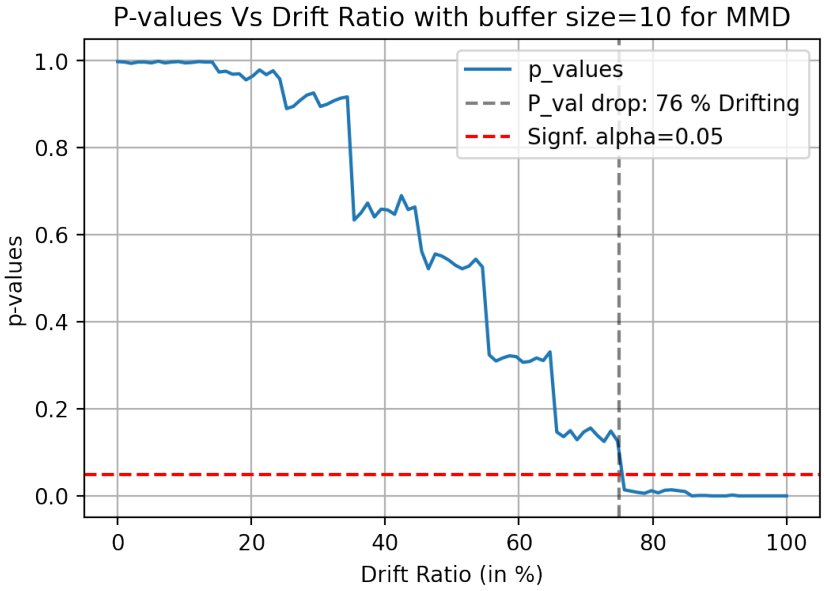


Figure 5.10: Relationship between p-values and drift ratios for buffer size = 100 for KS.

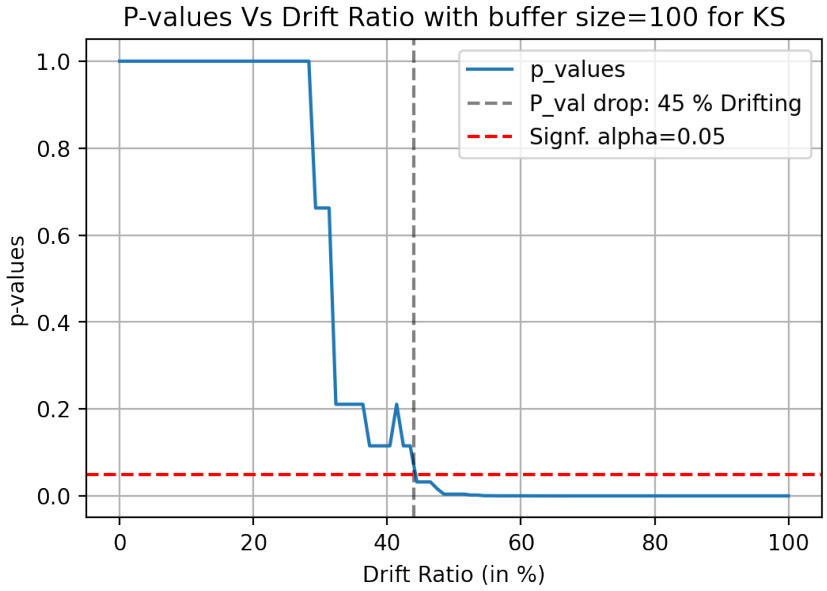
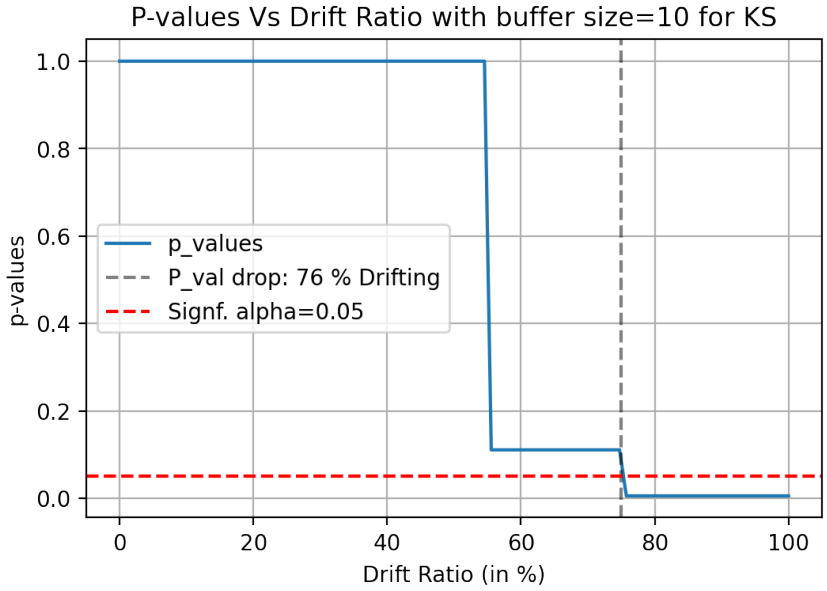



Figure 5.11: Relationship between p-values and Drifts for KS with size 10.



# 6

# Discussions

In this chapter, the results presented in Chapter 6 are explained in detail. The goal was to understand why the detectors performed well in some cases like the ideal drift case, the gradual drift case, and the recurrent drift case while poorly performing in the sudden drift case, the Sparse Random Projection Case, and the Untrained Auto-Encoder. Furthermore, the relationship between the detectors' performances and the order of the drifting data (recurrent and gradual) in fixed sizes is discussed.

## 6.1 The Performance in Ideal Drift Cases

In the Section 5.1, multiple results have been shown in different experimental conditions. The goal was to investigate the detectors' ability to identify drift in different scenarios. It can be clearly seen from the results in Tables 5.2 and 5.3 that both MMD and KS detectors were able to identify drifts perfectly in the ideal drift cases. However, this ideal ability to detect drifts comes with the cost of firing several false-positive alarms in the cases where no drift data existed, or in the case where only white noise existed in the monitoring buffer. The analysis of the performances of these detectors is demonstrated in the following subsections for each detector separately.

### 6.1.1 Performance of MMD Detector in Ideal Drift Cases

As explained in Section 3.3, the Maximum Mean Discrepancy-based detector takes the stacked features produced by the feature extractor as an input. Then, a distance between these features and reference features from training data was measured. This distance as discussed is between the mean of the reference's embedding in Hilbert Space and the input's embedding in Hilbert Space. These embeddings (also known as functions) are generated using a pre-defined kernel function. The trick comes after keeping this measured distance aside, implementing bootstrapping on the reference and input function, and remeasuring the distance between the functions. In the drift cases, not only new measures were taken, but they become smaller than the first taken measurement as shown in Figure C.1. The figure shows that for each experiment, the p-value corresponds to the ratio of the total numbers larger than the first measured distance (vertical red line) to the number of times the bootstrapping was run. However, for a no drift case, these new measures become larger than the first taken measurement as shown in Table 5.2. The problem with the false-positive results comes from the fluctuating behavior or the p-values as seen in Figure 5.2a.

Even though the threshold distances seem to increase along with the severity of drifts in each experiment (from no drift to novel class), the p-values keep fluctuating. As shown in Figure C.1, this was because of the bootstrapping process which occurs by randomly permuting the order of the features in the generated kernel function. Hence, the generation of random distance. The p-values were then estimated by counting how many measurements were above the distance threshold which was calculated before bootstrapping. If the number of distances below the threshold distance is low, this means that the threshold distance is not the lowest. Hence, the null hypothesis is subject to rejection. The rejection is assured if the p-value falls below 0.05.

### 6.1.2 Performance of KS Detector in Ideal Drift Cases

As explained in Section3.5, the measurement of the KS distance is based on taking the maximum point in the generated Cumulative Distribution Functions (CDF). Then the stable inner method was used to estimate the p-values as shown in Figure 5.5. In the ideal no drift case, the high p-values indicate that most of the points in the inner method's grid for each sample fall within the corridor drown by the estimated threshold. Furthermore, based on the relationship between distances and p-values in the Figure 5.5, each distance that was above 0.320 resulted in a p-value that was below 0.05.

### 6.1.3 Overcoming False-Positives with Sub-Sampling

One issue of the functionality of the MMD and the KS was their high sensitivity to neglectable drifts such as the white noise as shown in Figure 5.2. The reason these drifts were considered neglectable was due to the high accuracy that the model achieved which indicates that these drifts can be tolerated. Table 5.4, Table 5.5, Figure 5.3a and Figure 5.3b illustrate how sub-sampling the reference set into two sets samples overcame this issue by producing no drift alarm (no p-value was below 0.05). This modification comes with the cost of multiplying the performance time by the number of sub-samples. Luckily, The KS detector still showed faster performance than MMD as seen in Table 5.10.

## 6.2 The Performance in Non-Ideal Drift Cases

In non-ideal drift cases, the monitoring buffer consists of drifting data and non-drifting data. Based on the order of the drifting and non-drifting data, the experiments were segmented into three types as a general form of abstracting drift types. These types were the sudden drifts, the recurrent drifts, and the gradual drifts as discussed in Section 3.8.2. The results indicate that both MMD and KS detectors were able to detect the recurrent drifts and the gradual drifts successfully, but not the sudden drifts as shown in Figure 5.7a and Figure 5.7b.

### 6.2.1 Performance of MMD Detector in Non-ideal Drift Cases

For the MMD detector, the results show that the ability to detect drifts was independent of the order of the samples in the monitoring buffer. That was because of the bootstrapping process when calculating

the p-value. The permutations of the order in the kernel functions disregard the order of the input data. That means, if drifting data exist within the monitoring data, then the ability of the MMD detector to identify drifts would be based on the number of drifting samples within the test data in the monitoring buffer. Figure A.2 shows that in the monitoring buffer for 100 samples, the drift starts firing drift alarms when there exists a minimum of 34 drifting samples in the buffer.

### 6.2.2 Performance of KS Detector in Non-ideal Drift Cases

Similar to the MMD detector, the KS detector has proven to be able to detect recurrent and gradual drifts, but not sudden ones. Figure 5.7b shows the poor performance of the KS detector in the sudden drift. Similar to the MMD detector, it was shown that the number of drifting samples affects the performance of the KS detector as shown in Figure A.5. According to this figure, the KS detector can detect drifts if there exists a minimum of 45 samples in the monitoring buffer that takes 100 samples.

## 6.3 Effect of Feature Extraction on Drift Detection

Since the experimented drift detectors compare distributions of features, it was essential to investigate the effect of changing the representation of the features on the ability to detect drifts. The results in Table 5.11 and Table 5.12 indicate that using PCA maintains the high performance of the drift detector. Furthermore, the results indicate that the selection of the feature extractor significantly affects the performance of the drift detector. This was reflected by how the pre-trained ResNet neural network with no reducer has outperformed the Untrained Auto-Encode neural network in all the drift cases.

## 6.4 Effect of Sample Size on Drift Detection

From the results in Figure A.2 and Figure A.5, it was shown that the drift detectors were only able to detect drifting distributions if there exist a minimum number of drifting samples within the monitoring buffer. This explains why the detectors fail to detect sudden drifts as they have fewer drifting samples. Furthermore, these figures show that a relationship exists between the p-values and the test distances per sample size. It should be mentioned that, while the experiments implemented in this project used a buffer size of 100 samples, the relationship between the p-values and the drift ratios (number of drifts per buffer size) was investigated for different sizes such as 10 and 100. The results are demonstrated in Figures A.1, Figures A.3, Figures A.4, and Figure A.6. The results show that for each buffer size, there is a need to investigate the relationship between the p-values and the drift ratios as it significantly affects how fast drifts can be detected.

## 6.5 Effect of the Order of Drifts on Drift Detection

As discussed in the previous section, the results in Figure A.2 and Figure A.5 show that the detectors' performances depend significantly on the sample size regardless of the order. This again appeared clearly in the AUC for the detectors with recurrent, gradual, and sudden drifts as shown in Figure 5.7a and Figure 5.7b for MMD and KS respectively. In particular, the AUC values for detectors on the sudden

drifts were low due to the few drifting images in that experiment (10 drifting samples out of 100), while for the recurrent and gradual drifts (50 drifting samples out of 100), the AUC scores were 50 which is greater than 31 for MMD, and 45 for KS.

## 6.6 Which Algorithm Performs Best?

Now that the proposed detectors were tested and evaluated, the next step was to determine which detector performs better for covariate or feature drift detection. It was shown from the results in ideal and non-ideal conditions that both detectors perform properly with high AUC to detect drifts. Even though they outperform each other in different scenarios, the differences were not notable. The similarity in the performances of both detectors can be referred to the hypothesis testing structure that was adapted as the way to trigger drift events. However, the test statistics (MMD and KS) showed different performances. Unlike MMD distances, the observed KS distances kept changing clearly with each type of drifts (Figures 5.2a,5.2b,5.4a,5.4b). Also, the KS showed an outstanding time performance when compared to the MMD detector. Furthermore, before implementing sub-sampling, the KS had fewer false-positive drift alarms. In addition, the flow of the p-value over each phase seemed more stable than the p-values calculated by MMD. Hence, with the provided configurations in the implemented experiments, the KS detector showed to be faster, smoother, and more accurate than the MMD detector. Table 6.1 gives a general comparison based on the results obtained from all the experiments.

Table 6.1: A summarized (general) comparison between drift detectors based on the obtained results.

| **Criteria of Comparison** | **Detector** | | **Reference of Comparison** |
|---|---|---|---|
| | **MMD** | **KS** | |
| Dependency on sample Size | x | x | Section 6.4 |
| Independence on the order of drifting samples | x | x | Section 6.5 |
| Faster performance in drift checking | | x | Table 5.10 |
| Stable performance (In no drift cases) | | x | Figure 5.2 |
| Most accurate drift detection | | x | Figure 5.2 |

# 7

# Conclusions

In this chapter, conclusions about the findings in this project are presented. The objectives of this project are revisited and checked whether they were met or not. In addition, the delivered contributions are mentioned. Furthermore, the encountered challenges during the experiments and the literature review are discussed. Finally, the recommendations for future work and future points of focus are stated.

## 7.1 Revisiting The Problem Statement

As discussed in Section 1.4, the objectives of this project were defined as follows.

- To conduct a comprehensive study about the state-of-the-art regarding drift detection.
- To select methods for evaluation based on the findings of the study.
- To analyze the performance of the selected methods and evaluate them.
- To address the technical requirements to develop a monitoring task.

Therefore, a literature review of recent studies was required. The literature review was conducted and covered in Chapter 2. From the conducted literature review, two algorithms were selected for evaluation under different drift experiments as discussed in Chapter 3. The selection of the algorithms meets the second objective mentioned above. Furthermore, the selected algorithms were evaluated against different drift conditions and compared against each other. As appeared from the results in Chapter 5, Kolmogorov-Smirnov has outperformed the Maximum Mean Discrepancy in terms of the ability to detect drifts and in time performance. Finally, based on the developed experimental pipeline, the technical requirements to develop a data monitoring task were defined. The main components of a data monitoring task were the feature extractor, the hypothesis test, and the drift alarm as discussed in Chapter 3 and evaluated in Chapter 5.

## 7.2 Contributions

In this section, the contribution of this work is presented in detail. In addition to meeting the objectives discussed in Section 7.1, the following contributions were added by this project.

- A comprehensive study of data drifts was delivered from a statistical perspective.
- The interrelation between data drifts, anomalies, outliers, and out-of-distribution points is thoroughly investigated.
- A literature review of the existing techniques for drift detection was explored in the context of low dimensional data, and high dimensional data.
- An experimental machine learning pipeline was developed to conduct drift detection experiments using the selected algorithms.
- The performances of the selected algorithms were evaluated in a case study of image classification using the Stream-51 dataset for novelty detection.
- A modification on the hypothesis testing task was implemented to mitigate false alarms caused by negligible noises in images (white noise). Namely, segmenting the reference data set and implementing the hypothesis test per each segment, and averaging the estimated p-values.
- Provision of explained functionality of hypothesis testing for Maximum Mean Discrepancy and Kolmogorov-Smirnov per each experiment. Namely, explaining how each algorithm was utilized to detect drift events.
- The effect of the sample size and the order of drifting and non-drifting data on the drift detectors' performances were demonstrated for an efficient design of a data monitoring task.

## 7.3 Areas of Relevant Applications

In this section, the areas in which the findings of this research can be relevant are mentioned in the following points.

- Studying the main components of the drift monitoring tasks in depth (feature extractor and hypothesis testing framework).
- Identifying how fast a drift should be detected based on the sizes of the monitoring buffer and the samples.
- Identifying the needed configurations to develop a drift monitoring task.
- Triggering retraining machine learning models that are deployed in a changing environment.

While there exist several aspects of drift detection to explore, the findings of the research can provide a preliminary establishment of monitoring drifts.

## 7.4 Future work

During researching drift detection, several lessons were learned which remarkably contributed to the delivered results and the provided explanations. These learned lessons can be generally summarised as follows.

- The type of data is important to select which drift detector to use. According to the literature review conducted in 2, low dimensional data can use classical techniques. However, high dimensional data such as images, texts, or high dimensional features need sophisticated approaches.
- The hypothesis tests were run on the extracted features. Therefore, the selection of the feature extractor impacts the overall performance of the detectors as seen in Table 5.11.
- The sample sizes for the reference samples and the test samples significantly impact the detector's ability to detect drift events as seen in Figures A.2 and Figures A.5.

While the scope of research in this project focused on drift detection as part of a monitoring task, several issues require further investigation. These issues are listed as follows.

- Given that the hypothesis tests depend heavily on the sample size of observations, finding the sample size that enables the detectors to perform as desired is needed.
- Based on the type of input data, a proper feature extraction technique should be used. Therefore, developing a feature extractor that optimally facilitates drift detection is required.
- While using hypothesis testing enables to monitor drifts, determining how the drift is shaped (e.g sudden, recurrent, gradual, outlier) still requires further analysis.
- While using the p-values as an indicator for drift events, the test statistics could be a useful metric to assess drift severity.
- While the selection of reference set was randomly done from the training dataset, there is a need to investigate approaches of reference data selection such that the extracted reference features can be specified for drift monitoring. Namely, one can determine which features precisely are the test data drifting from.

Even though detecting unfamiliar data to a machine learning model is not a novel problem, still, ongoing researches are trying to identify how to solve this problem efficiently. While this project evaluates the ability to detect drifts for high-dimensional data such as images, further experimentation is needed in the context of natural language understanding and time-series analysis. Also, an extension of the evaluation of drift detection techniques to different types of drifts such as adversarial changes and outliers for high-dimensional data is needed. Additionally, explaining the outcomes of models or feature extractors that are black boxes requires further investigation. The reason is to determine what exactly in the input is drifting. Examples of explainability approaches are as follows.

- Accumulated Local Effects [58].

- The Anchors technique [70].
- Shaply Additive exPlainations-based (SHAP) techniques [53] [54].

Finally, more drift detection techniques such as the Wasserstein technique [91] and the covered methods from Chapter 2 can be reevaluated for all the use cases mentioned above.

# A

# Additional Analysis on P-values

## A.1 Relationship between P-values and Drift Ratios

Listed here are the relationships between p-values and drift ratios for sample sizes 10, 100, and 1000. The figures below demonstrate these relationships for Maximum Mean Discrepancy (MMD) and Kolmogorov-Smirnov (KS).

Figure A.1: Relationship between p-values and Drifts for MMD with size 10.

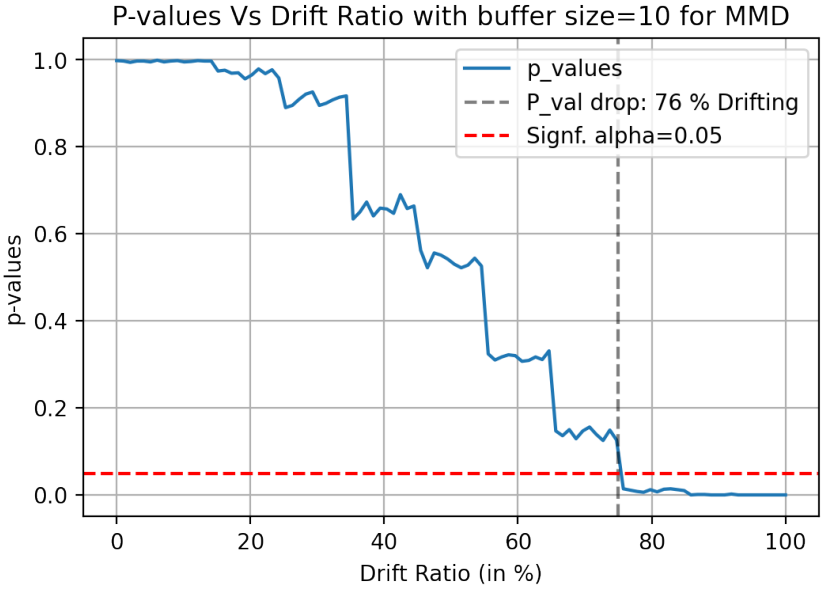


Figure A.2: Relationship between p-values and Drifts for MMD with size 100.

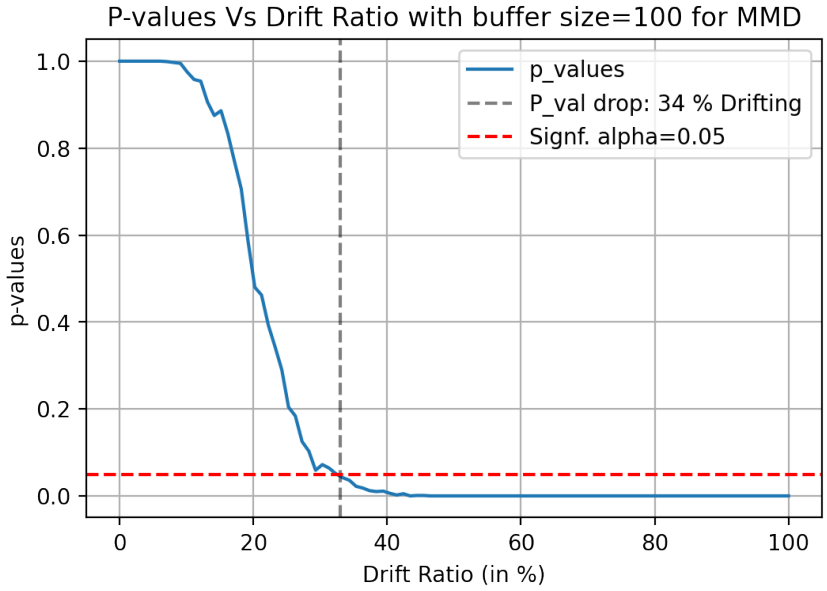



Figure A.3: Relationship between p-values and Drifts for MMD with size 1000.

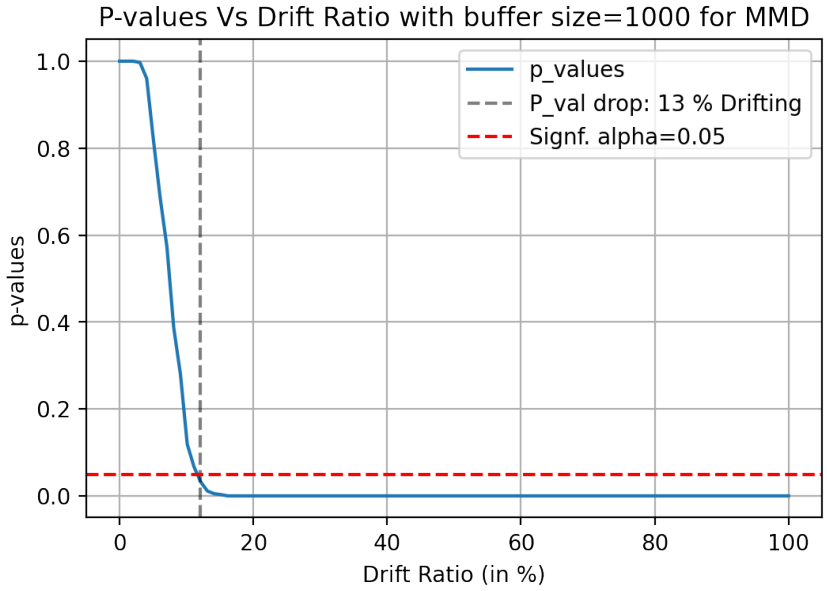


Figure A.4: Relationship between p-values and Drifts for KS with size 10.

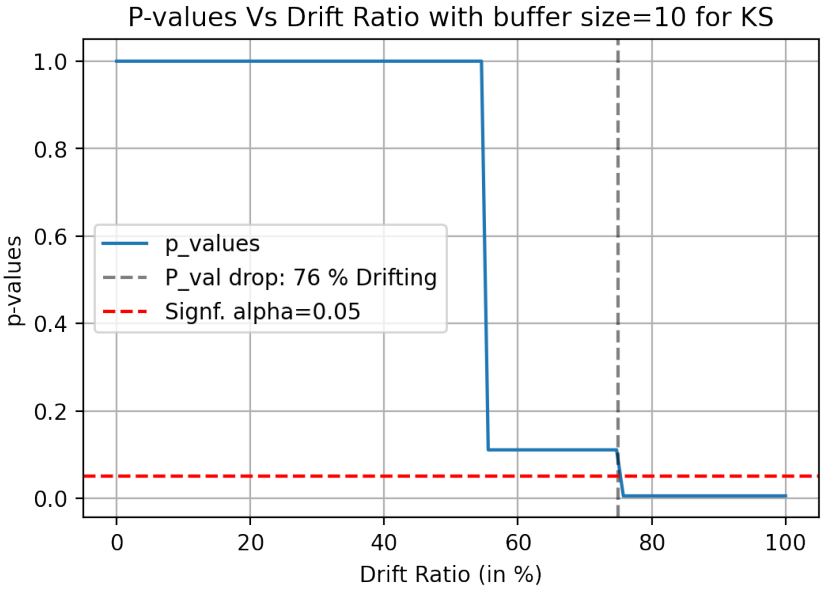



Figure A.5: Relationship between p-values and Drifts for KS with size 100.

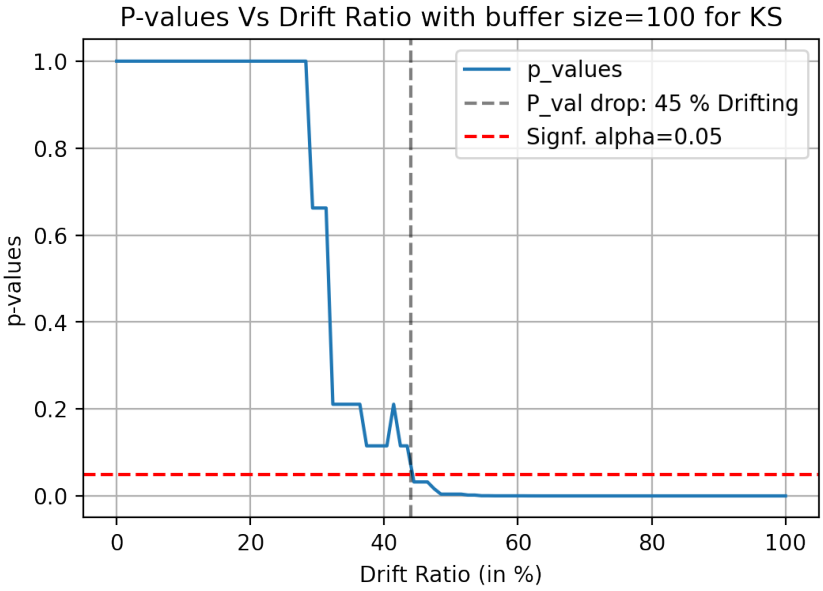
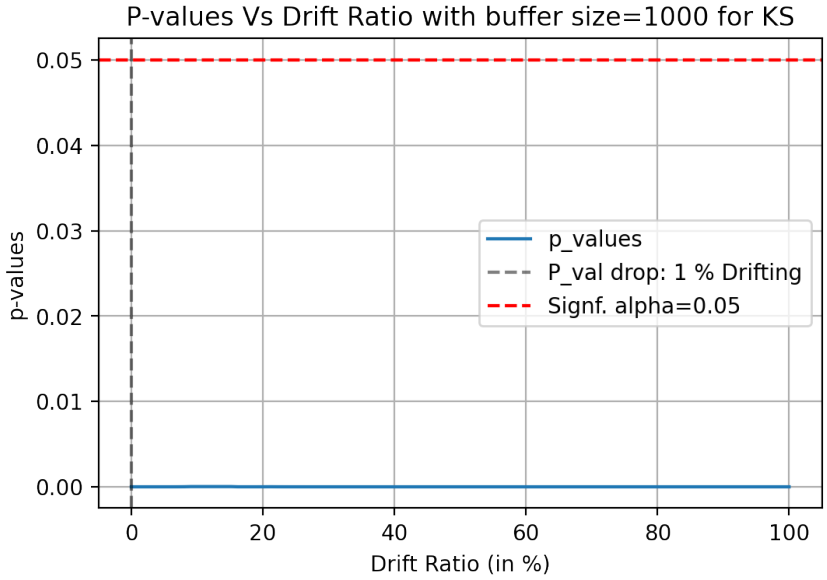


Figure A.6: Relationship between p-values and Drifts for KS with size 1000.



# B

# Relationship Between P-values and KS Distances

Presented here are the relationships between p-values and KS distances for sample sizes 10 and 1000 following the inner method for the KS detector.

Figure B.1: Relationship between p-values and distances based on the inner method for 10 sample sizes.

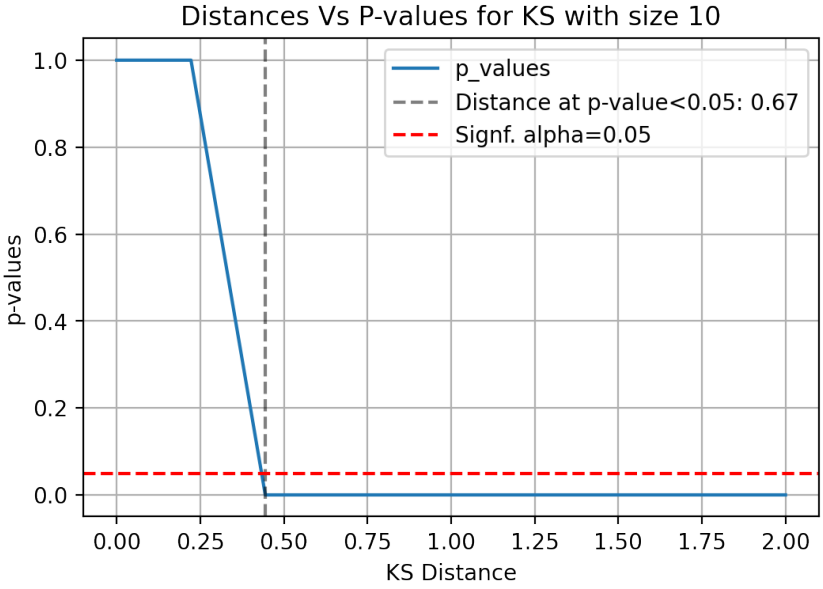
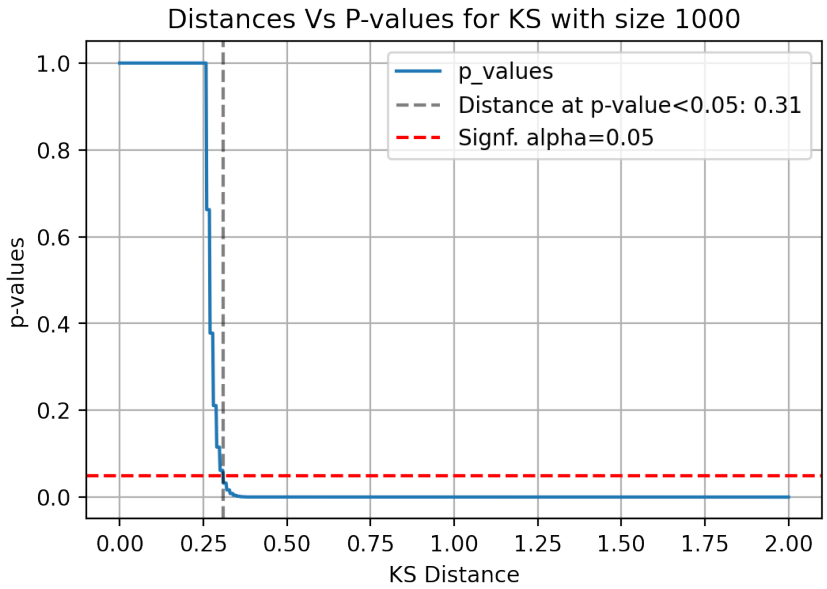


Figure B.2: Relationship between p-values and distances based on the inner method for 1000 sample sizes.



# C

# Analysis of MMD with Bootstrapping

Here, the performance of MMD with bootstrapping is explained. The aim is to provide visual analysis on how the p-value is estimated using bootsrapping. The two sets of figures demonstrate the MMD performances in ideal conditions and non-ideal conditions respectively.

Figure C.1: Relationship between p-values and distances based on the inner method for 100 sample sizes.

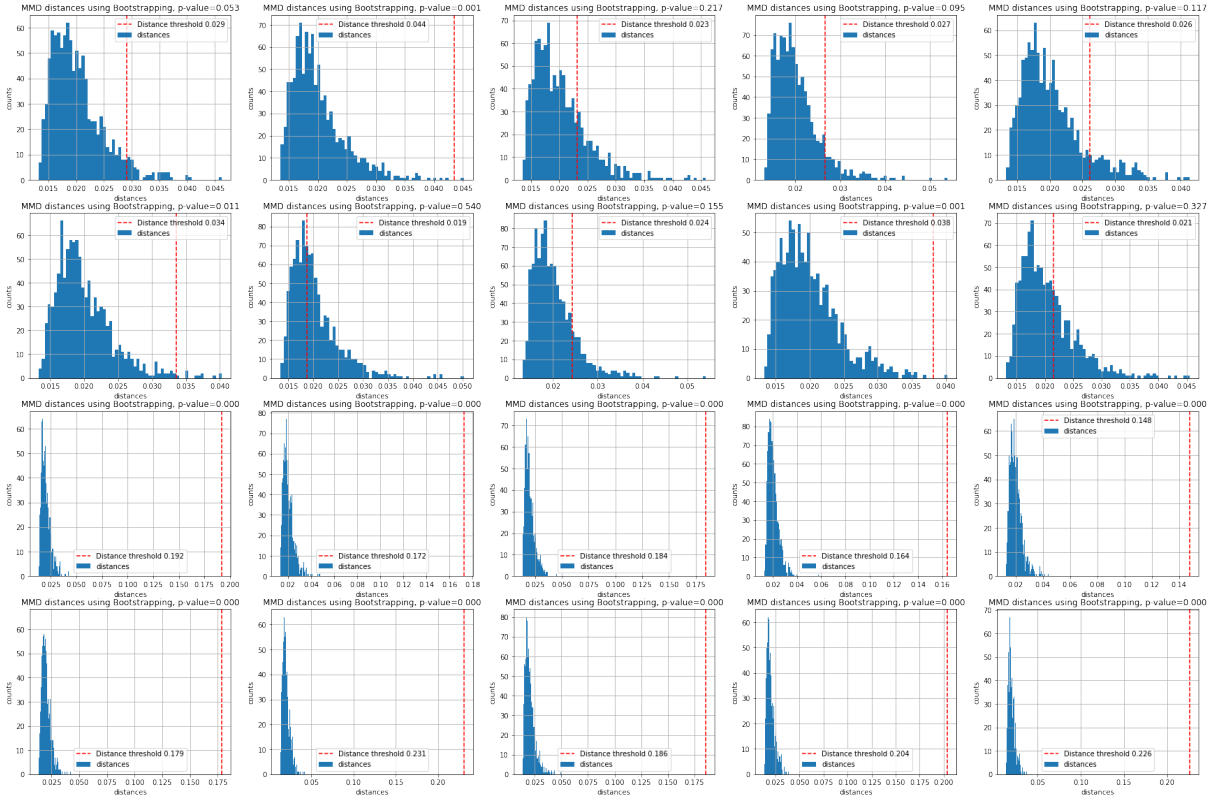
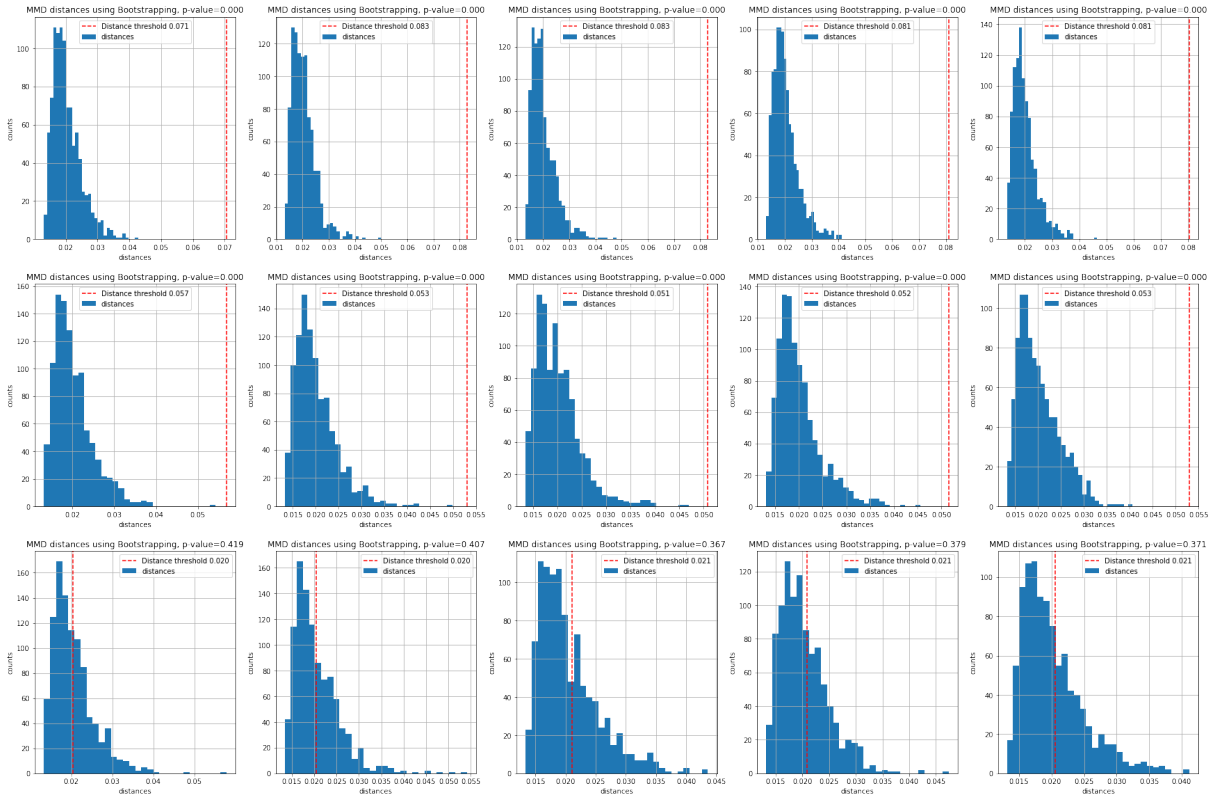


Figure C.2: Relationship between p-values and distances based on the inner method for 100 sample sizes.

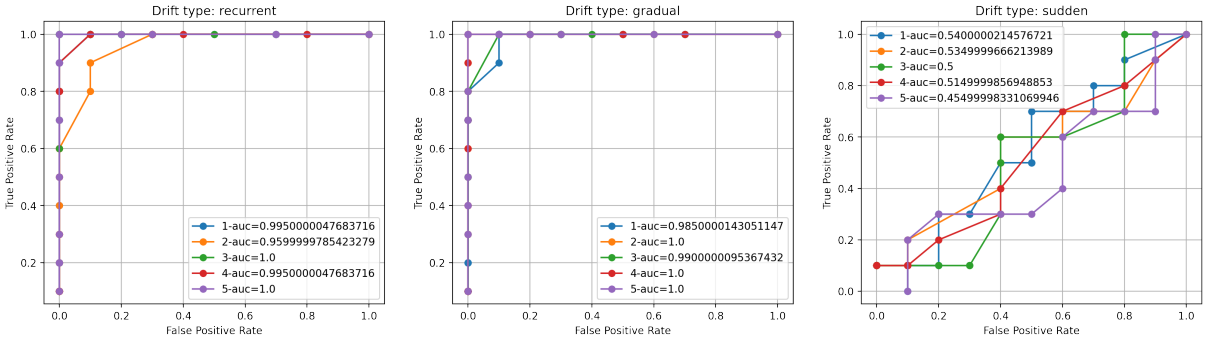


# D

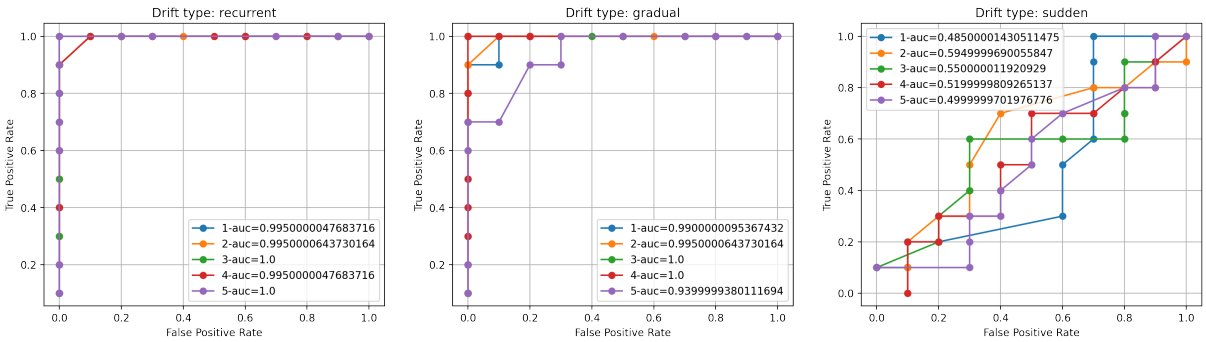
# AUC of Drift Detectors with Feature Extractors and Dimension Reducers

In this chapter, the ROC curves used to calculate the average AUC in Tables 5.11 and Table 5.12 are demonstrated. The figures focus on the ROC of the sudden, recurrent, and gradual drift because they show irregular behavior in each experiment unlike in the ideal drifts.

Figure D.1: ROC for MMD and KS detectors without a dimension reducer.

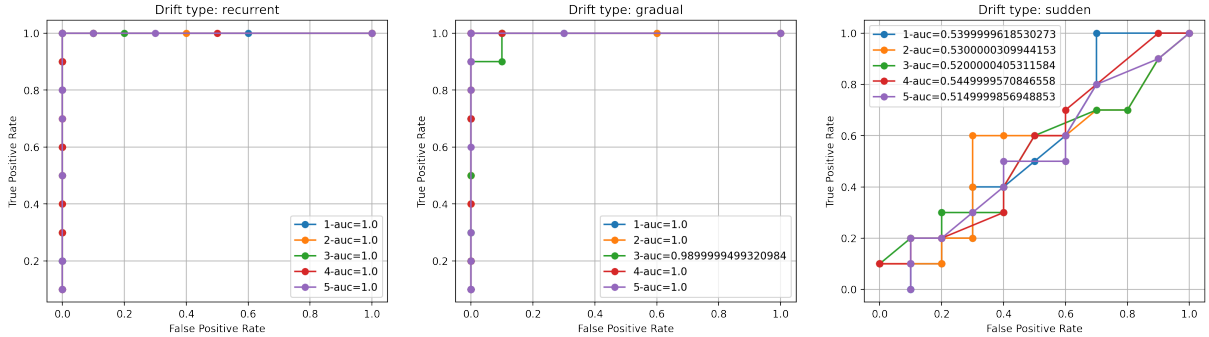



(a) ROC for MMD detector without a dimension reducer.

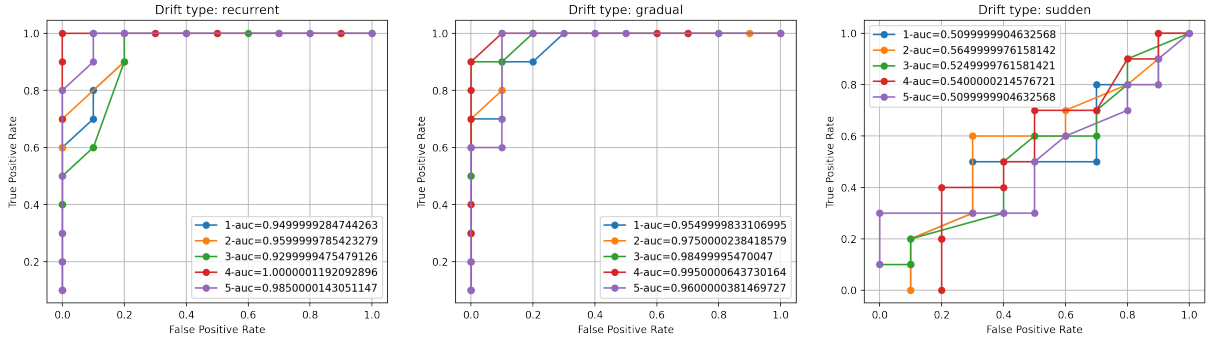



(b) ROC for KS detector without a dimension reducer.

Figure D.2: ROC for MMD and KS detectors with PCA.

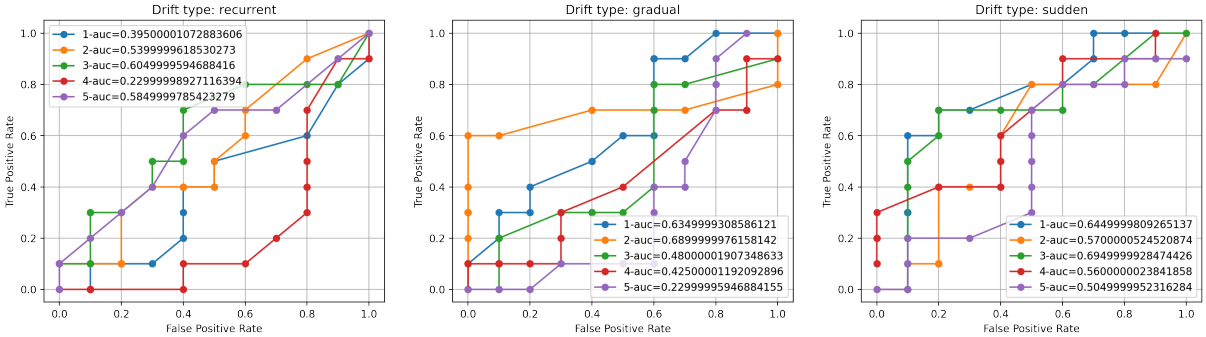



(a) ROC for MMD detector with PCA.

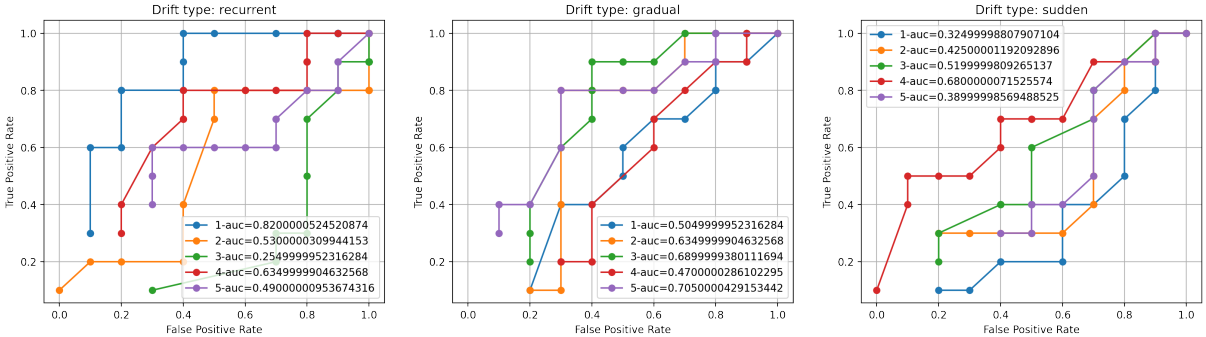



(b) ROC for KS detector with PCA.

Figure D.3: ROC for MMD and KS detectors with SRP.

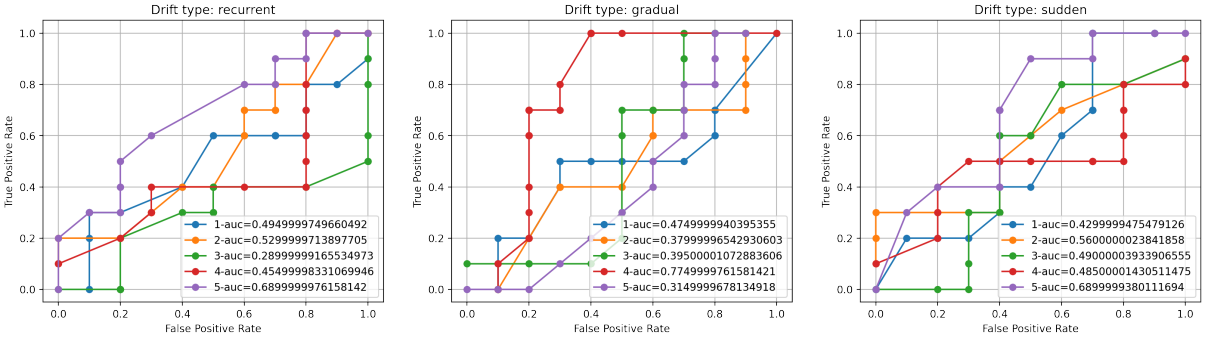



(a) ROC for MMD detector with SRP.

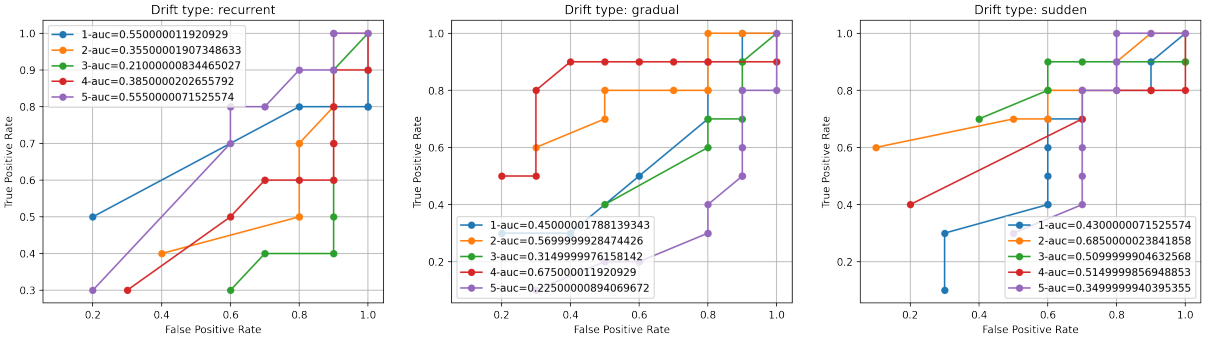



(b) ROC for KS detector with SRP.

Figure D.4: ROC for MMD and KS detectors with UAE.




(a) ROC for MMD detector with UAE.




(b) ROC for KS detector with UAE.

# References


[1] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[2] Dimitris Achlioptas. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 274–281, 2001.

[3] Samuel Ackerman, Parijat Dube, and Eitan Farchi. Sequential drift detection in deep learning classifiers. *arXiv preprint arXiv:2007.16109*, 2020.

[4] Charu C Aggarwal. High-dimensional outlier detection: the subspace method. In *Outlier analysis*, pages 149–184. Springer, 2017.

[5] Frank J Anscombe. Rejection of outliers. *Technometrics*, 2(2):123–146, 1960.

[6] César A Astudillo, Javier I González, B John Oommen, and Anis Yazidi. Concept drift detection using online histogram-based bayesian classifiers. In *Australasian Joint Conference on Artificial Intelligence*, pages 175–182. Springer, 2016.

[7] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and Rafael Morales-Bueno. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, volume 6, pages 77–86, 2006.

[8] Lucas Baier, Tim Schlör, Jakob Schöffer, and Niklas Kühl. Detecting concept drift with neural network model uncertainty. *arXiv preprint arXiv:2107.01873*, 2021.

[9] Daniel Barbara, Ningning Wu, and Sushil Jajodia. Detecting novel network intrusions using bayes estimators. In *Proceedings of the 2001 SIAM International Conference on Data Mining*, pages 1–17. SIAM, 2001.

[10] Jean Paul Barddal, Heitor Murilo Gomes, and Fabrício Enembreck. A survey on feature drift adaptation. In *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1053–1060. IEEE, 2015.

[11] Peter L Bartlett and Marten H Wegkamp. Classification with a reject option using a hinge loss. *Journal of Machine Learning Research*, 9(8), 2008.

[12] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavalda. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 139–148, 2009.

[13] Markus Borg. Agility in software 2.0–notebook interfaces and mlops with buttresses and rebars. *arXiv preprint arXiv:2111.14142*, 2021.

[14] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D Sculley. The ml test score: A rubric for ml production readiness and technical debt reduction. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1123–1132. IEEE, 2017.

[15] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[16] Simon Byers and Adrian E Raftery. Nearest-neighbor clutter removal for estimating features in spatial point processes. *Journal of the American Statistical Association*, 93(442):577–584, 1998.

[17] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[18] Dev Kumar Chaudhary, Sandeep Srivastava, and Vikas Kumar. A review on hidden debts in machine learning systems. In *2018 Second International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 619–624. IEEE, 2018.

[19] K Chen, SC Lu, and HS Teng. Adaptive real-time anomaly detection using inductively generated sequential patterns,". In *Fifth Intrusion Detection Workshop, SRI International, Menlo Park, CA*, 1990.

[20] Corinna Cortes, Giulia DeSalvo, and Mehryar Mohri. Learning with rejection. In *International Conference on Algorithmic Learning Theory*, pages 67–82. Springer, 2016.

[21] Corinna Cortes, Giulia DeSalvo, Mehryar Mohri, and Scott Yang. On-line learning with abstention. *arXiv preprint arXiv:1703.03478*, 2017.

[22] Claudio De Stefano, Carlo Sansone, and Mario Vento. To reject or not to reject: that is the question-an answer in case of neural classifiers. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(1):84–94, 2000.

[23] Christopher P Diehl and John B Hampshire. Real-time object classification and novelty detection for collaborative video surveillance. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, volume 3, pages 2620–2625. IEEE, 2002.

[24] Tom Diethe, Tom Borchert, Eno Thereska, Borja Balle, and Neil Lawrence. Continual learning in practice. *arXiv preprint arXiv:1903.05202*, 2019.

[25] DC Dowson and BV Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.

[26] Wenliang Du, Lei Fang, and P Ningi. Lad: Localization anomaly detection for wireless sensor networks. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 15–pp. IEEE, 2005.

[27] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[28] William J Faithfull, Juan J Rodríguez, and Ludmila I Kuncheva. Combining univariate approaches for ensemble change detection in multivariate data. *Information Fusion*, 45:202–214, 2019.

[29] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[30] João Gama, Indrundefined Žliobaitundefined, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4), March 2014. ISSN 0360-0300. doi: 10.1145/2523813. URL `https://doi.org/10.1145/2523813`.

[31] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016.

[32] Jan J Gerbrands. On the relationships between svd, klt and pca. *Pattern recognition*, 14(1-6): 375–381, 1981.

[33] Assaf Glazer, Michael Lindenbaum, and Shaul Markovitch. Learning high-density regions for a generalized kolmogorov-smirnov test in high-dimensional data. *Advances in neural information processing systems*, 25:728–736, 2012.

[34] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[35] Frank E Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1): 1–21, 1969.

[36] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. *Information systems*, 25(5):345–366, 2000.

[37] Greg Hamerly and Charles Elkan. Learning the k in k-means. *Advances in neural information processing systems*, 16:281–288, 2003.

[38] Ahsanul Haque, Latifur Khan, and Michael Baron. Sand: Semi-supervised adaptive novel class detection and classification over data stream. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[40] Guy G Helmer, Johnny SK Wong, Vasant Honavar, and Les Miller. Intelligent agents for intrusion detection. In *1998 IEEE Information Technology Conference, Information Environment for the Future (Cat. No. 98EX228)*, pages 121–124. IEEE, 1998.

[41] John L Hodges. The significance probability of the smirnov two-sample test. *Arkiv för Matematik*, 3 (5):469–486, 1958.

[42] Hanqing Hu, Mehmed Kantardzic, and Tegjyot S Sethi. No free lunch theorem for concept drift detection in streaming data classification: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(2):e1327, 2020.

[43] Shin-Ying Huang, Jhe-Wei Lin, and Rua-Huan Tsaih. Outlier detection in the concept drifting environment. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 31–37. IEEE, 2016.

[44] Google Inc. mlops: continuous delivery and automation pipelines in machine learning. URL `https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning`.

[45] Samuel Kaski, Jari Kangas, and Teuvo Kohonen. Bibliography of self-organizing map (som) papers: 1981–1997. *Neural computing surveys*, 1(3&4):1–176, 1998.

[46] Eamonn Keogh, Jessica Lin, Sang-Hee Lee, and Helga Van Herle. Finding the most unusual time series subsequence: algorithms and applications. *Knowledge and Information Systems*, 11(1):1–27, 2007.

[47] Janis Klaise, Arnaud Van Looveren, Clive Cox, Giovanni Vacanti, and Alexandru Coca. Monitoring and explainability of models in production. *arXiv preprint arXiv:2007.06299*, 2020.

[48] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

[49] Aleksandar Lazarevic, Levent Ertoz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 2003 SIAM international conference on data mining*, pages 25–36. SIAM, 2003.

[50] Erich Leo Lehmann, Joseph P Romano, and George Casella. *Testing statistical hypotheses*, volume 3. Springer, 2005.

[51] Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[52] David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests. *arXiv preprint arXiv:1610.06545*, 2016.

[53] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf`.

[54] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, 2(1):56–67, 2020.

[55] Bruno Iran Ferreira Maciel, Silas Garrido Teixeira Carvalho Santos, and Roberto Souto Maior Barros. A lightweight concept drift detection ensemble. In *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1061–1068. IEEE, 2015.

[56] Markos Markou and Sameer Singh. Novelty detection: a review—part 1: statistical approaches. *Signal processing*, 83(12):2481–2497, 2003.

[57] Leandro L Minku and Xin Yao. Ddd: A new ensemble approach for dealing with concept drift. *IEEE transactions on knowledge and data engineering*, 24(4):619–633, 2011.

[58] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[59] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6): 47–60, 1996.

[60] Jose G. Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521–530, 2012. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2011.06.019. URL `https://www.sciencedirect.com/science/article/pii/S0031320311002901`.

[61] Hai-Long Nguyen, Yew-Kwong Woon, Wee-Keong Ng, and Li Wan. Heterogeneous ensemble for feature drifts in data streams. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 1–12. Springer, 2012.

[62] Kyosuke Nishida and Koichiro Yamauchi. Detecting concept drift using statistical testing. In *International conference on discovery science*, pages 264–269. Springer, 2007.

[63] Mark Nixon and Alberto Aguado. *Feature extraction and image processing for computer vision*. Academic press, 2019.

[64] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

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

[66] Stephan Rabanser, Stephan Günnemann, and Zachary C Lipton. Failing loudly: An empirical study of methods for detecting dataset shift. *arXiv preprint arXiv:1810.11953*, 2018.

[67] G Naga Ramadevi, K Usha Rani, and D Lavanya. Importance of feature extraction for classification of breast cancer datasets, a study. *International Journal of Scientific and Innovative Mathematical Research*, 3(2):763–368, 2015.

[68] Siqi Ren, Bo Liao, Wen Zhu, and Keqin Li. Knowledge-maximized ensemble algorithm for different types of concept drift. *Information Sciences*, 430:261–281, 2018.

[69] Geoffrey Holmes Bernhard Pfahringer Peter Reutemann, Ian H Witten Mark Hall, Eibe Frank, and Ian H Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[70] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[71] Ryne Roady, Tyler L Hayes, Hitesh Vaidya, and Christopher Kanan. Stream-51: Streaming classification and novelty detection from videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 228–229, 2020.

[72] Volker Roth. Outlier detection with one-class kernel fisher discriminants. *Advances in Neural Information Processing Systems*, 17:1169–1176, 2004.

[73] Stan Salvador, Philip Chan, and John Brodie. Learning states and rules for time series anomaly detection. In *FLAIRS conference*, pages 306–311, 2004.

[74] Elena Samuylova. What is the difference between outlier detection and data drift detection?, Nov 2021. URL `https://towardsdatascience.com/what-is-the-difference-between-outlier-detection-and-data-drift-detection-534b903056d4`.

[75] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[76] Abdallah Abbey Sebyala, Temitope Olukemi, Lionel Sacks, and Dr Lionel Sacks. Active platform security through intrusion detection using naive bayesian network for anomaly detection. In *London Communications Symposium*, pages 1–5. Citeseer, 2002.

[77] Alireza Shafaei, Mark Schmidt, and James J Little. A less biased evaluation of out-of-distribution sample detectors. *arXiv preprint arXiv:1809.04729*, 2018.

[78] Eduardo J Spinosa, André Ponce de Leon F. de Carvalho, and João Gama. Olindda: A cluster-based approach for detecting novelty and concept drift in data streams. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 448–452, 2007.

[79] Abhijit Suprem, Joy Arulraj, Calton Pu, and Joao Ferreira. Odin: automated drift detection and recovery in video analytics. *arXiv preprint arXiv:2009.05440*, 2020.

[80] Damian A Tamburri. Sustainable mlops: Trends and challenges. In *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 17–23. IEEE, 2020.

[81] Sasha Targ, Diogo Almeida, and Kevin Lyman. Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*, 2016.

[82] Daniele Cortinovis Lisa Lozza Thomas Viehmann, Luca Antiga. Torchdrift, 2020. URL `https://github.com/TorchDrift/TorchDrift`.

[83] Srikanth Thudumu, Philip Branch, Jiong Jin, and Jugdutt Jack Singh. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, 7(1):1–30, 2020.

[84] J Tian and J Pearl. Causal discovery from changes: a bayesian approach, ucla cognitive systems laboratory. Technical report, Technical Report, 2001.

[85] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[86] Mark Treveil, Nicolas Omont, Clément Stenac, Kenji Lefevre, Du Phan, Joachim Zentici, Adrien Lavoillotte, Makoto Miyazaki, and Lynn Heidmann. *Introducing MLOps*. O'Reilly Media, 2020.

[87] Graham JG Upton. Fisher's exact test. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 155(3):395–402, 1992.

[88] Vladimir N Vapnik. The nature of statistical learning. *Theory*, 1995.

[89] Antanas Verikas and Marija Bacauskiene. Feature selection with neural networks. *Pattern recognition letters*, 23(11):1323–1335, 2002.

[90] Thomas Viehmann. Numerically more stable computation of the p-values for the two-sample kolmogorov-smirnov test. *arXiv preprint arXiv:2102.08037*, 2021.

[91] Thomas Viehmann. Partial wasserstein and maximum mean discrepancy distances for bridging the gap between outlier detection and drift detection. *arXiv preprint arXiv:2106.12893*, 2021.

[92] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[93] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.

[94] Guodong Wang, Anna Ledwoch, Ramin M Hasani, Radu Grosu, and Alexandra Brintrup. A generative neural network model for the quality prediction of work in progress products. *Applied Soft Computing*, 85:105683, 2019.

[95] Graham Williams, Rohan Baxter, Hongxing He, Simon Hawkins, and Lifang Gu. A comparative study of rnn for outlier detection in data mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 709–712. IEEE, 2002.

[96] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[97] Yiteng Zhai, Yew-Soon Ong, and Ivor W Tsang. The emerging" big dimensionality". *IEEE Computational Intelligence Magazine*, 9(3):14–26, 2014.

[98] Kun Zhang, Bernhard Schölkopf, Krikamol Muandet, and Zhikun Wang. Domain adaptation under target and conditional shift. In *International Conference on Machine Learning*, pages 819–827. PMLR, 2013.

[99] Nan Zhang, Shifei Ding, Jian Zhang, and Yu Xue. An overview on restricted boltzmann machines. *Neurocomputing*, 275:1186–1199, 2018.

[100] Ji Zhu, Saharon Rosset, Robert Tibshirani, and Trevor J Hastie. 1-norm support vector machines. In *Advances in neural information processing systems*, page None. Citeseer, 2003.