Acknowledgements

First and foremost, I give my deep thanks to Allah for giving me the opportunity and the strength to accomplish this work.

I would like to thank my supervisors Dr. Hatem Mohammed and Dr. Amgad Monir for their help and support during my work for creating a very inspiring research environment. They have been helpful with background information and have continually encouraged me and helped me with comments on my work. They always had time to discuss new ideas and give feedback on early ideas and research problems.

I would like to thank my family who was a constant source of rising and supporting my spirit. Thanks go out especially to my father, my mother, my wife, and my brothers for support and encouragement.

Finally, special thanks to my faculty, department, and my colleagues.

Thank you everyone,

Ahmed Madkour

December 2024

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER

1

Introduction

Governments and companies are producing vast streams of data and require effective data analytics and machine learning methods to assist in making predictions and decisions promptly. One crucial aspect is the machine learning pipeline, which involves training a prepared dataset to construct a model and subsequently utilizing this model to predict new instance outputs. Figure 1.1 shows that the process entails fetching historical data from the database during the training phase to construct the machine learning model. Then, the system can input new instances from the database to predict the output.

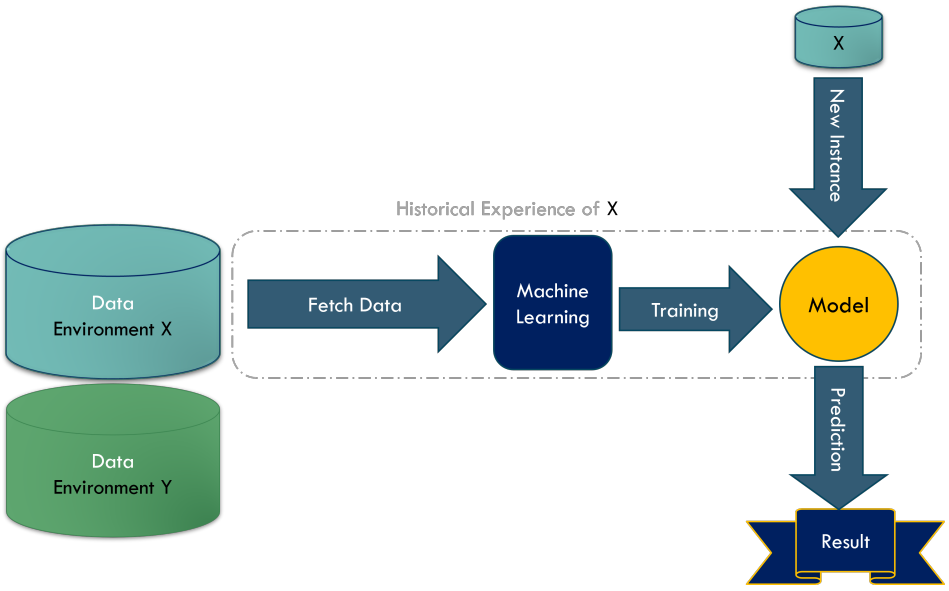


Figure 1.1: Machine Learning Workflow for Environment X.

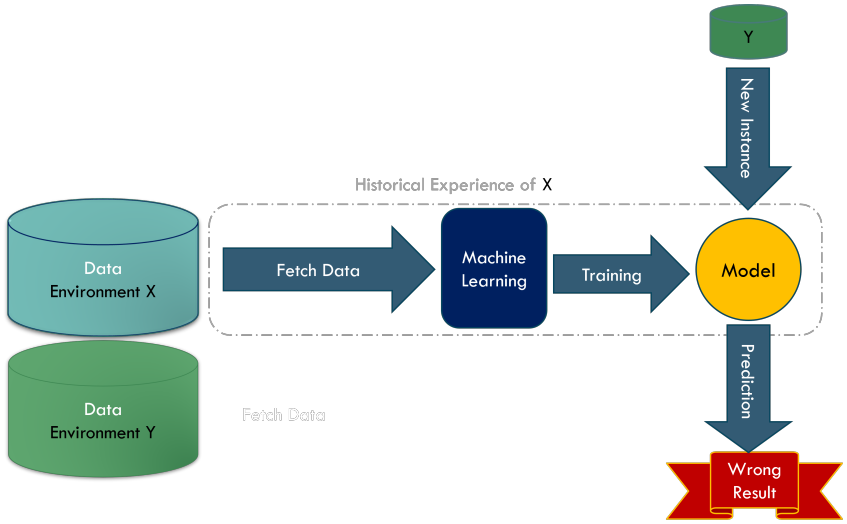
Nevertheless, when endeavoring to forecast outcomes for fresh instances sourced from an alternative database, as illustrated in Fig. 1.2 , there frequently emerges a conspicuous decline in accuracy. This disparity accentuates the imperative for model developers to intervene and rectify the issue. Addressing this, developers must adjust and retrain the model utilizing datasets from the new environment to ameliorate performance. This iterative process aims to refine the model's precision and ensure its efficacy across diverse contexts, thereby bolstering the reliability of decision-making and predictive capabilities. To confront this challenge, the field of auto machine learning endeavors to facilitate online updates to the model without necessitating direct intervention from developers for modification.

In recent years, high-speed data streams have presented significant challenges to machine learning models, particularly in streaming data analysis. These streams, characterized by continuous, dynamic, and high-volume data arrivals, demand adaptive learning algorithms to cope effectively with their evolving nature [1–3]. Among the critical challenges are concept drift, class imbalance, emerging new classes, and heterogeneous transfer learning.

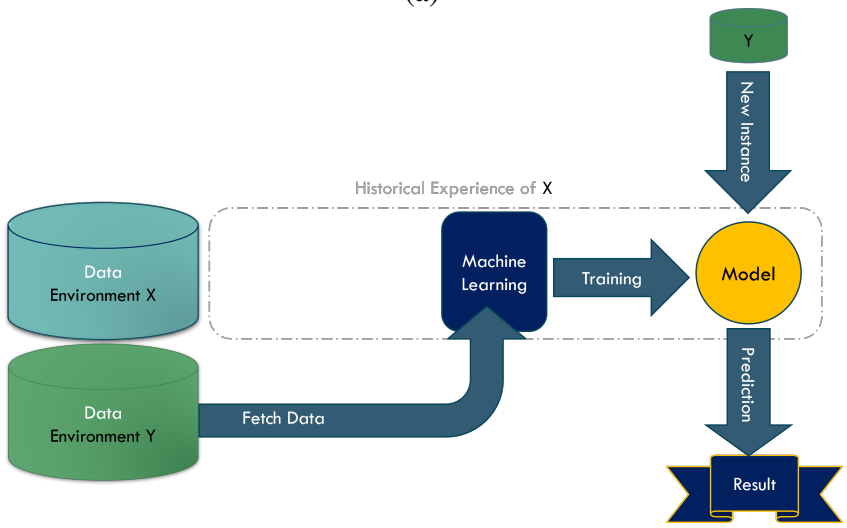
Concept drift refers to changes in the statistical properties of data over time [4, 5], which necessitates continuous adaptation of machine learning models. This phenomenon may manifest as shifts in underlying concepts, relationships between variables, or alterations in data distribution. Addressing concept drift involves employing detection mechanisms that monitor classifier performance or data distribution changes, triggering model updates, retraining, or replacement. These dynamic adjustments ensure model efficacy in the face of evolving data streams.

Class imbalance and class overlap pose additional challenges, particularly in multi-class scenarios. Class imbalance, where data is unevenly distributed across classes [6, 7], often leads to misclassification of underrepresented classes. Techniques such as oversampling, undersampling, algorithm adaptation, and hybrid approaches have been employed to address these challenges [8–11]. Class overlap, where instances from different classes occupy the same data regions [12, 13], complicates distinguishing between classes. Methods like class-overlap undersampling leverage local similarities to mitigate these issues.

Dynamic classifier ensembles, particularly Dynamic Ensemble Selection (DES) methods, offer solutions by adapting ensemble composition based on data charac-



(a)



(b)

Figure 1.2: Machine Learning Workflow for Environment Y.

teristics [14]. These methods optimize classifier subsets through criteria like diversity metrics and performance estimation, ensuring a balance between accuracy and computational efficiency [15]. DES approaches can dynamically select the most competent classifiers for each data instance [16–18], enhancing classification in non-stationary streams.

Transfer learning plays a pivotal role in addressing dynamic data streams and

concept drift, focusing on leveraging knowledge from source domains to improve target domain learning [4, 6]. Strategies include instance re-weighting, feature matching, and mitigating negative transfer effects to bridge domain gaps and improve adaptability.

Finally, the challenge of Streams with Emerging New Classes (SENC) involves scenarios where new classes, absent during initial training, emerge in data streams. Traditional models struggle to recognize and integrate these novel classes in real-time, necessitating adaptive learning mechanisms. This underscores the complexity of real-world data streams and highlights the importance of robust frameworks to manage such dynamics.

Overall, addressing these challenges requires integrating dynamic ensembles, adaptive sampling techniques, and transfer learning to ensure accuracy and scalability in non-stationary data environments. Figures and citations support these discussions, emphasizing the critical need for adaptive solutions [19, 20].

In this chapter, the problem definition for this research that naturally arise is discussed in Section 1.1. After this, the the motivation is presented in Section 1.2 . After this, the objectives and research contribution are presented in Sections 1.3 and 1.4. Next, the research plan is summarised in Section 1.5. Finally, the outline of this thesis is presented in 1.6.

1.1 Problem Definition

The rapid growth of high-speed data streams presents significant challenges for traditional machine learning models, particularly in non-stationary environments where data properties, concepts, and distributions evolve over time. These dynamic conditions cause models, initially trained on historical data, to experience a decline in accuracy when confronted with new data. The key challenges in managing non-stationary data streams are:

- **Concept Drift:** Changes in data distributions or relationships between variables over time, causing models to lose relevance and accuracy [1, 2].
- **Multi-class Imbalanced Data:** The uneven distribution of classes in multi-class data streams leads to biased predictions, with minority classes being misclassified [6, 7].

- **Class Overlap:** Instances from different classes occupying the same feature space, making it difficult for models to distinguish between them [12, 13].
- **Emergence of New Classes:** The appearance of new classes that were not present during training, causing instability and degraded performance as traditional models struggle to adapt.
- **Heterogeneous Transfer Learning:** The challenge of transferring knowledge between domains with differing characteristics, risking negative knowledge transfer in non-stationary data environments [4, 6].

These challenges hinder the performance and adaptability of machine learning models in real-time data streams, necessitating the development of advanced frameworks to address these issues effectively.

1.2 Thesis Motivations

The swift progress in machine learning, particularly in real-time data streams, introduces new challenges that demand innovative solutions. This thesis seeks to address the limitations of traditional models in dynamic, non-stationary data environments. The motivations behind this research are as follows:

- **Adapting to Emerging Classes in Real-Time:** New classes within data streams can cause a decline in accuracy. The goal is to develop adaptive mechanisms that quickly incorporate new classes, maintaining system relevance.
- **Proactive Management of Concept Drift:** Concept drift degrades model performance as data properties change. This research focuses on strategies to detect and manage concept drift to maintain model accuracy.
- **Dynamic Optimization of Classifier Ensembles:** To address emerging classes and shifting distributions, the goal is to develop techniques for the real-time optimization of classifier ensembles.
- **Addressing Multi-Class Imbalance:** Imbalanced multi-class distributions often lead to biased classification. The thesis aims to create methods that address class imbalances dynamically, ensuring fair classification.

- **Enhancing Transfer Learning in Non-Stationary Environments:** Transfer learning can suffer from negative transfer in non-stationary settings. The research focuses on frameworks that minimize negative transfer and enhance knowledge sharing across diverse domains.

1.3 Thesis Objectives

The objectives of this research are driven by the critical need to address the following key motivations:

- **Objective 1**. Handling Imbalanced Multiclass Drifted Data and overlapping classes streams.
- **Objective 2**. Addressing Emerging New Classes in Incremental Streams via Concept Drift and K-means Techniques.
- **Objective 3**. Addressing Heterogeneous Transfer Learning Problem in data streams via Concept Drift and Eigenvector Techniques.

1.4 Thesis Contributions

Our research focuses on developing advanced frameworks to manage non-stationary data streams with high accuracy and minimal computational complexity. The key contributions include:

- **Concept Drift Detection and Ensemble Classifier:** Integrating concept drift detection with ensemble classifiers for real-time adaptation in transfer learning.
- **Dynamic Classifier Ensemble for Emerging Classes:** A framework that dynamically adjusts classifiers to address emerging class issues in non-stationary streams.
- **Precise Weighting for Local Classifiers:** A novel method to refine local classifier contributions, enhancing overall ensemble accuracy.
- **Eigenvector-Based Framework for Heterogeneous Transfer Learning:** A framework using eigenvectors to facilitate knowledge transfer across diverse domains, improving performance.

- **Dynamic Adjustment for Multi-class Imbalanced Data:** A method combining drift detection and optimized ensemble selection to improve accuracy in imbalanced streams.
- **Adaptive Class Imbalance Method:** A dynamic approach to select oversampling methods, addressing class overlap in drifted data streams.

1.5 Thesis Plan

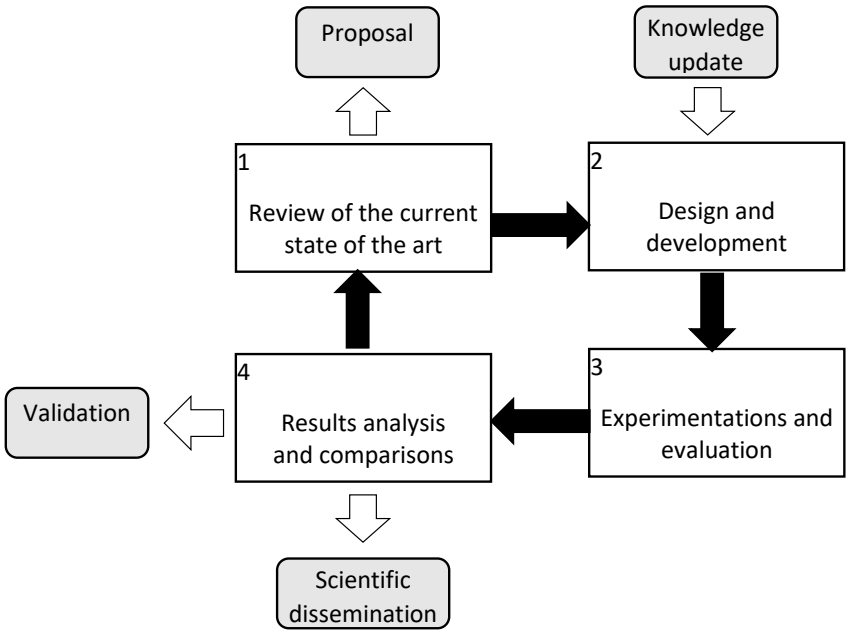


Figure 1.3: The Research Plan of Thesis.

The research in this thesis is progressing rapidly due to technological advancements and continuous contributions in machine learning (ML). An iterative research methodology was followed, where each cycle builds upon the knowledge gained in the previous phase, leading to increasingly effective and original solutions as shown in Fig. 1.3. The phases of this research methodology are as follows:

1. **Review of the current state-of-the-art:** Investigate existing research to identify challenges and inform the design of a solution.
2. **Design and development:** Design a novel solution using updated knowledge to address the identified challenges.

3. **Experimentation and evaluation:** Test the solution through experimentation, using established criteria for comparison.
4. **Results analysis and comparison:** Analyze and compare results with state-of-the-art to determine the effectiveness of the solution, and disseminate the findings.

1.6 Structure of the Dissertation

The structure of the remainder of this thesis dissertation is outlined below:

- **Chapter 2** reviews background about concept drift, concept drift types, concept drift components, adapting types.
- **Chapter 3** reviews state-of-the-art concept drift, classifier ensemble selection, imbalanced data streams, Streams with Emerging New Classes (SENC), and transfer learning.
- **Chapter 4** presents the first proposal to build an effective proposed approch for handling Imbalanced Multi-class Drifted Data streams.
- **Chapter 5** provides the second proposal to adressing emerging new classes in incremental streams via concept drift techniques.
- **Chapter 6** introduces a novel approach to addressing the heterogeneous transfer learning problem in incremental data streams using concept drift techniques.
- **Chapter 7** revisits the main goal and specific objectives posted earlier. This chapter summarizes the main contributions of the thesis and outlines potential directions for future research.

CHAPTER

2

Background

Governments and businesses increasingly rely on advanced data analysis and machine learning tools to predict trends and support decision-making in the face of vast streaming data. However, the dynamic nature of data, driven by the emergence of new markets, products, and behaviors, presents the challenge of concept drift [21]. Concept drift refers to changes in the statistical properties of the target variable over time, which can render historical patterns ineffective, impacting the reliability of decision support and early warning systems. In the context of big data, with its unpredictable distributions and types, addressing concept drift necessitates adaptive, data-driven systems capable of detecting and responding to these changes. This chapter explores drifted streams, the sources, types, and components of concept drift, and discusses the application of the K-Nearest Neighbors (KNN) algorithm in such dynamic environments.

2.1 Drifted Streams

A drifted stream refers to a data stream in which the underlying patterns or relationships between the input features and the target variable change over time. This shift in the data distribution, known as concept drift, can manifest in several ways, including changes in the underlying data distributions, the appearance of new classes, or the disappearance of existing ones. Drifted streams are common in many real-world applications, such as financial markets, sensor networks, and online recommendation systems, where user behavior or environmental conditions evolve.

In a drifted stream, traditional machine learning models may struggle to maintain high predictive accuracy, as they are built on the assumption that the relationships within the data remain static. The challenge in such scenarios is that the model must adapt to the changes in the stream to continue providing reliable predictions. This necessitates the development of adaptive algorithms that can detect concept drift and adjust their learning accordingly.

To handle drifted streams effectively, various methods have been proposed, such as drift detection algorithms (e.g., ADWIN, DDM) and ensemble learning techniques that can dynamically update their models in response to drift. Additionally, algorithms like KNN may need to be adapted to account for these changes by giving more weight to recent data or by periodically retraining on updated portions of the data stream to ensure that the predictions remain relevant.

2.2 Concept Drift Sources

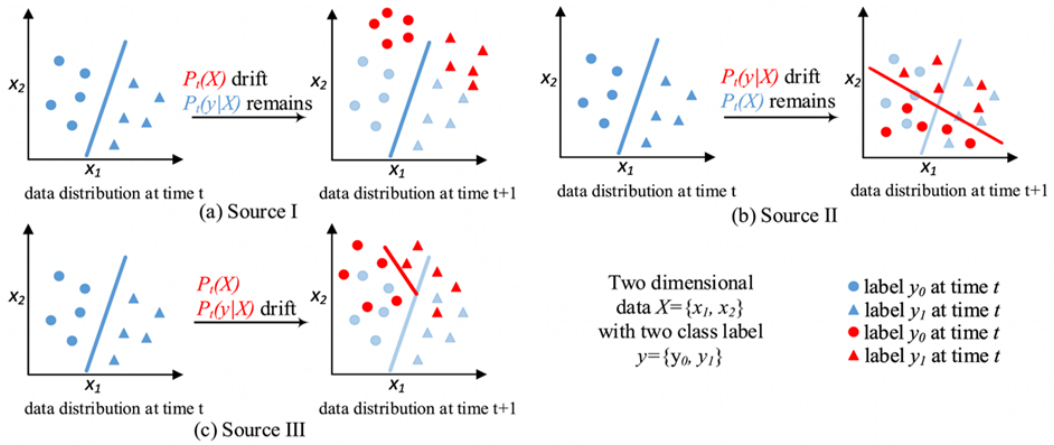


Figure 2.1: Sources of Concept Drift [22].

DOI: 10.1109/TKDE.2018.2876857

Concept drift, as illustrated in Fig. 2.1, occurs in three distinct scenarios. First, (a) depicts a shift in data distribution, indicating changes in the underlying patterns and characteristics of incoming data, challenging the model's ability to adapt to new trends [23–26]. Second, (b) shows a change in function output, requiring adjustments in the class delimiter's position as the relationship between input features

and output classes evolves. Lastly, (c) represents a dual shift in both data distribution and function output, a more complex form of concept drift that requires the model to adapt to simultaneous changes in patterns and output relationships. Effectively managing these shifts is essential for maintaining predictive accuracy and decision-making in dynamic environments, highlighting the importance of adaptive strategies in machine learning models.

2.3 Concept Drift Types

Concept drift is categorized into four types, as shown in Fig. 2.2. Types 1-3 focus on minimizing accuracy loss and maximizing recovery during concept transformation. Type 4, however, emphasizes leveraging historical concepts to identify the best-matched concept during new concept emergence. The term ”intermediate concept,” introduced by [25], refers to transitional phases between concepts. [11] further notes that concept drift can extend over time, with intermediate concepts representing a blend of starting and ending concepts in incremental drift or one of these in gradual drift. Understanding these intermediate concepts is key to grasping the dynamics of concept drift during transitions.

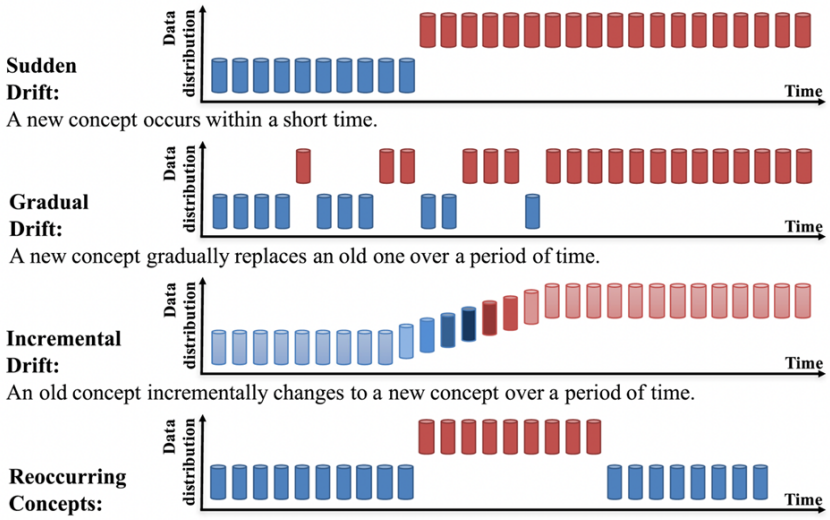


Figure 2.2: Types of Concept Drift [22].

DOI: 10.1109/TKDE.2018.2876857

2.4 Concept Drift Components

Conventional machine learning involves prediction and training, but learning under the concept drift paradigm introduces three additional steps: concept drift detection, drift understanding, and drift adaptation 2.3. Concept drift detection identifies changes in the data distribution, pinpointing when and where drift occurs. Drift understanding analyzes the timing, duration, and location of drift, offering critical insights for the adaptation step. Adaptation, or reaction, involves updating models in response to detected drift. Approaches for drift adaptation include Simple Retraining, Ensemble Retraining, and Model Adjusting. Drift detection methods vary, with some using constant chunk lengths and others employing variable lengths. Drift understanding helps inform decisions on whether to retrain a model from scratch or adjust the existing one, based on the severity of the drift as shown in Fig. 2.4.

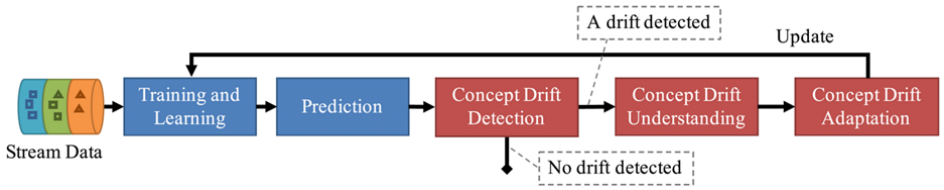


Figure 2.3: Main Components of Concept Drift [22].
DOI: 10.1109/TKDE.2018.2876857

2.4.1 Concept Drift Detection

Drift detection involves techniques and mechanisms to characterize and quantify concept drift by identifying change points or intervals [11]. The general framework for drift detection consists of four stages:

- **Stage 1 (Data Retrieval):** This stage focuses on retrieving data chunks from data streams. Given that a single data instance lacks sufficient information to infer the overall distribution [23], organizing data chunks meaningfully is crucial for effective data stream analysis [27].
- **Stage 2 (Data Modeling):** This optional stage abstracts the retrieved data, extracting key features that contain sensitive information impacting a system

in case of drift. This stage may involve dimensionality reduction or sample size reduction to meet storage and online speed requirements [11].

- **Stage 3 (Test Statistics Calculation):** This stage involves measuring dissimilarity or estimating distance to quantify drift severity and generate test statistics for hypothesis testing. Defining an accurate and robust dissimilarity measurement remains a challenging aspect of concept drift detection. Test statistics can also be used for clustering evaluation [28] and to determine dissimilarity between sample sets [29].
- **Stage 4 (Hypothesis Test):** This stage employs a specific hypothesis test to assess the statistical significance of the change observed in Stage 3, such as the p-value. These tests determine drift detection accuracy by establishing statistical bounds for the test statistics from Stage 3. Without Stage 4, the acquired test statistics are meaningless for drift detection, as they cannot establish the drift confidence interval. Commonly used hypothesis tests include estimating the distribution of test statistics [30, 31], bootstrapping [32, 33], the permutation test [23], and Hoeffding's inequality-based bound identification [34].

Without Stage 1, concept drift detection can be framed as a two-sample test problem, where the objective is to determine if two sample sets come from the same distribution [29]. Any multivariate two-sample test can be used in Stages 2-4 for concept drift detection [29]. However, if distribution drift is not present in the target features, selecting the appropriate target feature becomes crucial for the performance of the learning system, presenting a significant challenge in concept drift detection [35].

2.4.2 Understanding Phase

The severity of concept drift is a key factor in selecting suitable adaptation strategies. Figure 2.4 shows that when concept drift is minimal, resulting in only slight changes to the decision boundary, incremental learning is sufficient. However, with high severity, where the decision boundary experiences substantial shifts, retraining a new model may be more effective than incrementally updating the existing one. While some researchers have emphasized the ability to quantify drift

severity, this information is not widely applied in adaptation practices. The adaptation process offers three main approaches: Simple Retraining, where a new model is trained with recent data; Model Ensemble, which retains and reuses existing models for repeated concept drift; and Model Adjusting, which allows partial updates to a model as the data distribution changes.

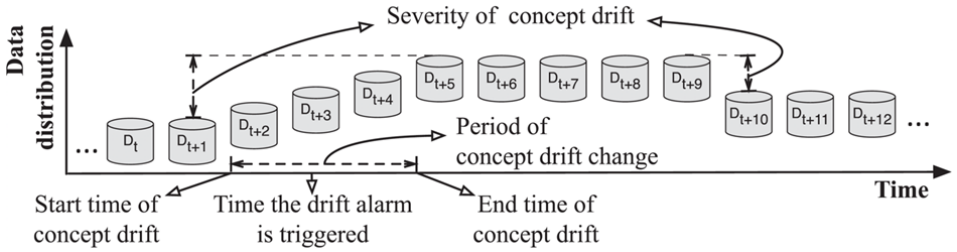


Figure 2.4: Understanding Phase of Concept Drift [22].
DOI: 10.1109/TKDE.2018.2876857

2.4.3 Adaptation Phase

This section delves into strategies for updating existing learning models in response to drift, referred to as drift adaptation or reaction. The three primary categories of drift adaptation methods are simple retraining, ensemble retraining, and model adjusting, each tailored to address specific types of drift.

A. Simple Retraining

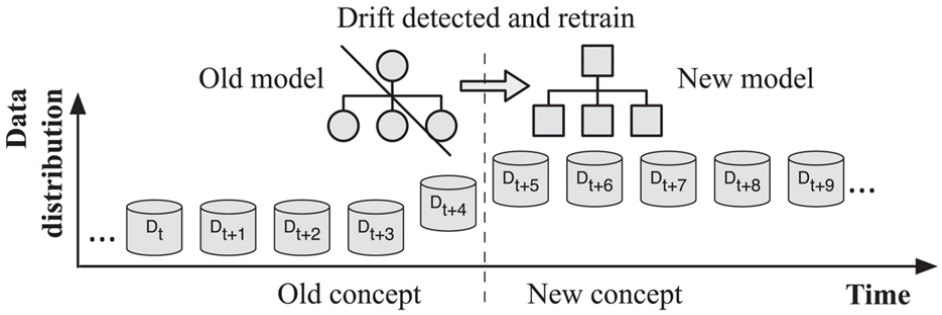


Figure 2.5: Approach for Retraining a New Model [22].
DOI: 10.1109/TKDE.2018.2876857

To handle concept drift, models can be retrained with the latest data, with a concept drift detector determining when retraining is necessary as shown in Fig. 2.5. A common method is the window strategy, where recent data is stored for retraining, and old data is used for drift detection. Paired Learners [36] use a stable and a reactive learner to replace the stable one when it misclassifies instances correctly classified by the reactive learner, though choosing the appropriate window size can be challenging. ADWIN [37] improves this by dynamically adjusting window sizes based on change rates.

B. **Model Ensemble Eetraining**

In recurring concept drift scenarios, ensemble methods, which reuse old models instead of retraining new ones, offer greater efficiency and adaptability [38]. These methods involve combining classifiers with varied types or parameters using voting rules. Adaptive ensemble techniques, including those based on classical methods, are designed to address concept drift challenges (Fig. 2.6). Classical methods like Bagging, Boosting, and Random Forests have been adapted for concept drift. Online Bagging [39] and Bagging with ADWIN [40] replace underperforming classifiers when drift is detected. Adaptive Boosting [41] and the Adaptive Random Forest (ARF) algorithm [42] also incorporate drift detection.

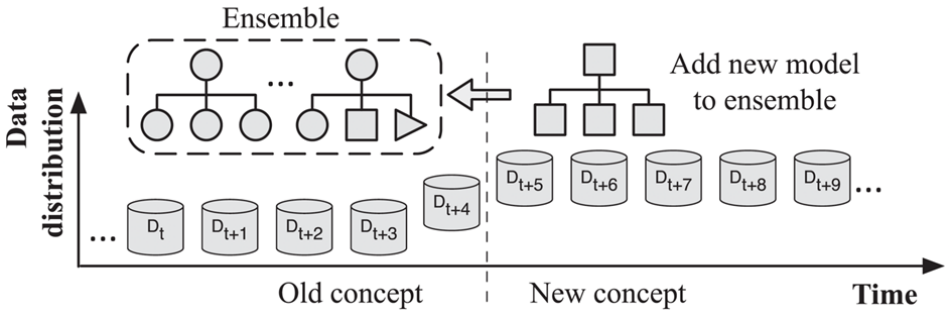


Figure 2.6: Ensemble Approach for the Adaptation Phase [22].
DOI: 10.1109/TKDE.2018.2876857

C. **Model Adjusting**

An alternative to retraining models entirely is using adaptive learning for partial updates in response to concept drift, particularly in localized regions [43]. Decision trees are effective due to their adaptability, with methods like

VFDT [44] using the Hoeffding bound for fast processing without storing instances. CVFDT [45] enhances VFDT by managing concept drift with a sliding data window and replacing sub-trees for better performance. Extensions [46, 47] improve classifier selection with adaptive leaf strategies, while recent critiques [48, 49] challenge VFDT's use of the Hoeffding bound and suggest alternative impurity measures.

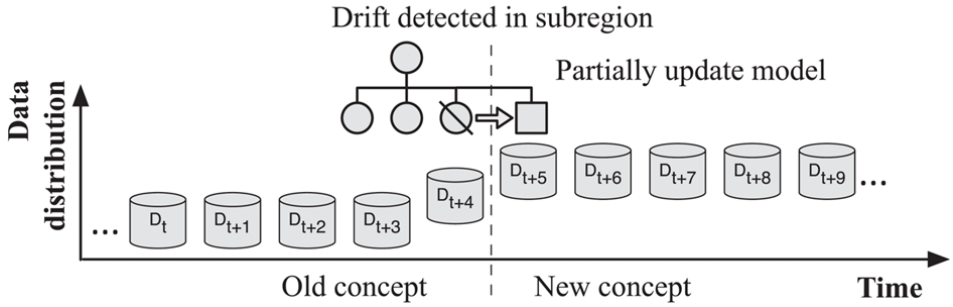


Figure 2.7: Partial Updating Approach for the Adaptation Phase [22].
DOI: 10.1109/TKDE.2018.2876857

2.5 K-Nearest Neighbors (KNN) Algorithm

The K-Nearest Neighbors [50] (KNN) algorithm is a simple, yet effective, method used in machine learning for classification and regression tasks. It operates on the principle of identifying the "K" closest data points to a given query point in the feature space and making predictions based on the majority class (for classification) or the average value (for regression) of those neighbors. The main advantages of KNN are its ease of implementation and ability to handle both linear and non-linear decision boundaries without requiring an explicit model to be built. In the context of streaming data, where new data points are constantly being added, KNN is often applied to predict the class of an incoming data point by comparing it to the most recent observations in the stream. However, one of the challenges when using KNN in dynamic environments is dealing with concept drift. Since KNN relies on the most recent data points to make predictions, its performance can degrade significantly when the statistical properties of the stream change over time, as is the case with concept drift.

CHAPTER

3

State-of-the-art

This chapter provides an in-depth review of recent advancements in stream classification, addressing key challenges in dynamic data environments, such as concept drift, imbalanced multiclass scenarios, class overlap, ensemble selection, new class emergence, and transfer learning. Section 3.1 explores methodologies for real-time concept drift detection and adaptation to maintain classification accuracy. Section 3.2 focuses on dynamic classifier ensemble selection strategies to enhance performance in evolving data streams. Section 3.3 tackles imbalanced multiclass problems, discussing oversampling techniques to balance minority classes in drifted streams. Section 3.4 examines methods for integrating new classes into existing systems to improve adaptability, while Section 3.5 highlights the role of transfer learning in leveraging knowledge from related tasks to improve performance, particularly in scenarios with limited labeled data. Section 3.6 provides a comparative analysis of recent works, evaluating contributions, limitations, and research gaps, and Section 3.7 concludes the chapter by outlining critical challenges and directions for future research.

3.1 Concept Drift

Concept drift refers to changes in the underlying data distribution over time, which can reduce the accuracy of previously trained machine learning models [51–53]. Detecting and responding to concept drift is crucial for maintaining model performance. Several detection methods have been proposed to address this challenge. The Drift Detection Method (DDM) [31, 40] uses a statistical test to identify

significant error rate increases, signaling concept drift. The Early Drift Detection Method (EDDM) [31, 54] extends DDM by considering a moving window of recent data. ADWIN [31, 54] employs a sliding window to monitor statistical differences and adjusts the window size to adapt to drift patterns. The Kolmogorov-Smirnov windowing method (KSWIN) [54] calculates the Kolmogorov-Smirnov distance to detect drift, while Hoeffding's bounds with moving average test (HDDMA) and its variant HDDMW [31, 40] compute bounds for the true mean to detect distribution changes. Lastly, the Page-Hinkley method [55] tracks the cumulative sum of errors, detecting drift when the sum exceeds a threshold. These methods enable machine learning models to adapt to evolving data streams, enhancing model performance and robustness.

3.2 Classifier Ensemble Selection

This study examines the overproduce-and-select approach for classifier ensemble selection [15, 18, 56]. The goal is to identify the optimal subset of classifiers from a larger ensemble, considering factors like performance, diversity metrics, meta-learning techniques, and performance estimation. This process reduces computational complexity, enhances efficiency, and improves ensemble performance, particularly for real-world applications. By selecting a smaller subset, the approach balances accuracy with computational resources and adapts to evolving data streams. There are two main selection strategies: static and dynamic. Static selection assigns classifiers to specific feature space partitions, while dynamic selection chooses classifiers based on local competencies for each data sample. Dynamic Ensemble Selection (DES) selects the best classifiers for each test instance, considering their competence in the local region. The Randomized Reference Classifier, proposed by Woloszynski and Kurzynski [16], enhances adaptability and robustness by introducing randomness through a beta distribution. This classifier is particularly effective in concept drift scenarios. However, using diversity measures, as shown by Lysiak [17], may lead to smaller ensembles but not necessarily improved accuracy. Overall, the overproduce-and-select approach offers a framework for tackling concept drift by dynamically adapting the ensemble composition,

improving classification performance, efficiency, and adaptability in dynamic environments.

3.3 Imbalanced data Streams

In imbalanced data classification, three main approaches have been identified [57], with this study emphasizing the first category, which addresses imbalanced data streams through sampling methods, particularly oversampling [58]. This technique creates synthetic instances to achieve balanced class distributions [59–62]. Imbalances can arise in both binary and multi-class settings, with this research concentrating on multi-class oversampling strategies.

To address multi-class imbalances, Multi-Label SMOTE (MLSMOTE) [9] extends SMOTE to multi-class learning by generating synthetic examples for minority class labels and ensuring their proper assignment. A more recent technique, Multi-Label Synthetic Oversampling based on Local Label Imbalance (MLSOL) [57], improves upon MLSMOTE by targeting local imbalances within multi-class classification. MLSOL uses distinct sampling strategies for each label, offering superior performance in classification accuracy and other metrics. It generates synthetic samples from minority class instances in a restricted neighborhood, improving computational efficiency and reducing overfitting, making it a promising technique that outperforms MLSMOTE in several areas.

3.4 Streams with Emerging New Classes (SENC)

Existing methods for detecting new class emergence in streaming data include clustering-based approaches , which require true labels, limiting their practical use. Tree-based methods, such as SENCForest [63] and SEEN [64], employ anomaly detection but suffer from high false positives and inefficiencies. SENNE [65] improves detection using nearest neighbor ensembles but lacks model retirement mechanisms, causing longer runtimes. The KNNENS [50] method combines a k-nearest neighbor ensemble with model updates to address new class detection and classification. However, existing methods generally overlook the use of concept

drift techniques, highlighting the need for approaches that can handle both concept drift and new class emergence.

3.5 Transfer Learning

Transfer learning has gained significant attention for addressing distribution disparities between source and target domains, with methods falling into two categories: instance re-weighting and feature matching [66]. Instance re-weighting methods, such as Kernel Mean Matching (KMM) [67], Kullback-Leibler Importance Estimation Procedure (KLIEP) [68], and TrAdaBoost [69], adjust the weights of source instances to align with the target domain. These methods have been extended to multisource transfer learning (e.g., MsTrAdaBoost [70]). Feature matching approaches, including Transfer Component Analysis (TCA) [70] and CORAL [71], focus on aligning feature representations between domains, often through transformations that minimize distribution differences. However, negative transfer, where transferred knowledge harms performance, is a challenge addressed by techniques like Transfer Joint Matching (TJM) [72]. Additionally, methods such as HE-CDTL [73] handle concept drift in transfer learning by incorporating historical knowledge and class-wise weighted ensembles. Finally, online transfer learning techniques like Melanie [70] manage non-stationary environments by dynamically adjusting model weights to accommodate concept drift.

3.6 Comparsion

This section presents a critical comparison of closely related works addressing the challenges of imbalanced multiclass streams 3.6.1, the emergence of new classes 3.6.2, and the integration of transfer learning 3.6.3 within streaming environments. The increasing complexity of real-world data streams necessitates advanced methodologies that can effectively manage the intricacies of these challenges. By examining various approaches in the literature, the goal is to highlight their contributions, strengths, and limitations in dealing with imbalanced data distributions, adapting to new class occurrences, and leveraging transfer learning techniques. This comparative analysis highlights the current state of research while

emphasizing specific gaps and unresolved challenges, paving the way for more robust and adaptive solutions in streaming data classification.

3.6.1 Imbalanced Stream

Addressing class imbalances is critical in multi-class classification. Multi-Label SMOTE (MLSMOTE) [9] enhances classifier performance by generating synthetic examples for minority class labels using neighboring examples in the feature space. However, Multi-Label Synthetic Oversampling based on Local label imbalance (MLSOL) [74] improves upon this by employing tailored sampling strategies for each label to address local imbalances. Research shows that MLSOL outperforms MLSMOTE in classification accuracy and computational efficiency by generating synthetic samples from minority instances within restricted neighborhoods, resulting in a more compact and efficient dataset while reducing overfitting. Figure 3.1 illustrates that MLSOL is more likely to select $x1$ as a seed instance because it is surrounded by more neighbors of the opposite class for $l3$. MLSMOTE assigns the label vector [0,1,0] to all synthetic instances based on their neighbors. In contrast, MLSOL creates more diverse instances by assigning labels according to their location. Moreover, synthetic instances $c2$ and $c3$ generated by MLSMOTE introduce noise, whereas MLSOL copies the labels of the nearest instance to the new examples. In summary, MLSMOTE tends to generate new instances biased toward the dominant class in the local area, whereas MLSOL effectively explores and exploits both the feature and label space.

Table 3.1 compares MLSMOTE and MLSOL, two methods for addressing imbalanced data in multi-class classification. MLSMOTE generates synthetic examples for minority classes to balance distributions but may blur class distinctions and struggles with overlapping classes, leading to misclassification. MLSOL improves upon this by using localized sampling strategies to address local imbalances more effectively. However, both methods face limitations with overlapping class boundaries, which impact their overall classification accuracy. Despite MLSOL's precision in handling local imbalances, the challenge of overlapping classes remains significant for both approaches.

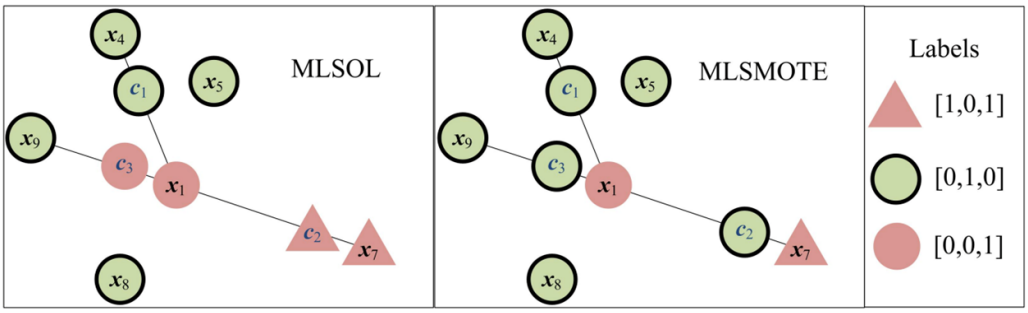


Figure 3.1: Comparsion of MLSMOTE and MLSOL Generated Instances [74].
DOI: 110.1016/j.patcog.2021.108294

Table 3.1: Comparison of the MLSMOTE and MLSOL Methods.

Method	Theory	Advantages	Limitations
MLSMOTE [9]	MLSMOTE significantly enhances classifier performance by generating synthetic examples for each minority class label.	Generating synthetic examples for each minority class label.	• Random synthetic samples may be related to the majority class. • Overlapping classes.
MLSOL [74]	MLSOL systematically combats local imbalances within the domain of multi-class classification by employing distinct sampling strategies for each label.	Generating synthetic examples for each minority class label within a restricted neighborhood.	• Overlapping classes.

3.6.2 Emergence of new classes

Detecting and adapting to new classes in streaming data is essential for maintaining classification accuracy. Tree-based methods like SENCForest [63] and SEEN [64] use anomaly detection but suffer from high false positive rates and

inefficiencies. SENNE improves detection using a nearest neighbor ensemble but has longer runtimes due to ineffective model retirement. The k-Nearest Neighbor Ensemble-based method (KNNENS) [50] enhances new class detection and known class classification through hypersphere ensembles and dynamic model updates. However, all these methods struggle to handle concept drift effectively, which is critical for detecting new classes and updating classification models.

Figures 3.2, 3.3, and 3.4 illustrate key approaches for classifying emerging and known classes. SENCForest divides the space into three regions (normal, outlying, and anomaly) and detects new classes using threshold path lengths. SENNE uses hyperplanes in three dimensions ($x1$, $x2$, and $x3$) to classify instances as emerging or known based on class rankings. KNNENS employs hyperplanes for all class samples and uses a voting mechanism to classify instances as emerging or known. These visualizations emphasize the differences in how SENNE and KNNENS handle class classification.

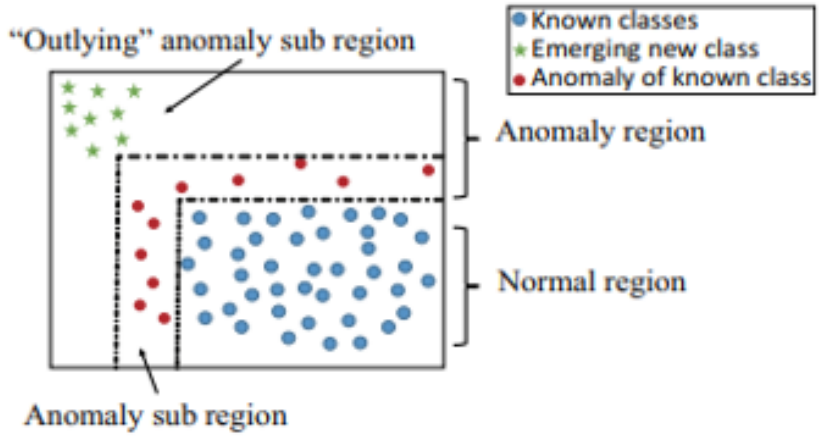


Figure 3.2: Overview of SENCForest Detection Flow Diagram [63].
DOI: 10.1109/TKDE.2017.2691702

Table 3.2 compares three emerging class detection methods: SENCForest, SENNE, and KNNENS. SENCForest uses iForest [20] for anomaly detection and a threshold path for identifying new classes, serving as both an unsupervised detector and supervised classifier, but it is prone to false positives and relies on a complex threshold mechanism. SENNE utilizes a nearest neighbor-based hypersphere ensemble

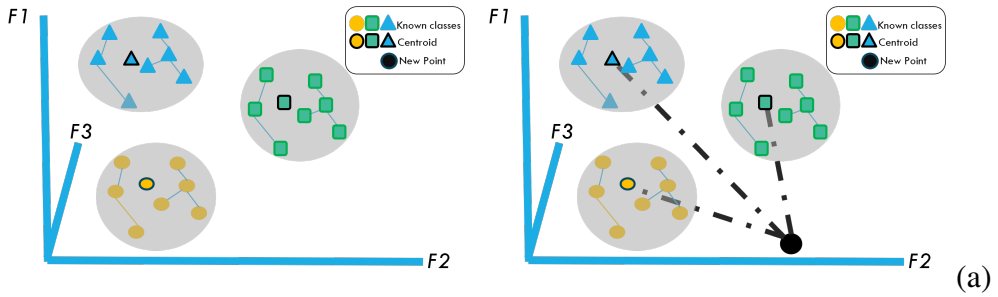


Figure 3.3: Overview of the Stream Emerging Nearest Neighbor Ensemble (SENNE) [64].

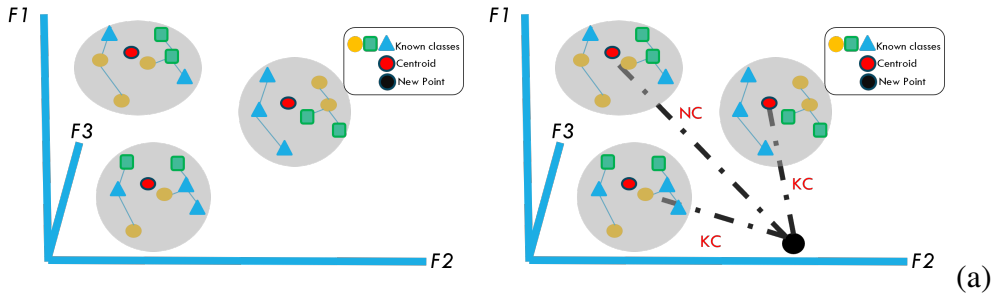


Figure 3.4: Overview of the k-nearest Neighbor Ensemble-based [50] (KENNE).

to analyze local neighborhood information, effectively handling varying geometric distances between classes, though it assumes static class distributions and has lengthy update times. KNNENS improves by using a hypersphere ensemble for all classes, reducing false positives and enabling updates without true labels, but it shares SENNE's limitation of assuming unchanged known class distributions.

3.6.3 Transfer Learning

In transfer learning, CORAL, Melanie, and HE-CDTL are key approaches related to the third proposed method 6. CORAL [70] aligns sub-space bases through second-order statistics using a learned transformation matrix, minimizing domain discrepancies and negative transfer. Melanie [2] employs an online ensemble learn-

Table 3.2: Comparison of the SENCForest, SENNE, and KENNE methods.

Method	Theory	Advantages	Limitations
SENCForst [63]	employs anomaly detection method iForest for a new class detection and then applies threshold path to detect the anomalies.	SENCForest serves as both an unsupervised anomaly detector and a supervised classifier.	• Potential for High false positives. • Dependency on path length threshold (more complexity).
SENNE [64]	nearest neighbor-based hypersphere of one class ensemble to explore local neighborhood information and sort distance to calculate distance.	SENNE is able to handle both the low and high geometric distance between two classes in the feature space.	• Assumes that the distribution of known classes remains unchanged. • Take long time for update.
KENNE [50]	nearest neighbor-based hypersphere of all class ensemble to explore local neighborhood information.	KNNENS to reduce false positives for the new class. KNNENS does not require true labels to update the model.	• Assumes that the distribution of known classes remains unchanged.

ing strategy to address non-stationary environments by incrementally training models from source and target domains, dynamically adjusting weights, and combining models via a weighted-sum approach. HE-CDTL extends these concepts specifically for Concept Drift Transfer Learning (CDTL), leveraging historical and source domain knowledge through a class-wise weighted ensemble and AW-CORAL to reduce domain disparities. Experiments show HE-CDTL outperforms baseline methods, demonstrating its efficacy in managing transfer learning under concept drift. Table 3.3 compares CORAL, Melanie, and HE-CDTL for transfer learning. CORAL minimizes domain discrepancies and reduces negative transfer by projecting source data into the target domain using a transformation matrix and Singular

Value Decomposition (SVD) but struggles with non-stationary and heterogeneous data. Melanie addresses non-stationary environments in online learning by dynamically training and combining models from source and target domains, yet faces challenges with the complexity of online learning and data heterogeneity. HE-CDTL reduces domain shifts by aligning second-order statistics and leveraging historical knowledge but depends on source domain quality and also encounters issues with heterogeneous data.

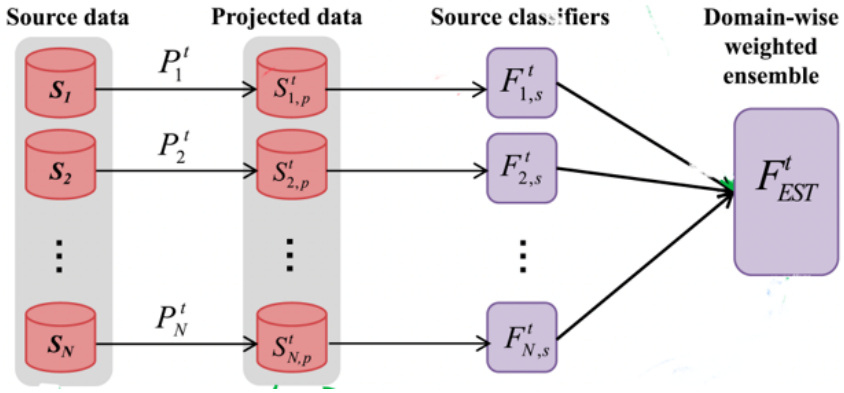


Figure 3.5: Overview of CORrelation ALignment (CORAL)[70].

3.7 Related Works Challenges

By comparing the literature on ensemble learning for classification tasks, the proposals in this thesis differ from other studies in several ways:

- As highlighted in the literature on imbalanced streams, most studies have concentrated on generating synthetic samples while ignoring class overlap. *To address this challenge*, an approach is proposed to generate non-overlapping classes in imbalanced streams.

- Oversampling techniques often perform inefficiently in the presence of concept drift. *To tackle this issue*, a methodology is introduced that selects the oversampling technique based on the current and historical distribution of the stream chunks.

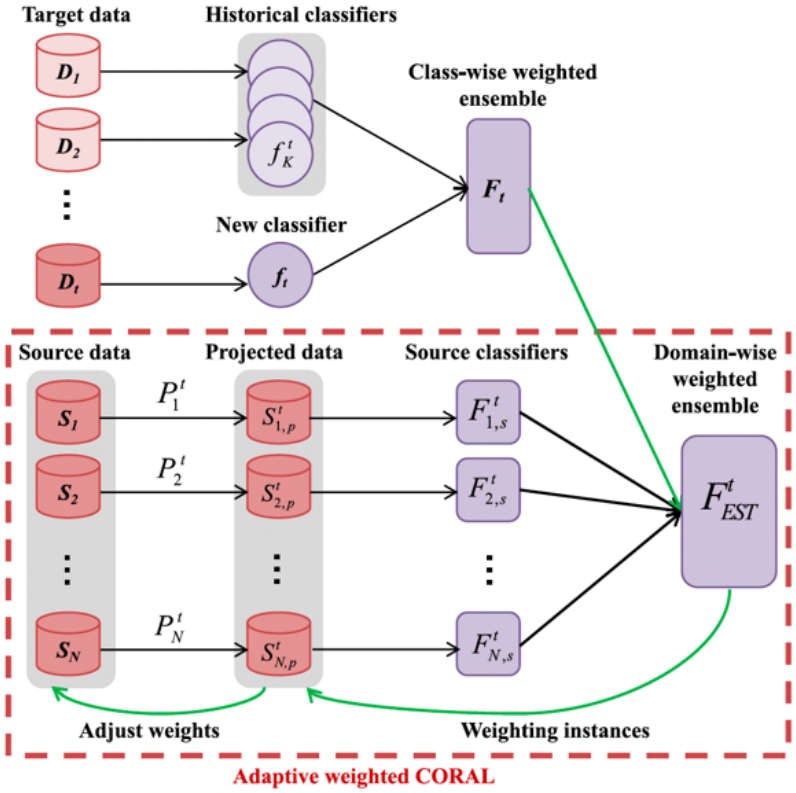


Figure 3.6: Overview of Concept Drift Transfer Learning (CDTL) [70].

- As highlighted in the literature on non-stationary environments reveals that most works focus on detecting emerging new classes while overlooking distribution changes. *To overcome this challenge*, a combined approach is proposed that utilizes Dynamic Ensemble Selection (DES) to select the best classifier for each chunk based on stream distribution, k-means clustering, and concept drift to address both emerging new class detection and distribution changes.

- As highlighted in the literature on transfer learning, most studies focus on homogeneous multisource transfer and neglect heterogeneous multisources in non-stationary environments are observed. *To resolve this issue*, a combined approach is proposed that integrates Dynamic Ensemble Selection (DES), Concept Drift Transfer Learning (CDTL), eigenvector techniques, and con-

Table 3.3: Comparison of the CORAL, Malanie, and CDTL Methods.

Method	Theory	Advantages	Limitations
CORAL [70]	Correlation Alignment (CORAL) uses a learned transformation matrix and Singular Value Decomposition (SVD) to project the source instances into the target domain.	CORAL can minimize domain discrepancy across source and target domains, meanwhile reducing the negative knowledge transfer.	• Non-stationary environments. • Heterogenous multisource.
Melanie [2]	Multi-sourcE onLine TrAnsfer learning for Non-statIonary Environments (Melanie). utilize the class-wise weighted .	It considers an online problem in which the data in source and target domains are generated from non-stationary environments.	• Based on the online learning only. • Heterogenous multisource.
HE-CDTL [70]	HE-CDTL uses the class-wise weighted and domain wise ensemble for historical knowledge and reduce the disparities between the source and target domains .	HE-CDTL minimizes domain shift by aligning the second-order statistics of source and target distributions.	• Depend on source domain quality. • Heterogenous multisource.

cept drift to address heterogeneous transfer learning in non-stationary environments.

CHAPTER

4

Dynamic Classification Ensembles for Handling Imbalanced Multiclass Drifted Data Streams

In recent years, the explosion of high-speed data streams has presented new challenges for machine learning models. Three critical issues that have emerged are concept drift, class imbalance, and class overlap. Concept drift refers to the phenomenon where the statistical properties of a data generation process change over time [1, 2]. This signifies that the underlying concepts, relationships between variables, or data distribution can change, leading to a fundamental shift in the nature of data. Dealing with concept drift poses a fundamental challenge in machine learning and data mining. This can cause models trained on historical data to become inadequate when applied to new data affected by concept drift, leading to a decline in the model performance [2]. To address this issue, concept drift detectors are used to identify changes in data stream distributions by leveraging information associated with classifier performance or the incoming data items themselves. These signals frequently prompt model updates, retraining, or substitution of an old model with a new one. In addition, class imbalance [2, 4], which is characterized by uneven class distribution, poses a challenge for traditional classifiers [5], particularly in multiclass scenarios where minority class samples are at risk of

misclassification owing to their limited representation [6]. Addressing imbalanced data classification requires specialized techniques to ensure accurate minority class classification without compromising the performance of the majority class [7–9]. This challenge becomes more daunting when minority class instances are scattered in unknown configurations, thereby increasing the likelihood of overfitting during the learning process. To address this issue, three primary methods are employed in the context of imbalanced data classification, which are effective in both binary and multiclass imbalance scenarios [10]. The first approach involves sampling methods that address class imbalance by either reducing the number of majority class instances (undersampling) or generating artificial minority class instances (oversampling) [11, 75]. The second and third groups encompass adaptive algorithms and hybrid methods, respectively. Adaptive algorithms include one-class and cost-sensitive classification [76]. Hybrid methods merge data preprocessing with classification techniques, often utilizing ensemble classifiers to effectively mitigate class imbalance and enhance classifier performance [12, 19, 20]. Class overlap occurs when instances from different classes inhabit the same region in the data space [13, 14]. This overlap complicates the task of distinguishing between representative instances of various classes and posing performance challenges for traditional classifiers. This issue is commonly referred to as class overlap. Researchers have proposed class overlap undersampling techniques to address class imbalance problems [15]. These techniques aim to leverage local similarities among minority instances to identify potentially overlapping majority instances. Although these methods have demonstrated promising results in improving model performance on specific datasets, many of them rely heavily on nearest-neighbor approaches to detect overlapping regions around minority instances. This approach often neglects the global similarity within the overlapping domain, which can lead to local optimal values getting stuck during calculations. Furthermore, determining appropriate parameters for these models poses a significant challenge. If the parameter selection is too extensive, it may lead to the exclusion of valuable instances, while conservative parameters may overlook instances with overlap. This issue can significantly affect classifier performance, particularly when handling streams containing both a minority class and instances with class overlap. Therefore, both class imbalance and class overlap present significant challenges in the realm of data stream analyses.

Consequently, addressing class imbalance is crucial in multiclass learning, leading to research efforts that focus on both concept drift and class imbalance challenges. Researchers have explored techniques such as DES and multiclass oversampling to address these issues. Dynamic classifier ensembles adapt their composition based on data characteristics, making them particularly valuable in evolving data conditions [16]. Classifier ensemble selection aims to identify the optimal subset of classifiers from a larger ensemble, often using the overproduce-and-select strategy. This process considers various criteria, including individual performance metrics, diversity measures, meta-learning techniques, and performance estimation methods, which are crucial for balancing accuracy with computational constraints. Ensemble selection methods are categorized as static or dynamic. Static selection assigns classifiers to predefined feature partitions, while dynamic selection adapts classifier choices based on their competency [17]. Dynamic selection includes two approaches: Dynamic Classifier Selection (DCS) and Dynamic Ensemble Selection (DES). DCS selects the most suitable classifier for each data point by evaluating local competencies, whereas DES identifies the best-performing classifiers for instances within localized regions [18, 21, 23].

The main goal of this study is to formulate a precise classification approach that addresses changing conditions. Specifically, the first proposed approach aims to address scenarios where there is an uneven distribution among several classes, overlapping instances of classes, and instances where the fundamental concept of data evolves. To address these challenges, dynamic classifier ensembles are employed. These ensembles utilize oversampling techniques, implemented either on a global or local scale, as a preprocessing step to address class imbalance. Furthermore, multiclass learning techniques are enhanced to counteract class imbalance through method adaptation.

The remainder of this chapter is organized as follows: In Section 4.1,the motivations and the contributions are presented. The first proposed framework are discussed in detail in Section 4.2. The experimental results and the discussion are presented in Sections 4.4. Section 4.5 presents a summary of this chapter.

4.1 Motivations and Contributions of this Chapter

1. A classification approach is introduced that dynamically adjusts to multiclass imbalanced data, incorporates mechanisms for detecting concept drift, and optimizes classifier ensemble selection. The objective is to enhance the classification performance, specifically for multiclass imbalanced non-stationary streams.
2. Additionally, an adaptive method is proposed for the class imbalance issue, considering the data distributions and historical instances of class imbalance. This is particularly relevant in cases where class overlap occurs within the multiclass and drifted datan performance by selecting the most suitable oversampling method based on the unique characteristics of the data stream.

4.2 Proposed Methodology

This section outlines the primary phases of the study, comprising three distinct stages. Following this introduction, the synthetic data-generator method is explored in detail, offering a comprehensive breakdown of the four essential steps. Our proposal aims to develop a robust approach designed to overcome the challenging domain of multiclass classification for imbalanced and drifting data streams. To achieve this objective, the proposed solution tackles the four primary challenges associated with developing this approach.

- **Multi-class imbalanced streams:** This study focuses on addressing the widespread problem of imbalanced data streams in the context of multiple classes, aiming to tackle this issue using well-known techniques, particularly MLSMOTE and MLSOL.

- **Overlapping class:** A critical challenge involving class overlapping, which is known to substantially affect model performance [18, 21], is also addressed. An adaptive method is introduced to generate non-overlapping synthetic instances, thereby enhancing the overall performance of the model.

- **Drifted streams:** First proposed approach integrates a concept drift detector to identify shifts in the underlying data distribution. This dynamic detection

mechanism enables the model to promptly recognize changes and adjust its classifiers, thereby ensuring its effectiveness in handling drifting stream.

- **Classifier performance:** To enhance the classifier performance, tfirst approach employs Dynamic Ensemble Selection (DES). This technique creates a pool of classifiers and dynamically selects the most suitable classifier for each incoming data point, further improving classification performance and robustness.

4.2.1 First Proposed Approach (PA1) Description

First proposed approach is designed with three distinct phases that work together to improve its performance in managing multiclass imbalanced and drifting data streams.

- **DES phase (dynamic ensemble selection phase):** The first phase, known as the dynamic ensemble selection (DES) phase, is responsible for selecting the most appropriate classifier for the incoming data. This ensures that the selected classifier is well-suited for the current data chunk.

- **Drift detector phase:** The second phase of the first approach is the drift detector phase, which operates in real-time to continuously monitor the data stream. Its primary function is to identify any signs of concept drift, which indicates shifts in the underlying data distribution over time.

- **Synthetic data generator phase:** The final phase of the first approach is the synthetic data generator phase, which is dedicated to generating synthetic data for the minority classes. This step is crucial for addressing class imbalance by producing additional samples for underrepresented classes, thereby significantly enhancing the model's ability to accurately classify instances from minority classes.

- **Synthetic data generator phase:** The final phase of the first approach is the synthetic data generator phase, which is dedicated to generating synthetic data for the minority classes. This step is crucial for addressing class imbalance by producing additional samples for underrepresented classes, thereby

significantly enhancing the model's ability to accurately classify instances from minority classes.

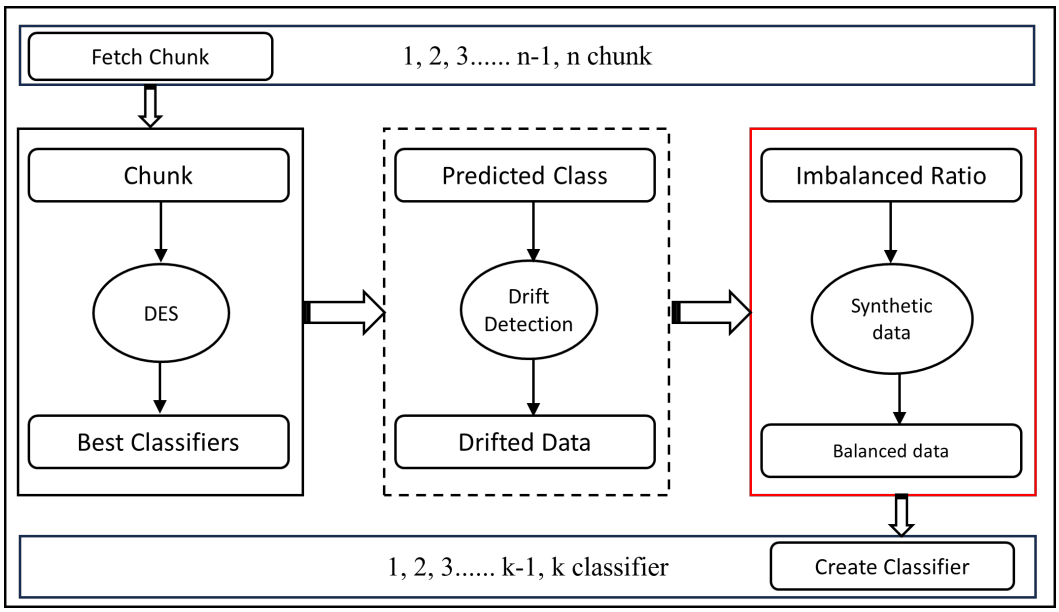


Figure 4.1: First Proposed Approach (PA1) for Imbalanced Multi-class Drifted Streams.

Specifically, as shown in Fig. 4.1, the DES phase retrieves the current data chunk from the stream and applies the DES technique to select the most suitable classifiers for the received chunk. The selected classifiers were then passed to the second phase, where they were employed to predict the class of each instance within the received data chunk. Simultaneously, detectors like ADWIND or DDM are employed to monitor any occurrence of concept drift. If the discrepancy between the class frequency and standard deviation of the current chunk is significant, as described in reference [31], and the imbalance ratio exceeds the average imbalance ratio, the current chunk is forwarded to the third phase, as indicated by the red rectangle in Fig. 4.1. In the third phase, first proposed method uses a set of equations 4.14.24.3 to identify minority classes. The initial equation computes the frequency of each class within the current chunk, where i represents the current chunk, c denotes the input class, and y_i refers to the predicted class. The second equation

determines the optimal frequency for each class based on the size of the chunk (n) and the number of classes in the current chunk(C).

Algorithm 1: First Proposed Approach for Imbalanced Multi-class Streams.

Input: data stream, maximum classifiers pool size κ
Output: Prediction P

$\psi, \Psi, \Omega, \mu \leftarrow \emptyset$;
$\omega \leftarrow 0$;
for *stream have chunk* **do**
  if a *is the First chunk* **then**
    $k \leftarrow$ `trainingNewClassifier`(a);
    $P \leftarrow$ `getPrediction`(a, k);
  else
    $k \leftarrow$ `DES`(a, Ψ);
    $P \leftarrow$ `getPrediction`(a, k);
    $\psi \leftarrow$ `conceptDriftDetector`(P);
    if $\psi > 0$ **then**
      $\Omega \leftarrow$ get classes frequency according to Eq.1;
      $\omega \leftarrow$ best frequency according to Eq.2;
      $\mu \leftarrow$ get minority classes according to Eq.3;
      $b \leftarrow$ utilize a and μ to get the synthetic data according to Algorithm 2;
      trainingData $\leftarrow a + b$;
      $k \leftarrow$ `trainingNewClassifier`(trainingData);
      $\Psi \leftarrow \Psi + k$;
      if $\Psi > \kappa$ **then**
        `removeWorstClassifier`(Ω);
    $P \leftarrow$ `getPrediction`(a, k);
return P

Finally, the third equation identifies the classes as minority classes if their frequency deviates significantly from the standard deviation of the current chunk, where sd_c represents the standard deviation of the class instances, and frq_i refers to the standard deviation of the current chunk. Figure 4.2 shows that phase These identified minority classes are then fed into the synthetic data generator phase, which increases the minority class samples to balance any imbalanced chunks. This ensures optimal performance for the new classifiers. Algorithm 1 provides a comprehensive outline of the process of the first proposed approach, which is the

main contribution of this study and is designed to effectively address multiclass imbalanced and drifting data streams, uses streaming data as input, and systematically executes each step within the approach. The outcome of this process is the classification prediction generated using the first proposed approach.

4.2.2 Synthetic Data Generator Phase Description

Figure 4.2 presents a comprehensive overview of the synthetic data generator phase, which is an essential component responsible for generating synthetic samples by considering both data distribution and historical chunk behaviors. This phase has several advantages and can perform a wide range of tasks.

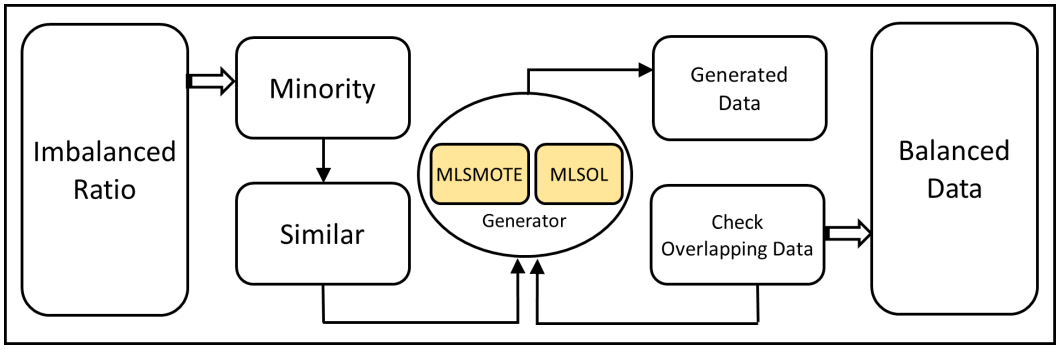


Figure 4.2: Flow Diagram of the Synthetic Data Generator.

- **Similar chunk analysis:** Initially, the phase analyzes the current chunk distribution and identifies a similar chunk from historical data. This analysis forms the basis for generating synthetic samples that align with prevailing distribution patterns.

- **Oversampling method selection:** This phase utilizes the knowledge of the oversampling technique applied to the identified similar chunk. Consequently, it employs an alternative oversampling technique, using MLSMOTE and MLSOL, to create the most effective synthetic data. This step is designed not only to optimize the current classification but also to preemptively address potential drifts in similar future chunks.

- **Class overlap validation:** This step involves generating synthetic samples for the minority classes to effectively address the issue of minority classes and consequently enhance the classifier performance. The process utilizes the K-Nearest Neighbor (KNN) algorithm, as applied in [23], to identify overlaps between newly generated data instances and existing instances. If overlaps are detected, the first proposed approach iteratively removes these samples because their presence can potentially diminish the overall classifier performance [18, 21]. Consequently, the first proposed approach generates alternative samples to address this challenge and preserve the primary objectives of the synthetic data generator step.

- **Continuous refinement:** This process iterates until it successfully generates high-quality synthetic data that aligns with the data distribution, minimizes overlap, and mitigates potential concept drift. The generated data were subsequently utilized in the training phase to improve classifier performance.

Algorithm 2: Synthetic Data Generator Algorithm.

Input: Minority classes μ, current chunk a, sample size η, historical chunks h
Output: Generated data b
$b \leftarrow \emptyset$;
$f \leftarrow$ MLSMSOTE;
$knn \leftarrow$ kNearestNeighbor(a);
$chunk \leftarrow$ similarChunk(a, h);
$f \leftarrow$ similarChunkOverSamplingMethod$(chunk)$;
if $f =$ *MLSMSOTE* **then**
    $f \leftarrow$ MLSOL;
else
    $f \leftarrow$ MLSMSOTE;
while $|b| < \eta$ **do**
    $p \leftarrow$ generateSyntheticPoint(μ, f);
    $similarPointsClass \leftarrow$ KNN.getKneighbor(b);
    if $similarPointsClass = \mu$ **then**
        $b \leftarrow b \cup \{p\}$;
return b;

In Algorithm 2, also known as the Synthetic Data Generator, three essential elements are inputted: the minority class samples, the current data chunk, and the desired size for generating synthetic data. The primary objective of this algorithm is to produce synthetic data samples. To achieve this, the KNN algorithm is employed to identify any overlapping instances within the current chunk (Line 3). This algorithm utilizes two specific techniques, MLSMOTE [31] and MLSOL [77]. MLSMOTE was chosen for its introduction of randomness during instance generation, reducing its reliance on the local characteristics and distribution of the minority class. This randomization diminishes the likelihood of producing overlapping instances, particularly in cases where minority class instances are situated in overlapping regions. In contrast, MLSOL considers the local behavior of minority classes, resulting in synthetic points that closely resemble the minority class. This approach significantly improves the performance of the classifier (lines 4-10). Additionally, in this algorithm, specifically from lines 11 to 17, these lines are dedicated to generating synthetic instances that ensure non-overlap with existing classes. This procedure depends on the selected oversampling method and utilizes the KNN algorithm to guarantee that the generated instances do not overlap with the existing ones.

4.3 Mathematical Foundation for PA1

The mathematical foundation for PA1 involves key equations designed to identify and classify data chunks based on class frequencies and distributions. These equations are fundamental to analyzing imbalanced data streams effectively.

Equation 4.1 calculates the frequency of a specific class c within a data chunk. For each instance y_i, the frequency frq_c is incremented if the instance belongs to class c:

$$frq_c = \sum_{i=1}^{\text{chunk size}} \begin{cases} 1, & \text{if } y_i = c \\ 0, & \text{otherwise} \end{cases}, \quad i = 1, 2, 3, \ldots \text{chunk size}. \tag{4.1}$$

Equation 4.2 defines the *best frequency* for a chunk, $freq_n$, as the proportion of instances n relative to the total number of classes C:

$$\text{best } freq_n = \frac{|n|}{|C|}. \tag{4.2}$$

Finally, Equation 4.3 determines the type of each class in the chunk (*Minority* or *Majority*) by comparing the difference between the standard deviation (sd_c) and the class frequency (frq_i) against the best frequency of the chunk ($freq_{\text{chunk}}$):

$$\text{classes type}_{\text{chunk}} = \sum_{c=1}^{C} \begin{cases} \text{Minority}, & \text{if } diff(sd_c - frq_i) > \text{best } freq_{\text{chunk}} \\ \text{Majority}, & \text{otherwise.} \end{cases} \tag{4.3}$$

These equations collectively enable precise identification of class types within data chunks, forming a critical component of the PA1 framework.

4.4 Experimental Results

The objective of the experiments conducted in this study was to evaluate the efficacy of multiclass oversampling techniques in enhancing the performance of the first proposed method in imbalanced drifted multiclass classification streams. Our primary goal was to develop a novel approach that combines Dynamic Ensemble Selection (DES) to improve classification performance and robustness in such streams. These experiments yielded valuable insights that could further refine the performance of the first proposed approach and its ability to effectively handle imbalanced data streams. These findings provide a better understanding of the capabilities of the first proposed approach and offer insights into an optimal strategy for tackling minority class issues and concept drift in imbalanced data streams. This study contributes to the advancement of stream mining techniques for generating more accurate and robust classification models in dynamic data stream environments. By addressing the challenges posed by minority classes and concept drift, this study offers valuable insights for improving the performance of the first proposed approach and enhancing the overall efficiency of stream mining.

4.4.1 Experimental setup

The evaluation of the first proposed approach incorporated the utilization of various metrics such as recall, precision, specificity, F1 score, balanced accuracy score (BAC), and geometric mean score (G-mean) [32]. The experimental protocol utilized for evaluation was the test-then-train approach [33], where the classification classifier was trained on a specific data chunk and subsequently evaluated on

the subsequent one. The chunk size was standardized for all utilized data streams to 2,000 instances. Four classification classifiers are employed as base estimators: K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Gaussian Naive Bayes (GNB), and Hoeffding Tree (HT), as implemented in scikit-learn [34]. A pool of classifiers was constructed with a maximum size of L = 8, where the DES selected the best classifier for each chunk. If the pool surpassed the set threshold (L), the classifier with the lowest performance was eliminated. The experiments were conducted using Python programming language, and the source code was publicly available on GitHub[1] . A comparison was conducted between multiclass oversampling techniques (MLSMOTE and MLSOL) and the first proposed approach to demonstrate the effectiveness of the contribution. Additionally, these experiments are conducted that uses two different concept drift detectors, ADWIN [26] and DDM [25], to demonstrate the adaptability and robustness of first proposed approach across varying drift detectors.

4.4.2 Data Streams

In this study, the first approach was assessed using various datasets including benchmark datasets, a real application stream dataset, and synthetic data streams. The Stream-learn Python library was used to conduct the evaluations [29]. Table 4.1 illustrates the benchmark dataset employed in this study, which consists of the Covertype dataset containing 40 features, seven classes, and 581,010 instances. For real application stream evaluation, the Sensor stream dataset was used, which consisted of five features, 58 classes, and 392,600 instances. This represents a real-world application scenario and provides valuable insights into the performance of the first proposed approach in practical settings. Synthetic datasets were generated using Scikit-learn Python library to evaluate the performance of the first proposed approach. The synthetic dataset was designed to simulate data streams and comprised 10 features and four classes divided into 200 chunks of 2,000 instances each. The performance of the first approach was systematically evaluated using these datasets and a stream-learn library. These evaluations provided insights into

[1] https://github.com/Amadkour/dynamic__classification_ensembles_for_handling_imbalanced_multi-class_drifted_data_streams.git

the effectiveness of the first approach in handling different types of data streams, including benchmark datasets, real application streams, and synthetic data streams.

Table 4.1: Summary of Dataset Characteristics Utilized in the PA1 Experimental.

Dataset	Number of Features	Number of Classes	Number of Instances
Covertype dataset[1]	40	7	581,010
Sensor Stream dataset[2]	5	58	392,600
Synthetic stream	8	3	200,000

4.4.3 Analysis of Experimental Results

The performance of the first proposed approach was comprehensively assessed on multiple data streams, considering two distinct concept drift detectors, ADWIN and DDM. To ensure thorough evaluation, six key performance metrics—F1 score, recall, precision, G-mean, specificity, and balanced accuracy—were carefully presented using two visualization diagrams: radar and line. A radar diagram was strategically utilized to provide an overview that effectively depicted the performance of each algorithm across the six metrics. The mean value of each metric was calculated to present the overall performance of each method (MLSMOTE, MLSOL, PA1). It is important to note that the PA1 is represented by red lines.

4.4.3.1 Results on the Benchmark Stream

Figures 4.3 and 4.4 unveil the intricate dynamics of applying MLSMOTE, MLSOL, and PA1 to the Covertype dataset under the guidance of ADWIN and DDM as drift detectors. These figures not only explore the multifaceted nature of classifier ensemble adaptation but also illustrate the critical role of drift detection mechanisms in shaping performance metrics, including G-mean—a pivotal metric that encapsulates the delicate balance between sensitivity (recall) and specificity.

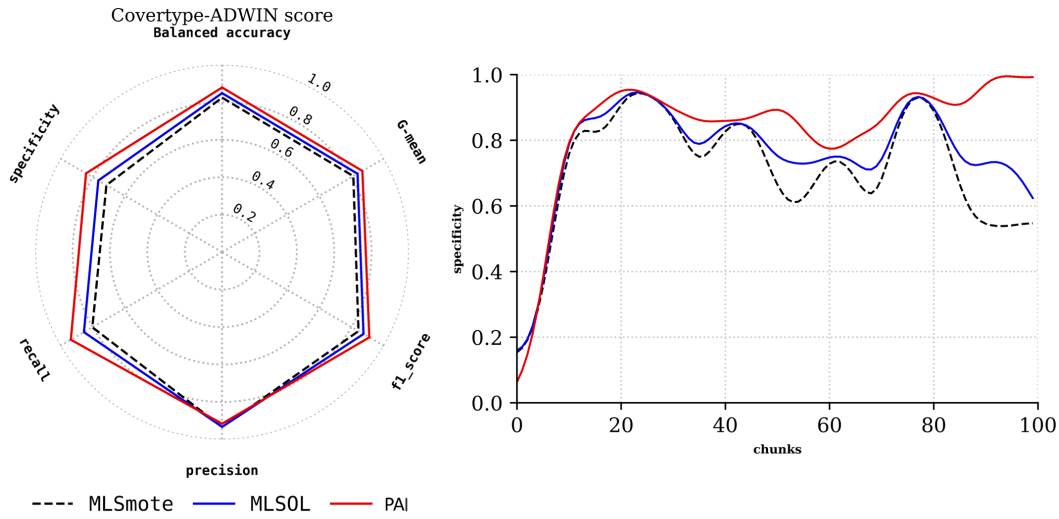


Figure 4.3: Comparison of PA1, MLSMOTE, and MLSOL on Covertype Dataset with ADWIN Concept Drift Detector.

In Figure 4.3, ADWIN's influence is evident as the radar diagram vividly portrays the comparative performance of the three methods across six critical metrics: precision, recall, specificity, F1 score, G-mean, and balanced accuracy. PA1, symbolized by the red line, asserts its dominance by consistently outperforming MLSOL (blue line) and MLSMOTE (dashed black line). Its superior performance in G-mean reflects a methodical synergy between precision and recall, highlighting PA1's ability to reconcile conflicting objectives in stream classification. In contrast, MLSOL offers a more conservative, balanced performance, excelling marginally in specificity but faltering in achieving broader metric optimization. MLSMOTE, by comparison, reveals fundamental limitations, as its deficiencies in recall and balanced accuracy suggest a struggle to maintain consistent adaptability and robustness in evolving data streams. The temporal evolution of classification performance, captured in the line diagram, provides deeper insights into the relationship between the chunk number and specificity values. During the first 20 chunks, all three methods exhibit suboptimal specificity scores, a reflection of the inherent challenges posed by the initialization phase, where the classifier pool is still adapting to the data distribution. However, as the number of chunks increases, PA1 demonstrates a marked improvement in specificity, particularly between chunks 20 and 40, where its strategic use of historical knowledge allows it to dynamically

optimize its classifier ensemble. This ability to leverage past chunks for generating non-overlapping, representative samples underscores PA1's resilience, enabling it to maintain a consistently high specificity throughout subsequent chunks. In contrast, MLSOL exhibits moderate growth in specificity, benefiting from its localized learning strategies but lagging behind PA1 due to its less aggressive approach to historical knowledge utilization. MLSMOTE, however, consistently trails its counterparts, as its limited adaptability and inability to reconcile recall and specificity prevent it from achieving competitive specificity values. The gradual stabilization of specificityvalues beyond chunk 40 for PA1 and MLSOL indicates the maturation of their classifier pools, while the relative stagnation of MLSMOTE underscores its inefficacy in addressing long-term concept drift. Figure 4.4, which incorporates DDM as the drift detector, provides a complementary perspective. The radar plot mirrors the performance hierarchy observed with ADWIN, albeit with a slight degradation in metric values across all methods. This decline reflects DDM's inherent limitations in capturing subtle shifts in data distribution, resulting in less precise ensemble adaptations. The line diagram further reinforces this observation, as specificityvalues exhibit reduced stability compared to ADWIN. While PA1 retains its dominance, the performance gap between PA1, MLSOL, and MLSMOTE narrows, suggesting that the rigidity of DDM diminishes the relative advantage conferred by PA1's historical knowledge strategies. The interplay between chunk number and specificity in Figure 4.4 reveals a similar trend: all methods struggle in the early stages due to limited classifier diversity, but PA1's strategic design enables it to recover more effectively than its counterparts. Despite this, the overall reduction in specificity values highlights DDM's comparative inefficiency in adapting to concept drift, particularly in scenarios characterized by rapid and unpredictable changes. These figures underscore the profound influence of drift detection mechanisms on classifier ensemble performance. ADWIN's demonstrated superiority in fostering stable and adaptive specificity trajectories highlights the importance of dynamic, context-sensitive drift detection in navigating the complexities of evolving data streams. Meanwhile, PA1's sustained excellence exemplifies the philosophical alignment of algorithmic design with the realities of non-stationary environments. Its ability to harmonize sensitivity and specificity, as evidenced by

consistently high specificity values across chunks, marks it as a paradigm for robust and adaptive learning in machine learning research. By contrast, the limitations of MLSMOTE and MLSOL serve as reminders of the challenges inherent in balancing local adaptation with global coherence in dynamic, heterogeneous data environments.

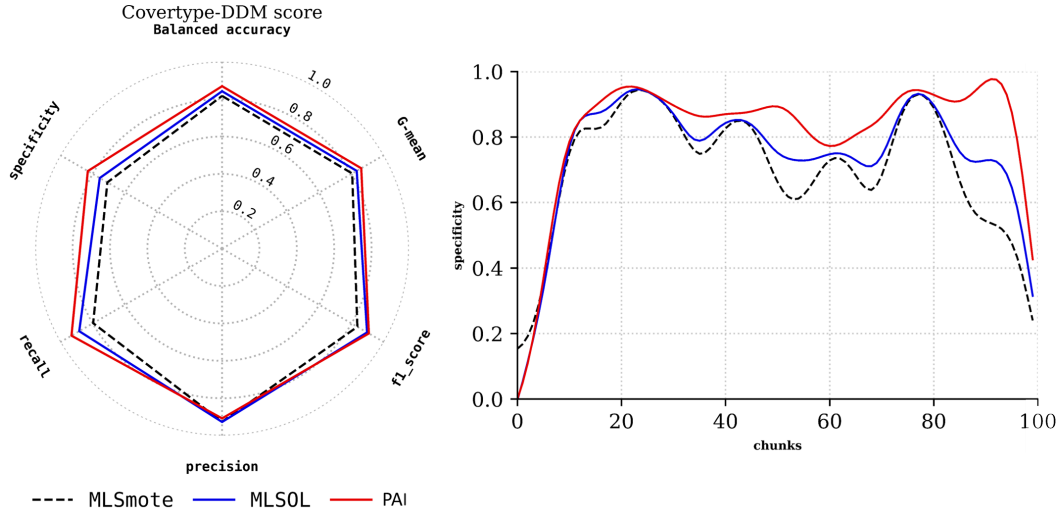


Figure 4.4: Comparison of PA1, MLSMOTE, and MLSOL on Covertype Dataset with DDM Concept Drift Detector.

4.4.3.2 **Results on the Real Application Stream**

Figures 4.5 and 4.6 illuminate the nuanced performance dynamics of PA1, MLSMOTE, and MLSOL on the Sensor data stream, as guided by ADWIN and DDM as concept drift detectors, respectively. These visualizations encapsulate the intricate interplay between the methods and the ever-changing nature of the Sensor data stream, characterized by inherent drift and imbalance. In Figure 4.5, ADWIN, as a drift detector, provides a fertile ground for the methods to showcase their potential. The radar plot reveals that PA1 consistently outperforms MLSMOTE and MLSOL across most metrics, with a pronounced edge in specificity, F1 score, balanced accuracy, and G-mean. This dominance speaks to PA1's philosophical alignment with the complexities of evolving data streams, where achieving equilibrium between sensitivity and precision is paramount. The nearly identical precision and recall values among all methods suggest that MLSMOTE and MLSOL address the

basic challenges of imbalanced streams adequately, yet they lack the strategic depth and adaptability demonstrated by PA1. The temporal line plot in Figure 4.5 adds a temporal dimension to the analysis. It reveals PA1's journey of adaptation, where its early struggle during the initial 30 chunks mirrors the challenge of limited classifier diversity. Beyond chunk 30, PA1's ascent is unmistakable, achieving stability and sustained superiority in specificity as it harmonizes sensitivity and specificity. Meanwhile, MLSMOTE and MLSOL exhibit sporadic fluctuations, with neither achieving the same level of consistent adaptability, underscoring their comparative limitations.

In contrast, Figure 4.6 shifts the lens to DDM as the drift detector. The radar plot reveals a general decline in metric values compared to ADWIN, reflecting DDM's limitations in capturing subtle drift nuances. However, the line plot continues to affirm PA1's resilience and dominance. Despite the overall convergence of performance among the three methods over time, PA1 remains the most stable and adaptable, consistently achieving higher specificity values even under the more rigid framework imposed by DDM. This performance gap highlights the sensitivity of drift detection mechanisms to the data stream's characteristics and emphasizes the importance of matching drift detection strategies with the demands of the learning algorithm.

The juxtaposition of Figures 4.5 and 4.6 serves as a philosophical reflection on the intricate interdependencies between drift detection and classifier performance. ADWIN emerges as a more capable arbiter of change, enabling PA1 to leverage its full potential. In contrast, DDM's rigidity underscores the challenges of achieving optimal adaptability in dynamic environments. Yet, PA1's persistent robustness across both scenarios affirms its role as a paradigm of adaptability and resilience. These findings underscore the importance of harmonizing classifier ensemble design with drift detection mechanisms to navigate the multifaceted challenges posed by real-world, non-stationary, and imbalanced data streams. PA1's ability to sustain high performance across metrics and adapt to diverse drift conditions exemplifies the convergence of algorithmic innovation and the philosophical pursuit of balance in machine learning. In this context, specificity emerges as a profound metric, embodying the trade-offs and synergies required to achieve holistic success in evolving data environments.

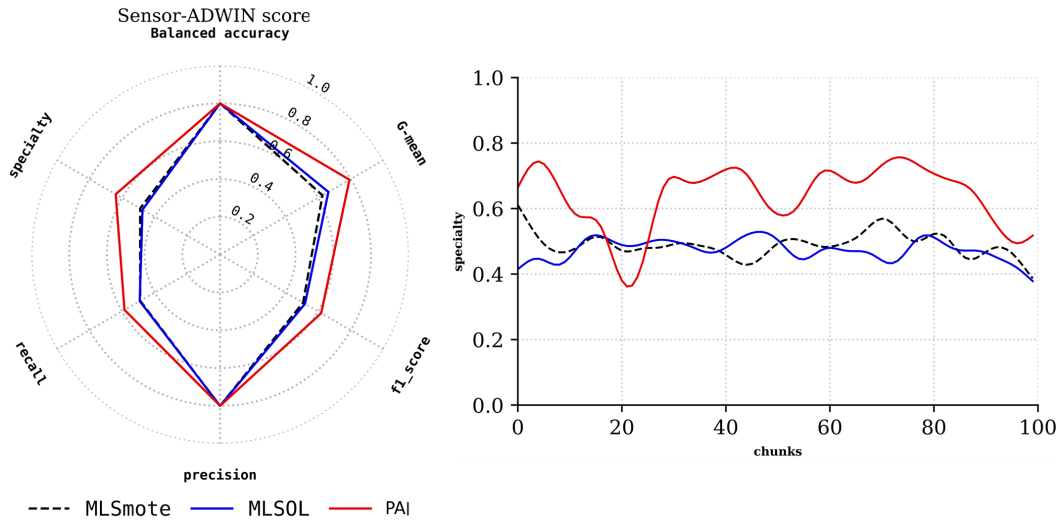


Figure 4.5: Comparison of PA1, MLSMOTE, and MLSOL on Sensor Dataset with ADWIN Concept Drift Detector.

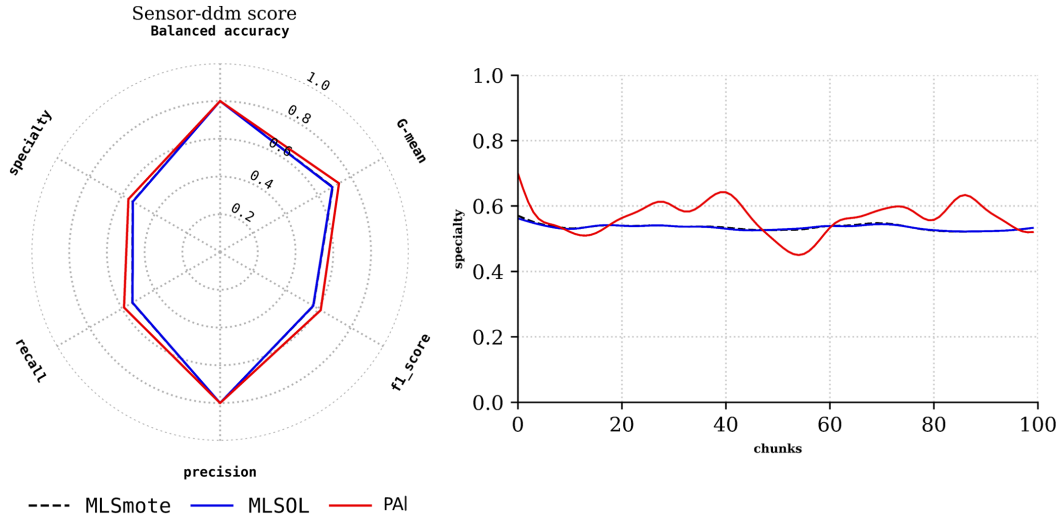


Figure 4.6: Comparison of PA1, MLSMOTE, and MLSOL on Sensor Dataset with DDM Concept Drift Detector

.

4.4.3.3 **Results on the Synthetic Stream**

Figures 4.7 and 4.8 delve into the interplay of algorithms and concept drift detectors on synthetic data streams, offering a profound narrative on adaptability and

resilience amidst frequent and abrupt changes. In Figure 4.7, ADWIN acts as the sentinel of change, facilitating an environment that reveals the nuanced strengths and limitations of each method. The radar plot paints a vivid portrait of metric values clustering between 0.6 and 0.8, symbolizing the relentless nature of synthetic data drifts. Within this chaotic landscape, MLSOL stands out with its precision, an achievement that reflects its singular focus on minimizing false positives. However, this narrow emphasis leaves it vulnerable across other metrics. Conversely, MLSMOTE's consistent underperformance highlights its inability to navigate the complexities of frequent drifts effectively. PA1, in contrast, emerges as a paradigm of balance and adaptability. Its superiority across metrics other than precision is not merely a testament to its robustness but also a reflection of its philosophical alignment with the dynamic nature of non-stationary streams. The transitional phase observed during the first ten chunks, where PA1 struggles, mirrors the natural process of adaptation: a period of recalibration before achieving mastery. Once this initial inertia is overcome, PA1 demonstrates a sustained trajectory of excellence, harmonizing the competing demands of sensitivity and specificity. The line plot becomes a visual metaphor for resilience—PA1 not only adapts to but thrives within the evolving challenges of the data stream.

Figure 4.8 extends this narrative under the gaze of DDM, a drift detector with a contrasting philosophy to ADWIN. While the radar plot reaffirms the earlier insights, it is the subtle degradation in MLSOL and MLSMOTE's performance that becomes a focal point. This decline underscores their dependence on the nuances of the drift detection mechanism, revealing their fragility when faced with an altered evaluative lens. PA1, however, remains steadfast, its consistent superiority reaffirming its ability to transcend the limitations of individual drift detectors. This resilience highlights the algorithm's intrinsic capacity for adaptability, a hallmark of effective learning in uncertain environments. The comparative analysis between ADWIN and DDM unveils a deeper truth: ADWIN's finesse in capturing abrupt and frequent drifts makes it a superior ally in handling synthetic data streams. Yet, the overarching narrative is not about the drift detectors alone but about PA1's versatility and robustness across diverse conditions. Its ability to navigate the complexities of synthetic, Covertype, and Sensor datasets while maintaining high performance reflects a profound balance between precision and recall. PA1 embodies

the ideal of equilibrium, where adaptability and robustness converge to address the multifaceted challenges of real-world, non-stationary data. Through this lens, PA1 transcends its technical role, becoming a philosophical exemplar of learning in the face of change. It illustrates that success in dynamic environments is not solely about excelling in isolated metrics but about harmonizing competing objectives to achieve sustained performance. This balance resonates with the broader challenge of adapting to non-stationarity, offering a roadmap for tackling the inherent uncertainties of real-world applications. PA1, therefore, stands not just as an effective algorithm but as a testament to the principles of adaptability, resilience, and balance that define progress in complex systems.

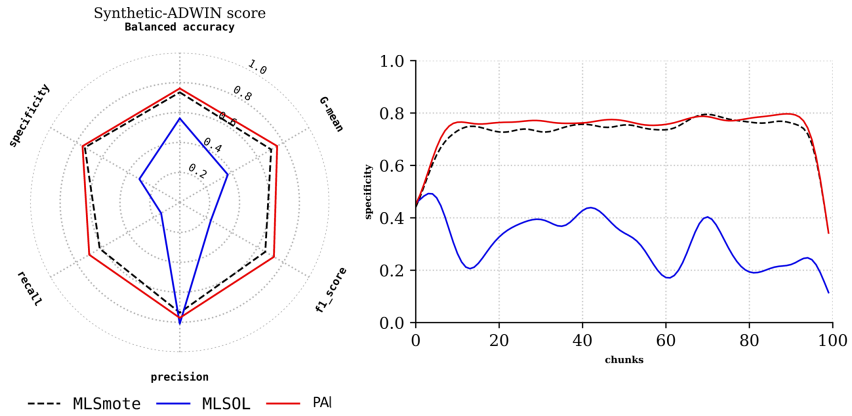


Figure 4.7: Comparison of PA1, MLSMOTE, and MLSOL on Synthetic Stream with ADWIN.

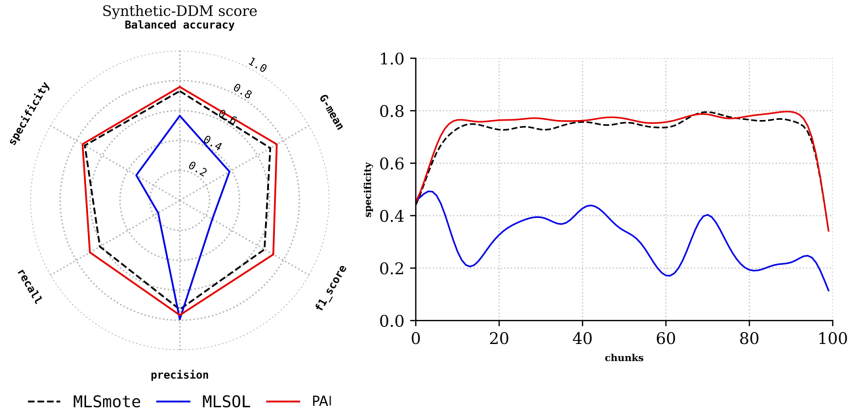


Figure 4.8: Comparison of PA1, MLSMOTE, and MLSOL on Synthetic with DDM.

4.4.4 Analysis of Overlapoing between PA1, MLSMOTE, and MLSOL.

This experimental study focuses on identifying the key factors influencing the selection of optimal approaches for addressing minority class challenges in imbalanced and drifted data streams. The factors under consideration include dataset characteristics, the choice of concept drift detection methods, and the algorithm's capability to minimize class overlap. The concept of overlapping classes refers to situations where instances from different classes in a dataset share similar feature values, making it difficult to distinguish between them. This overlap complicates classification tasks, as the boundaries between classes become less distinct. To evaluate these aspects, the overlapping class behavior of the first proposed approach (PA1) was compared with MLSMOTE and MLSOL through experiments grouped into ten chunks each. Results were visualized using bar diagrams, where each bar represents the number of overlapping samples for ten chunks of various data streams under two drift detectors, ADWIN and DDM. Figures 4.9, 4.10, and 4.11 present the experimental outcomes. Each figure contains ten bar groups, where each group represents overlapping samples for chunks processed by MLSMOTE, MLSOL, and PA1.

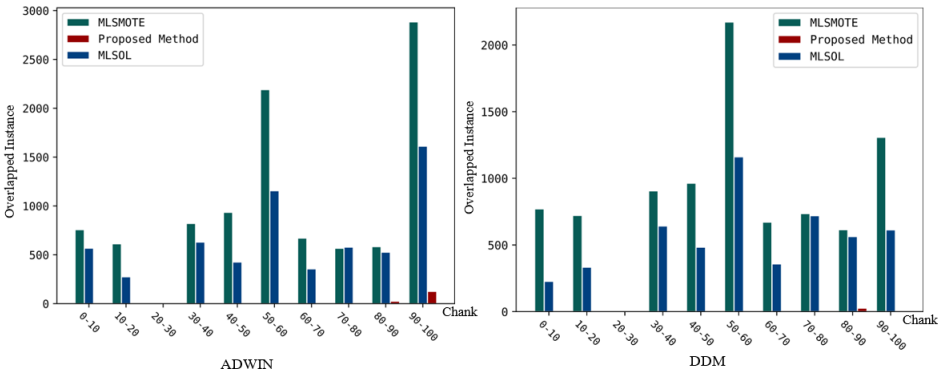


Figure 4.9: Overlapping Points of PA1, MLSMOTE, and MLSOL on Covertype Stream.

Figure 4.9: This figure highlights overlapping samples for a synthetic stream under both ADWIN and DDM drift detectors. The ADWIN diagram shows reduced overlapping samples for chunks 20–30 due to the absence of concept drifts,

which eliminated the generation of synthetic samples. Conversely, chunks 60–70 and 90–100 exhibit the highest number of overlapping samples, corresponding to frequent drifts. Across all chunk groups, MLSMOTE generates the most overlapping samples, while PA1 consistently records the lowest overlap, except in the final group (chunks 90–100). In comparison, the DDM diagram exhibits more overlapping samples than the ADWIN diagram because DDM detects fewer drifts, resulting in prolonged training on outdated models. The highest Y-axis value in the ADWIN diagram reaches 3,000 samples, while in the DDM diagram, it is 2,000 samples, emphasizing the difference in drift sensitivity between the detectors. Figure 4.10: This figure compares the overlapping sample behavior of the three methods on the Sensor stream. The ADWIN diagram demonstrates fewer overlapping samples than in Figure 4.9, indicating a lower number of drifts in the Sensor dataset. The results further confirm PA1's effectiveness in minimizing overlapping samples compared to MLSOL and MLSMOTE, with MLSMOTE consistently exhibiting the highest overlap. For the DDM diagram, the number of overlapping samples is lower than in the ADWIN diagram, reinforcing PA1's advantage in reducing overlap, even under fewer detected drifts.

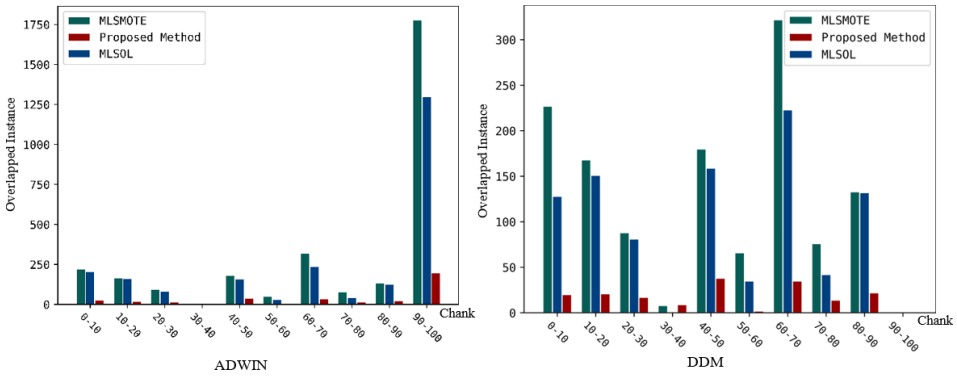


Figure 4.10: Overlapping Points of PA1, MLSMOTE, and MLSOL on Sensor Stream.

Figure 4.11: This figure examines overlapping samples in the synthetic data stream, where the ADWIN and DDM diagrams exhibit the largest number of overlapping samples compared to earlier figures. This indicates that the synthetic stream experienced frequent drifts and greater noise levels. The highest Y-axis value for both diagrams reaches 7,000 samples. Despite this, PA1 maintains the

lowest overlapping sample count across most groups, while MLSMOTE records the highest overlap consistently, regardless of the drift detector employed.

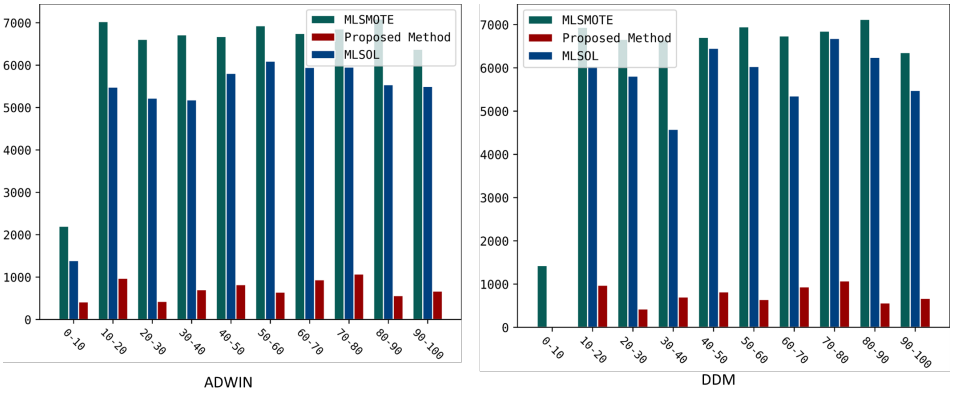


Figure 4.11: Overlapping Points of PA1, MLSMOTE, and MLSOL on Synthetic Stream.

4.4.5 Analyzing Runtime Between PA1, MLSMOTE, and MLSOL.

The runtime of an algorithm encapsulates its operational efficiency, representing the total time consumed during both training and prediction phases over a data stream. Measured in seconds, runtime serves as a critical metric that reflects not just computational performance but also the algorithm's ability to adapt to dynamic challenges like concept drift while delivering predictions.

In Table 4.2, runtime metrics are compared across the proposed approach and other methods, including MLSMOTE and MLSOL, applied to datasets such as Covertype, Sensor, and synthetic data streams. The recorded runtimes provide a clear narrative of computational efficiency. For instance, using the ADWIN concept drift detector, the proposed approach required 8344 seconds to process the Covertype dataset with ensemble classifiers. A slight reduction in runtime was observed when DDM was employed, with the runtime decreasing to 8019 seconds. These observations highlight the algorithm's adaptability and its ability to optimize performance under varying drift detection mechanisms. Comparatively, the proposed approach demonstrates a consistently shorter runtime than MLSMOTE and MLSOL across all datasets and drift detectors. This superiority is emphasized in

the table through bolded runtime values, underscoring the efficiency advantage of the proposed algorithm. Such efficiency stems from its innovative ability to reduce the generation of overlapping samples during training, effectively minimizing redundant computational overhead. This design not only streamlines training processes but also accelerates prediction phases, a critical requirement for real-time data streams. The choice of drift detector further influences runtime dynamics. DDM, characterized by a lower rate of drift detection compared to ADWIN, leads to fewer instances where classifier retraining is triggered. This reduction in training frequency directly contributes to the observed decrease in runtime when DDM is used. However, the slightly higher runtime with ADWIN reflects its sensitivity to detecting more frequent and abrupt concept drifts, which necessitates more frequent retraining of classifiers. While this increases computational demands, it also ensures robust adaptation to rapidly changing data streams.

By reducing computational effort through fewer overlapped samples and aligning seamlessly with drift detection mechanisms, the proposed algorithm establishes itself as a practical solution for handling non-stationary data. This efficiency is particularly vital in real-world applications, where processing speed and responsiveness are as critical as predictive accuracy. The findings emphasize the importance of algorithmic design choices in optimizing runtime, reaffirming the proposed approach's position as a robust, scalable, and efficient solution for real-time data stream processing.

Table 4.2: Runtimes (in seconds) Comparison of PA1, MLSMOTE, and MLSOL.

Stream	**Concept Drift Detector**	**MLSMSOTE**	**MLSOL**	**PA1**
Benchmark	ADWIN	9559	8655	**8344**
	DDM	8388	8031	**8019**
Real Application	ADWIN	1291	1310	**1102**
	DDM	585	607	**521**
Synthetic	ADWIN	12870	4866	**4834**
	DDM	12397	4958	**4687**

4.4.6 Analyzing Non-parametric Tests between PA1, MLSMOTE, and MLSOL.

A comprehensive statistical analysis was performed, encompassing 12 experimental comparisons across three distinct datasets, three algorithms (PA1, MLSMOTE, and MLSOL), and two concept drift detection mechanisms (ADWIN and DDM). To ensure rigor, the Kruskal-Wallis test, a non-parametric statistical method, was employed to evaluate the differences in G-mean performance across the experiments. The results of the Kruskal-Wallis test revealed substantial variations in G-mean values, underscoring the influence of the algorithms and drift detectors on performance outcomes. These observed differences were statistically significant and unlikely to result from random chance [35]. This conclusion was drawn based on the acceptance or rejection of the null hypothesis (H0), which posits that no significant differences exist between the groups. The null hypothesis was accepted in most experiments, indicating statistically significant disparities between expected and observed data. Acceptance of H0 was contingent upon the P-value falling below a critical threshold, which was determined at a 95% confidence level.

Table 4.3: Kruskal-Wallis test results for MLSMOTE, MLSOL, and PA1.

Dataset	Drift detector	Comparison	P-value	Critical value	H0
Covertype stream	ADWIN	PA1 - MLSSMOTE	0.001	0.05	Accept
		PA1 - MLSOL	0.014	0.05	Accept
	DDM	PA1 - MLSSMOTE	0.361	0.05	Rejected
		PA1 - MLSOL	0.401	0.05	Rejected
Sensor stream	ADWIN	PA1 - MLSSMOTE	0.001	0.05	Accept
		PA1 - MLSOL	0.001	0.05	Accept
	DDM	PA1 - MLSSMOTE	0.001	0.05	Accept
		PA1 - MLSOL	0.001	0.05	Accept
Synthetic stream	ADWIN	PA1 - MLSSMOTE	0.001	0.05	Accept
		PA1 - MLSOL	0.001	0.05	Accept
	DDM	PA1 - MLSSMOTE	0.001	0.05	Accept
		PA1 - MLSOL	0.001	0.05	Accept

For precision and clarity, the P-value was rounded to three decimal places. Table 4.3 provides detailed insights into the comparative performances of the three

methods. The statistical results consistently highlighted the superiority of PA1 in handling non-stationary and imbalanced data streams, supported by significant variations in G-mean across most experiments. However, in the third and fourth experiments, a notable similarity in the performance of PA1, MLSMOTE, and MLSOL emerged, leading to the rejection of H0. This convergence in performance can be attributed to the behavior of the DDM drift detector, which identified fewer concept drifts in these particular experiments. The reduced drift detection frequency by DDM likely diminished the opportunities for the algorithms to exhibit their adaptive capabilities, resulting in a more uniform performance across methods. This statistical analysis underscores the nuanced interplay between drift detection mechanisms and algorithmic performance. The ADWIN detector, with its heightened sensitivity to frequent drifts, consistently highlighted performance differences among the methods, enabling a clearer evaluation of their capabilities. In contrast, the less sensitive DDM detector contributed to the observed convergence in performance, emphasizing the impact of drift detection frequency on experimental outcomes. The significant disparities observed in most experiments validate the robustness of PA1 and highlight the necessity of harmonizing algorithmic design with appropriate drift detection mechanisms to address the challenges of non-stationary and imbalanced datasets effectively.

4.5 Summary

This study presents an innovative framework combining oversampling techniques, concept drift detection, and Dynamic Ensemble Selection (DES) to address multiclass imbalanced data streams. Extensive experimentation on benchmark, real-world, and synthetic datasets validates its effectiveness in adapting to evolving class distributions and maintaining high performance. Key features include advanced oversampling to tackle minority class challenges, the use of KNN to avoid overlapping instances, and DES for optimal classifier selection. The approach is distinguished by its adaptability, scalability, and ability to dynamically update models in real-time, ensuring consistent performance and relevance in non-stationary environments.

CHAPTER

5

Addressing Emerging New Classes in Incremental Streams via Concept Drift Techniques

In various real applications, data streams have introduced new challenges to learning algorithms. Data streams are continuous with high volumes of data arriving rapidly and dynamically. This data deluge poses unprecedented challenges for learning algorithms because they must adapt to the dynamic nature of the data environment [1–3]. Among the various research areas in machine learning, sequential learning under data Streams with Emerging New Classes (SENC) has garnered considerable attention because of its practical relevance and unique challenges [78], [79], [64]. SENC refers to a scenario in which new classes that were not present during the initial training of a learning model emerged in the data stream. This poses a significant challenge for traditional learning approaches that are typically designed to handle fixed or predefined class distributions. The ability to effectively recognize and adapt to these novel classes in real-time is crucial for maintaining accurate and up-to-date models. Furthermore, the inherent limitations of data streams, such as limited memory and storage constraints, impose additional complexities in the learning process. Learning algorithms must operate efficiently within these resource constraints to ensure real-time processing and to avoid

overwhelming computational overhead. To tackle challenges in data streams with emerging new classes, Dynamic Ensemble Selection (DES) and Adaptive Windowing (ADWIN) are widely used techniques. DES dynamically adapts ensembles of classifiers by continuously evaluating their performance and selecting the most competent subset for current data. This approach enhances adaptability and ensures improved performance over time by incorporating classifiers best suited to prevailing data conditions. Ineffective classifiers, affected by concept drift or the emergence of new classes, are excluded to avoid degrading overall system performance [15, 18, 56]. ADWIN addresses concept drift, where statistical properties of data evolve, causing a decline in classifier accuracy. It uses sliding windows of varying sizes to monitor statistical measures like mean or variance and detects significant changes in data distribution. When drift is detected, ADWIN updates the ensemble by retraining classifiers or incorporating new ones optimized for the updated data distribution. This adaptability ensures classifiers remain effective and accurate in changing conditions [31, 52, 54]. Both approaches aim to create flexible and responsive classification systems that maintain high accuracy over time, effectively handling dynamic data streams and emerging classes.

The subsequent sections of this paper adhere to a well-structured organization. Section 5.2 introduces the second proposed approach, providing intricate explanations of its constituents, including dynamic classifier ensembles, concept drift handling, and emergency class identification, and the adaptive proposed method of the emerging pool size. Section 5.4 outlines the experimental setup and presents the results, providing details regarding the employed datasets, evaluation metrics, and procedures. Finally, Section 5.5 presents a summary of this chapter.

5.1 Motivations and Contributions of this Chapter

This research proposes efficient algorithms for classifying data streams in real-time scenarios, focusing on addressing the issues caused by emerging new classes in data distributions. The goal is to develop novel algorithms that can effectively address the emergence of new classes in dynamic data streams, achieve high classification performance, and reduce computational complexity. The key contributions of this study are outlined as follows:

1. Our first contribution involves utilizing the ADWIN, DES, and K-means techniques in combination with the ensemble stratified bagging technique to detect and adapt to emerging new classes. This approach allows us to dynamically update the classification model to accommodate an evolving data environment.
2. The second contribution is the introduction of an adaptive method to adapt the emerging pool size depend on the stream distribution.

5.2 Proposed Methodology

This section outlines the key stages of the research, which are comprised of three distinct steps aimed at addressing the challenges of handling emerging new classes in dynamic data streams. Second proposed approach aims to develop a robust framework capable of tackling these complex challenges through a comprehensive, three-step methodology.

1. **Emerging new classes:** Our study addresses the widespread issue of emerging new classes in drifted streams using well-known techniques, including concept drift detection and K-means clustering.

2. **Drifted streams:** To address changes in the data distribution, the second approach integrates a concept drift detector that dynamically identifies shifts, allowing the model to promptly adapt its classifiers to maintain effectiveness.

3. **Classifier performance:** Classifier Performance: We utilize Dynamic Ensemble Selection (DES) to boost classifier performance. This method involves constructing a pool of classifiers and dynamically choosing the most appropriate one for each new data point, thereby enhancing both performance and robustness.

5.2.1 Second Proposed Approach Description

Second proposed approach is composed of three main phases that work together to manage emerging new classes and drifting data streams effectively:

1. **Dynamic ensemble selection:** The initial phase, DES, identifies the most suitable classifier for each incoming data chunk to ensure optimal performance.

2. **Drift detector phase:** In this phase, the system continuously observes the data stream in real time to detect concept drift, indicating changes in the underlying data distribution.

3. **Emerging new class identifier phase:** The final phase identifies unknown classes in drifted streams, creating new classifiers for emerging classes to improve the model's proficiency in accurately classifying new instances.

Figure 5.1 illustrates that DES extracts the current data chunk and utilizes the DES technique to identify the most effective classifiers for that chunk (black box of Fig. 5.1). These classifiers are used in the second phase to predict class labels, while drift detectors like ADWIN or DDM monitor for concept drift.

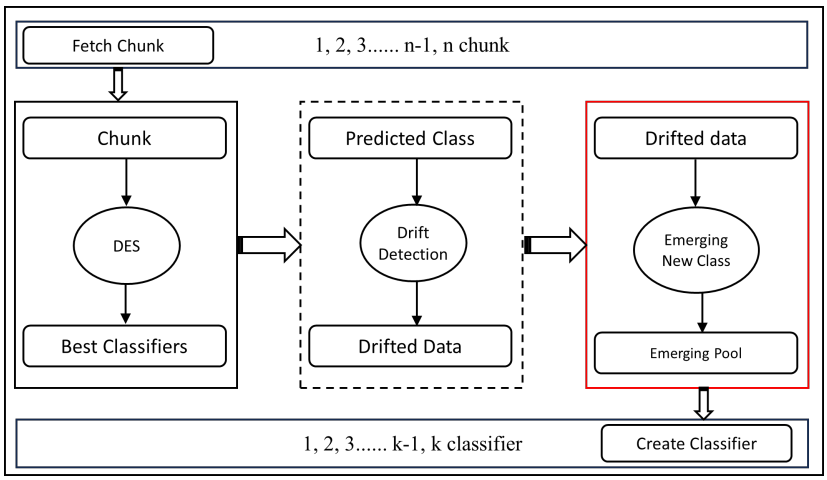


Figure 5.1: Flow Diagram of the Second Proposed Approach.

Algorithm 3: Second Proposed Approach Algorithm.

```
Input: data stream, classifier threshold κ
Data: current chunk a, classifier k, classifiers pool Ψ, drifted pool ψ
Output: prediction p
ψ ← ϕ
Ψ ← ϕ
for stream has chunk do
    if a is the first chunk then
        k ← trainingNewClassifier(a)
        P ← getPrediction(a, k)
    else
        k ← DES(a, Ψ)
        P ← getPrediction(a, k)
        ψ ← conceptDriftDetector(P)
        if ψ > 0 then
            k ←
              utilize a and ψ to get new classes according to Algorithm 4
            Ψ ← Ψ + k
            if |Ψ| > κ then
                removeWorstClassifier(a)
            end
            p ← getPrediction(a, k)
        end
    end
end
return p
```

If a drift is detected (highlighted by the red rectangle), the chunk is forwarded to the third phase, where new classes are identified (see Fig. 5.2). The overall approach is implemented in Algorithm 3, which takes a data stream and a DES Pool threshold as inputs. Key components include training the initial ensemble (Lines 4-6), using DES to select the best classifier for the current chunk (Line 8), detecting drifted chunks (Line 10), creating new classifiers for emerging classes (Line 12), and removing the worst classifier if the DES Pool threshold is exceeded (Lines 14-16).

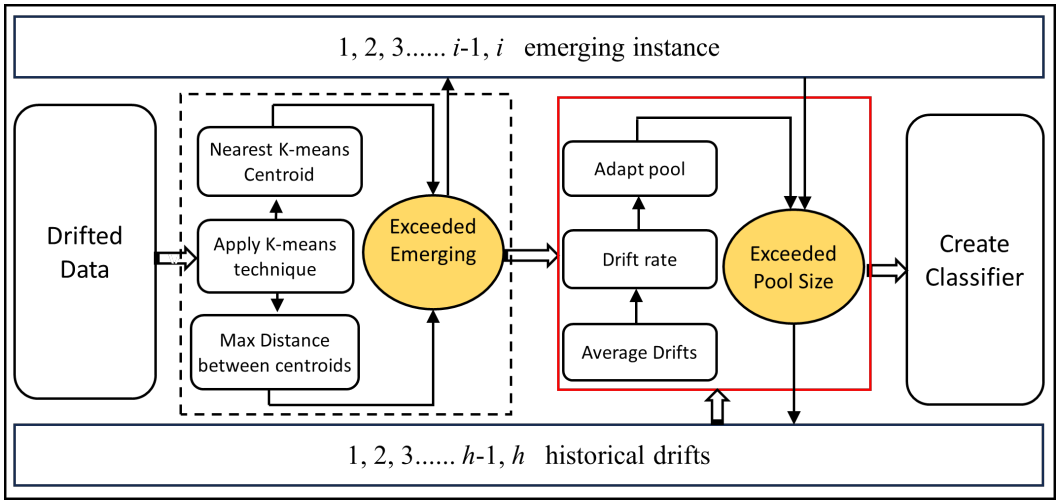


Figure 5.2: Flow Diagram of the Emerging Phase.

5.3 Mathematical Foundations for PA2

This section provides a mathematical analysis for the second proposed approach using the following equations:

The calculation of the maximum dissimilarity among class centroids is represented in Equation 5.1, where d_c identifies the class centroid with the maximum Euclidean Distance (ED) between any two centroids c_i and c_j:

$$d_c = \arg\max_{d} \sum_{i=1}^{i} \sum_{j=i+1}^{i} ED(c_i, c_j) \tag{5.1}$$

Equation 5.2 computes d_x, the minimum Euclidean Distance (ED) between a new data instance x and the existing class centroids c_i, identifying the closest class to x:

$$d_x = \arg\min_{i} \sum_{i=1}^{i} ED(x, c_i) \tag{5.2}$$

The decision-making process is governed by Equation 5.3, which determines the prediction p. If $d_x > d_c$, the new instance x is classified as an Emerging Class (EC); otherwise, the Dynamic Ensemble Selection (P_{DES}) process is used to predict its label:

$$p = \begin{cases} EC & \text{if } d_x > d_c \\ P_{DES} & \text{otherwise} \end{cases} \tag{5.3}$$

These equations collectively ensure that the HTL framework effectively identifies new emerging classes while maintaining accurate predictions for known classes through adaptive modeling and ensemble selection.

5.3.1 Emerging New Classes Phase Description

In this section,the Emerging new classes phase details is presented. Figure 5.2 shows that the emerging phase contains two primary steps:

1. **Emerging new classes identifier:** The emerging class detection step plays a crucial role in the second proposed approach (the black rectangle of Fig. 5.2) by utilize the K-means technique to clustering the drifted chunk into a set of clusters. And identifying new emerging classes by evaluating the distances between instances and their nearest class centroid. This process involves comparing the distance between an instance and the nearest class centroid with the maximum distance between class centroids (d_c), as described in Eq.5.1 and Eq.5.2, where ED represents the Euclidean Distance method and c_i, c_j denotes the centroids of the current classes. If the calculated distance (d_x) exceeds this threshold (d_c), this indicates the presence of a new emerging class, denoted by EC in Eq. 5.3; otherwise, it is P_DES (prediction class of the new instance).

2. **Adaptive the emerging classes pool size:** This critical step adapts the pool size based on the emergence rate of new classes and drift distribution, using statistical measures like standard deviation, first derivative, and average historical drift updates.

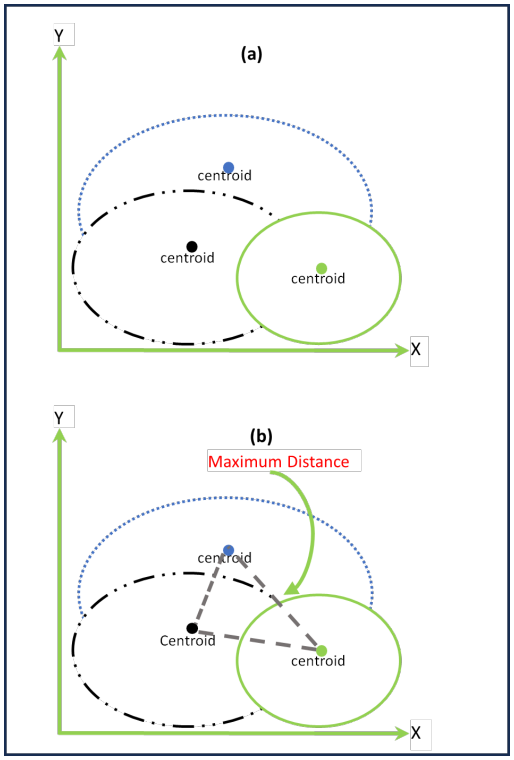


Figure 5.3: Steps for clustering a dataset with two features and three classes (Blue, Black, Green).

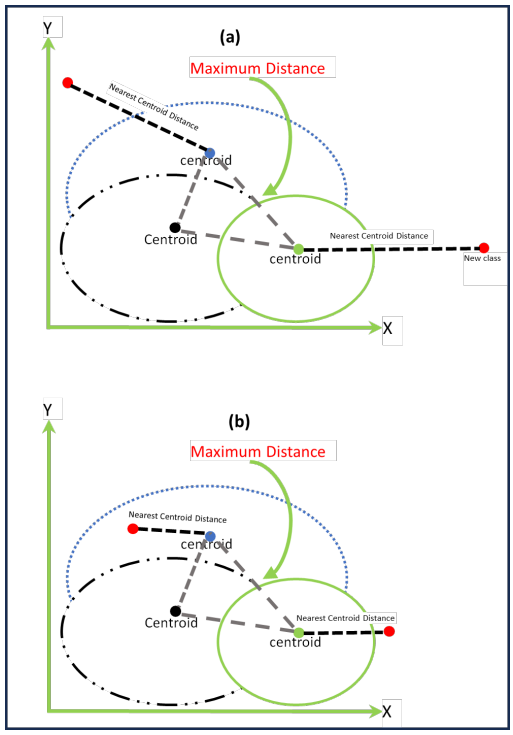


Figure 5.4: Steps for handling new points (red) with two features (x, y).

The Emerging New Classes phase is implemented in Algorithm 4, which utilizes drifted instances and current changes as inputs. Key steps include applying K-means (Line 1), calculating cosine similarity between centroids (Line 2), determining the maximum centroid distance (Line 3), setting the adaptive pool threshold (Lines 4-7), using KNN to find the nearest neighbor (Line 9), storing instances in the emerging pool if the distance exceeds the threshold (Lines 11-12), and creating a new classifier if the pool size surpasses the adaptive limit (Lines 14-17). A simulated scenario is illustrated in Figures 5.3 and 5.4. In Fig. 5.3 (a), the drifted chunk is divided into three clusters using the K-means algorithm. Figure 5.3 (b) shows the calculation of the maximum distance between cluster centroids, which serves as a threshold for identifying new classes. The procedure for classifying instances is as follows: when a drifted instance is detected, it is considered a new class if the distance between the instance and its nearest centroid exceeds the maximum inter-

centroid distance Fig. 5.4 (a). If the distance is within the threshold, the instance is classified as a known class Fig. 5.4 (b).

Algorithm 4: Flow Diagramof the Emerging Phase.

Input: current chunk a, drifted pool ψ, historical drifts h
Data: current chunk a, classifier k, Stream Emerging New Classes SENC, SENC Pool Ψ, K-means centroids λ, centroid distance Φ, maximum centroid distance dc, chunk distance i, Nearest neighbor n, historical drifts h, average historical drifts μ
Output: classifier k

$\lambda \leftarrow$ apply Kmeans(a)
$\Phi \leftarrow$ apply cosine similarity(λ)
$dc \leftarrow \max(\Phi)$
$\mu \leftarrow \frac{\sum h}{|h|}$
$\sigma \leftarrow$ std(ψ)
$\Delta \leftarrow$ first derivative(h)
threshold $\leftarrow \mu + \Delta \times \sigma$
for ψ *has instance* **do**
    $n \leftarrow$ apply KNN(λ, i)
    $d \leftarrow$ apply cosine similarity(i, n)
    if d $\geq$ dc **then**
        $\Psi \leftarrow \Psi + i$
    end
    if $|\Psi| \geq$ *threshold* **then**
        $k \leftarrow$ trainingNewClassifier(a)
        $h \leftarrow h + \psi$
        $\Psi \leftarrow \phi$
    end
end
return k

5.4 Experimental Results

The experiments conducted in this study were designed to assess the effect of concept drift on the performance of the second proposed approach in detecting emerging new classes. The primary goal was to identify the machine learning algorithm in conjunction with DES, Dynamic Ensemble Selection technique, which improves the performance of classification when faced with the emergence of new

classes. By conducting these experiments, valuable insights were gained, leading to potential enhancements in the effectiveness of the second proposed approach in managing imbalanced data streams and its overall performance. These experiments provide valuable insights into the capabilities of the framework and shed light on the optimal approach for handling emerging new classes in the presence of concept drift. The results provide valuable guidance for selecting the most suitable classification machine learning algorithm and DES configuration aims to enhance both classification performance and robustness. The experiments detailed in this study offer crucial insights into improving the second proposed approach's performance and tackling challenges related to emerging new classes and concept drifts in incremental drifted streams. These insights are instrumental in advancing stream mining techniques and developing more precise and resilient classification models for dynamic and evolving data-stream environments.

5.4.1 Experimental Setup

The evaluation of the second proposed approach involves a comparison with SENCForest [63], SENNE [1], and KENNES [50]. The evaluation utilizes several metrics, including recall, precision, F1 score [80], BAC [81], and G-mean [82]. The procedure for experimentation followed a test-then-train technique [83], where the classifier is first trained on a specific chunk and then evaluated on the subsequent chunk. Each chunk was standardized to 2000 instances. Four different classification models served as base estimators: Gaussian Naive Bayes (GNB) technique, K-Nearest Neighbors (KNN), the Support Vector Classifier (SVC), and the Hoeffding Tree (HT), all features are implemented using scikit-learn [84]. We establish an ensemble classifier pool with a set limit of L = 8, wherein each ensemble consists of N = 4 base models. While these constraints remained fixed across all experiments, the threshold for the pool classifier in each approach was maintained at eight. Consequently, if the threshold is surpassed, the least-performing classifier is systematically eliminated. ADWIN [54] and DDM [31] are employed as concept drift detectors as implemented in the River library [1] , to identify concept drifts, as they are considered more accurate techniques for use in incremental data streams

[1] `https://riverml.xyz/0.21.0/introduction/basic-concepts/`

[31, 51, 52, 54]. This configuration was applied consistently across all approaches to ensure fair engagement. The experiments were conducted using Python, and the source code is available publicly on GitHub [1].

5.4.2 Data Streams

In this section, the performance of the second Proposed Approach (PA2), was evaluated using several datasets, including synthetic data streams, a real-world application stream, and benchmark datasets. To conduct the evaluations, the stream-learn and River python libraries [84] were used. As detailed in Table 5.1, the Covertype dataset stream is used as the benchmark dataset in the study. This dataset comprises 52 features, 7 classes, and a total of 581,010 instances. It serves as a standard benchmark dataset widely used in stream mining research. For the evaluation of real application streams, the Sensor stream dataset was utilized. This dataset includes 5 features, 58 classes, and 392,600 instances in total, reflecting a real-world application scenario. Synthetic dataset stream was generated using the scikit-learn Python package [84]. This synthetic stream was generated to mimic data streams and assess the performance of the second proposed method. It included 10 features and four classes, divided into 200 chunks, each with a size of 2,000. The effectiveness of the second proposed method was rigorously assessed using these datasets along with a stream-learn library. This thorough evaluation offered important insights into how well the approach manages various types of data streams, encompassing benchmark datasets, synthetic data streams, and real-world application streams.

Table 5.1: Summary of Dataset Characteristics Utilized in the PA2 Experimental.

Data stream	Features	Classes	Instances	Chunk Size
Sensor stream [2]	5	58	392600	2000
Covertype stream[3]	52	7	581010	2000
Synthetic stream	10	4	200000	2000

[1] `https://github.com/Amadkour/transfer_learning_with_concept_drift.git`

5.4.3 Compared Approaches

In the following section, a comparative analysis between GNB methodology and three established benchmark techniques is presented, detailed as follows:

1. **SENCForest [63]:** The SENCForest method employs anomaly detection techniques to identify new classes and is founded on entirely random trees. It constructs isolation trees for both classification and detection by subsampling from each class, utilizing 20 random trees and a subsample size of 20. Although it can operate with incomplete or no label information, it frequently suffers from elevated false positive rates and suboptimal runtime efficiency in practical applications.

2. **SENNE [64]:** Utilizing a hypersphere ensemble mechanism, SENNE calculates scores to identify both new and known classes. It operates with a buffer capacity of 100 and sets a new class-score threshold of 0.5.

3. **KENNES [50]:** The KNNENS method addresses the dual challenge of detecting new classes and classifying known classes within a unified framework. It was configured with a buffer size of 100 and a new class score threshold of 0.5.

5.4.4 Examination of Experimental Findings

This section offers an in-depth analysis of the second approach's performance across various data streams. To ensure a thorough evaluation, five performance metrics— recall, F1 score, precision, recall, G-mean, and BAG—are displayed through two types of visual diagrams: radar and line charts. The radar chart effectively summarizes the performance of each algorithm by highlighting their performance across the six key metrics. Additionally, to evaluate the overall performance of each method—including PA2, SENCForest, SENNE, and KENNES—the average value for each metric was computed. Additionally, a line diagram using the G-mean metric was used to compare these methods for each chunk across 200 chunks. A radar diagram provided a detailed and nuanced evaluation of the second approach's performance across various experimental scenarios.

5.4.4.1 **Results On The Benchmark Dataset**

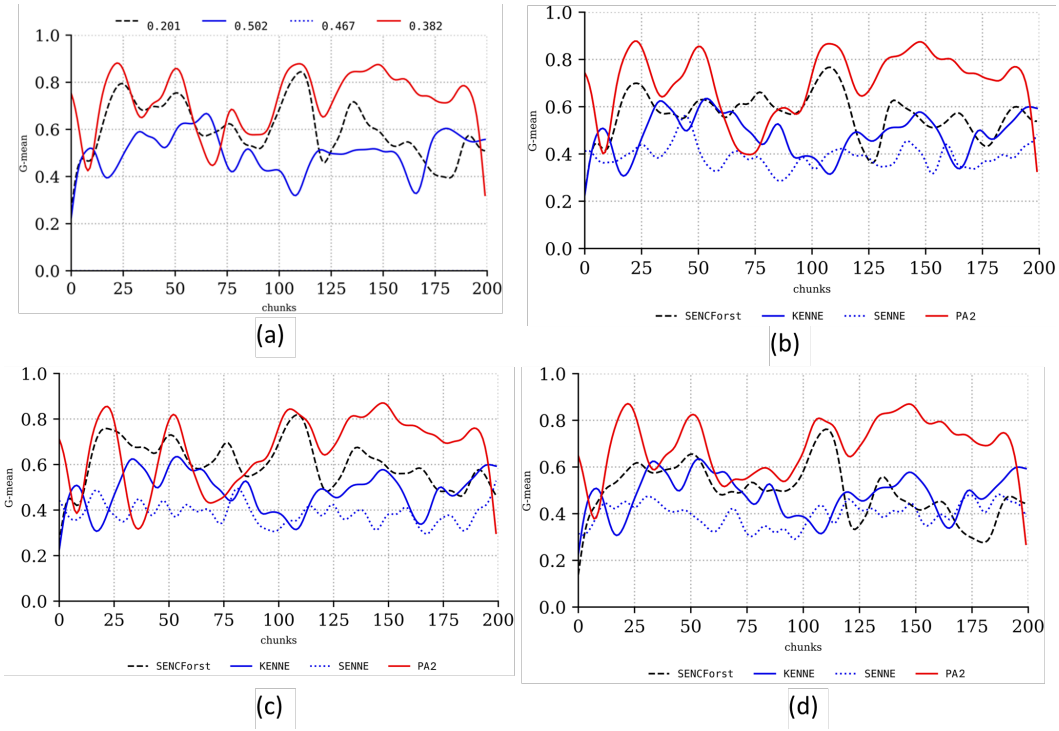


Figure 5.5: Covertype Stream Performance.

This experiment sought to evaluate the performance of the second approach on the Covertype data stream using different buffer sizes, with ADWIN employed as the concept drift detector. From a philosophical perspective, this experiment exemplifies the idea of adaptation to context and the gradual refinement of understanding. The experiment's findings are presented through scatter (line) plots in Figure 5.5, which depict the effects of emerging buffer sizes—5, 10, 20, and adaptive sizes—on performance. In the initial analysis (Fig. 5.5(b)), the G-mean of various methods—SENCForest, SENNE, KENNES, and PA2—across 200 chunks with a buffer size of 5 instances reveals that early performance is suboptimal. This initial period of underperformance mirrors philosophical concepts of learning through trial and error and the necessity of time for refinement. Similar to how human cognition or understanding improves after prolonged exposure to data or experience, the classifier pool grows in the experiment, which gradually enhances performance.

By chunk 20, the classifiers begin to adapt, as the dynamic ensemble selection (DES) technique can more effectively identify the most competent classifiers for future chunks. This process of selection reflects the philosophical notion of wisdom through discernment, where the system, like an informed agent, selects the best approach to navigate complexities. The performance improvements are particularly evident after chunk 20, with PA2 consistently outperforming other methods, while KENNES and SENNE lag behind, representing thediffering capacities of systems to adapt to and learn from new information.

The experiments with buffer sizes of 10 and 20 (Fig. 5.5(b) and Fig. 5.5(c)) show that while performance slightly decreases compared to the buffer size of 5, the larger buffer sizes may offer fewer opportunities for updates, illustrating the tension between stability and flexibility in knowledge systems. The adaptive buffer size experiment (Fig. 5.5(d)) further underscores the concept of continuous adaptation, where PA2's superior performance highlights the algorithm's ability to adjust dynamically to the data stream, providing the best outcomes across all chunks. The experiment reflects philosophical ideas of progressive refinement and adaptive responsiveness. Just as knowledge systems evolve by selecting the most relevant information at the right time, the classifier system's ability to choose the most effective classifiers through dynamic adaptation mirrors this ongoing process of learning and improvement.

5.4.4.2 **Results On Real Application Stream**

This experiment aimed to evaluate the performance of the algorithms SENCForest, SENNE, KENNES, and PA2 across 200 chunks of a sensor data stream, using varying buffer sizes similar to the previous experiment. The results, presented in Figure 5.6, are visualized through scatter plots. In Figure 5.6(a), the classification performance of each algorithm is shown with a buffer size of 5. The PA2 algorithm outperformed the others overall, except in chunks 43 to 48, where some instances were mistakenly identified as outliers instead of emerging classes. In contrast, SENCForest and SENNE exhibited the lowest performance. Further experiments with buffer sizes of 10 and 20 (Figures 5.6(b) and 5.6(c)) revealed a decline in performance compared to the buffer size of 5, likely due to fewer updates resulting from the larger buffer sizes. The adaptive buffer size experiment in Figure

5.6(d) reinforced the effectiveness of the PA2 algorithm, which consistently outperformed the other algorithms across all chunks, with the adaptive emerging pool size delivering the best results. From a philosophical perspective, this experiment highlights the importance of dynamic adaptability in response to changing data, as well as the trade-off between stability and flexibility. The PA2 algorithm's ability to adjust its performance through varying buffer sizes can be seen as an example of how systems of thought and knowledge can evolve by learning from emergent patterns and refining their strategies over time. The experiment underscores the balance between responsiveness and efficiency, suggesting that continuous adaptation can lead to more effective problem-solving and decision-making in dynamic environments.

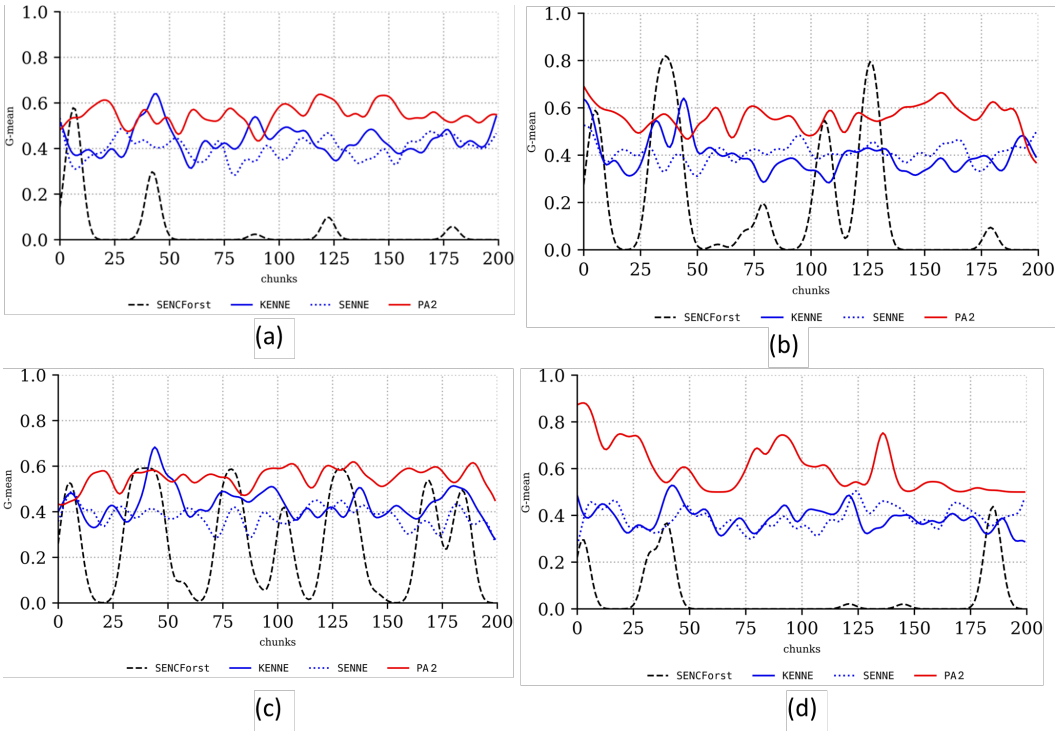


Figure 5.6: Sensor Stream Performance.

5.4.4.3 **Results On Synthetic Stream**

This experiment sought to evaluate the performance of previously tested methods on a synthetic data stream. The results, presented in Figure 5.7, illustrate that

the PA2 algorithm consistently achieved the highest performance across different buffer sizes of 5, 10, and 20. As shown in Figures 5.7(a), 5.7(b), and 5.7(c), scatter plots reveal PA2's superior performance in all these scenarios. Furthermore, Figure 5.7(d) highlights that the adaptive pool size yielded the best results on the synthetic stream, emphasizing the adaptive nature and efficiency of the PA2 algorithm. Philosophically, this experiment underscores the importance of adaptability in complex and dynamic environments. The PA2 algorithm's consistent superior performance reflects a system's ability to respond flexibly to emerging patterns and changing conditions, embodying the notion that the most effective strategies are often those that can continuously refine and adjust based on new information. This adaptability mirrors the process of philosophical growth, where knowledge systems must remain open to revision and improvement in response to evolving contexts, leading to more refined and robust outcomes over time.

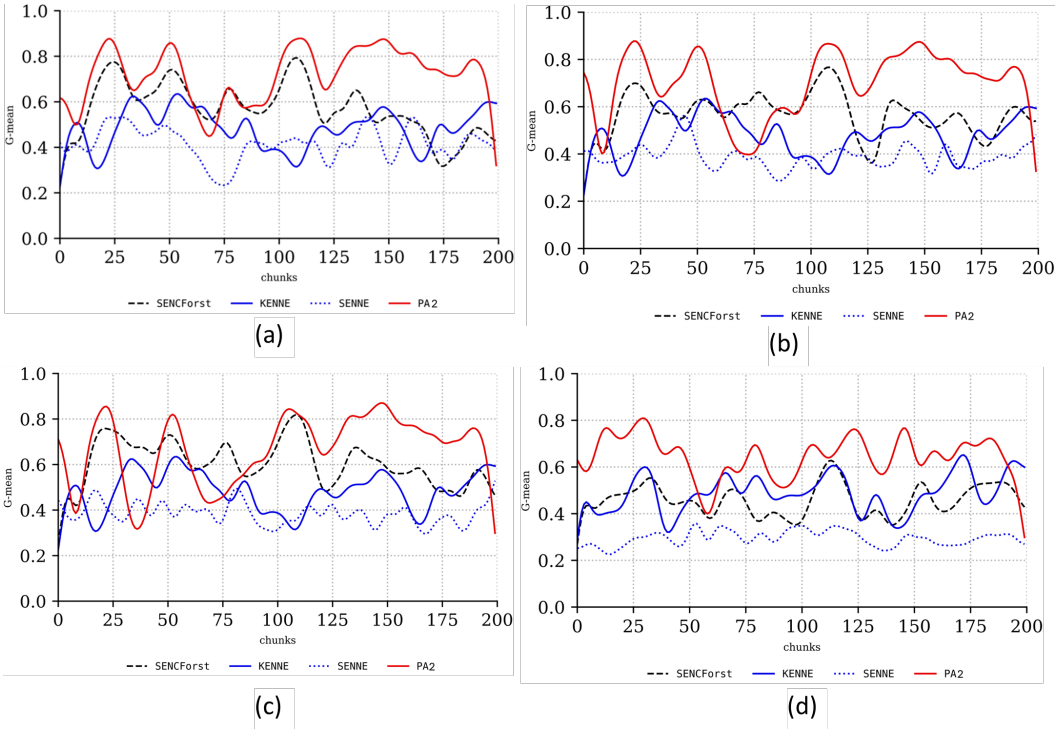


Figure 5.7: Synthetic Stream Performance.

5.4.5 Runtime Analysis Of The Best Algorithms

Our experimental results indicate that the selection of the optimal algorithm for detecting emerging new classes is influenced by several factors, including dataset characteristics, buffer length, and the algorithm's runtime requirements. A series of experiments were conducted to evaluate the runtime performance of different algorithms, with the PA2 algorithm demonstrating exceptional efficiency. Specifically, using the adaptive pool size approach, the PA2 algorithm trained bagging classifiers 150 times within 2131 seconds on the Covertype stream, outperforming others despite SENCForest achieving the lowest runtime but with fewer updates compared to PA2, as shown in Table 5.2. On the Sensor stream, PA2 delivered the best runtime for 66 updates, while SENCForest, KENNES, and SENNE required more time to complete the same or fewer iterations. Similarly, PA2 exhibited superior runtime performance on the Synthetic stream. These results underscore the PA2 algorithm's efficiency, making it an ideal choice for time-constrained scenarios. The superior runtime performance of PA2 is further highlighted in bold in Table 5.2. These findings reflect the notion of pragmatic efficiency—the ability to achieve desired outcomes with optimal resource utilization. In decision-making and knowledge systems, the most effective strategies are often those that can maximize results while minimizing costs, both in terms of time and computational resources. The PA2 algorithm's ability to consistently perform well under time constraints mirrors the importance of practical adaptability in real-world applications, where solutions must be both effective and efficient, capable of meeting demands without unnecessary complexity.

5.4.6 Comparison between GNB, KNN, and HT, SVC

In this section, a comparison of various algorithms—KNN, SVC, GNB, and HT—was conducted as base classifiers for the bagging technique. This comparison, shown in Figure 5.8, focuses on the Covertype dataset stream, where GNB (blue line) and HT (red line) consistently achieved the highest scores across most chunks. Both classifiers also performed well in radar plots across key performance metrics, such as balanced accuracy, precision, recall, G-mean, and F1 score. Based on these results, GNB and HT were identified as the most suitable base classifiers

Table 5.2: Runtimes (in seconds) Comparison of SENCForest, SENNE, KENNE, and PA2.

Stream	Algorithm	Training Times	Time (seconds)
Covertype	SENCForest	142	**1997**
	SENNE	5	2167
	KENNES	3	1742
	PA2	150	2131
Sensor	SENCForest	23	1244
	SENNE	17	3825
	KENNES	152	2109
	PA2	66	**1075**
Synthetic	SENCForest	12	107
	SENNE	26	224
	KENNES	149	202
	PA2	47	**91**

for the bagging technique. Further comparisons between GNB and HT in terms of runtime and update frequency, as presented in Table 5.3, revealed that GNB offers the best runtime performance. However, HT demonstrated superior overall performance, likely due to its higher number of updates compared to GNB, as reflected in Table 5.3 and Figure 5.8. This analysis touches on the concept of trade-offs in decision-making and the balancing of multiple objectives. In this case, the trade-off between runtime efficiency (GNB) and performance improvement through frequent updates (HT) mirrors the broader challenge of optimizing strategies in real-world scenarios. The findings suggest that different approaches are suitable for different goals, with GNB excelling in time-sensitive contexts, while HT performs better when frequent adjustments and continuous learning are prioritized. This reflects the broader philosophical notion that optimality is context-dependent, and the best solution often depends on the specific demands of the environment or problem at hand.

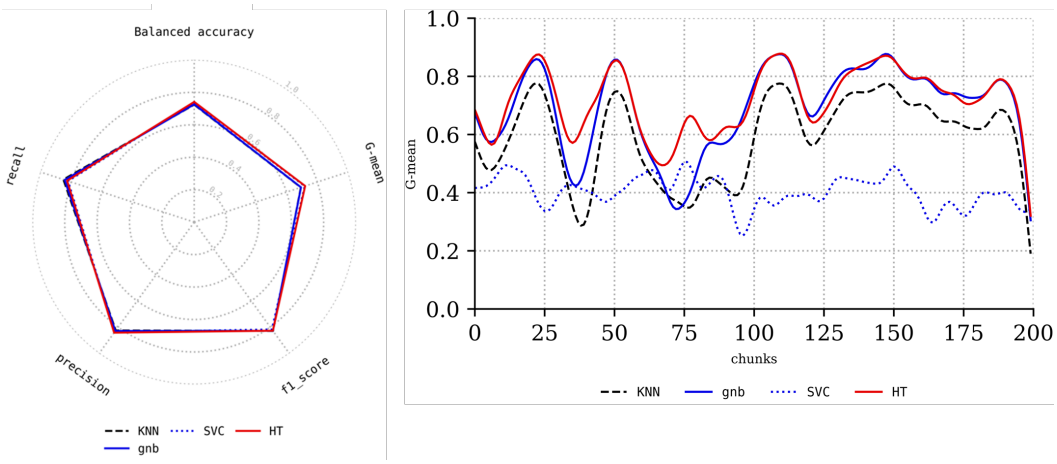


Figure 5.8: Performance Metrics for Covertype Stream with ADWIN and DDM.

Table 5.3: Runtimes (in seconds) of KNN, SVC, GNB, and HT.

Algorithm	**KNN**	**SVC**	**GNB**	**HT**
Training Times	140	150	**139**	148
Runtimes (in seconds)	586	878	**291**	1169

5.4.7 Comparison between ADWIN and DDM

In this section, two concept drift detection methods, the Drift Detection Method (DDM) [31] and the Adaptive Window (ADWIN) [31, 54], are compared. Both methods are highly regarded for their performance in managing incremental drifted streams [31, 51, 52, 54]. Figure 5.9 illustrates that when using the Synthetic stream with GNB as the base classifier, ADWIN consistently outperforms DDM in both radar and line plots across all performance metrics, including G-mean. Additionally, a comparison of their runtime and update frequency, shown in Table 5.4, reveals that ADWIN is the superior drift detector. It identifies the highest number of drifts (139) in the least amount of time (272 seconds), demonstrating its efficiency and effectiveness. This comparison highlights the concept of efficiency versus accuracy in decision-making. ADWIN's superior performance can be interpreted as a reflection of adaptive learning—the ability to respond swiftly and effectively to changes in data, a key aspect of systems designed for dynamic environments. This adaptability mirrors the philosophical idea of pragmatism, where the best solution

is not necessarily the most static or fixed but one that evolves in response to changing circumstances. In the context of concept drift, the ability to efficiently identify and react to changes is crucial, suggesting that dynamic systems that can self-adjust are often more effective than those that require fixed, predefined responses.

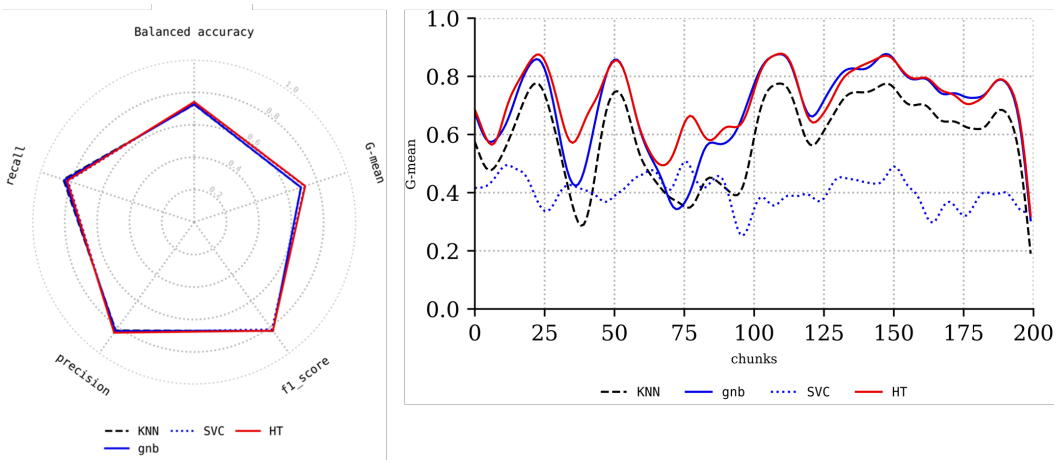


Figure 5.9: Performance Metrics for Synthetic Stream with ADWIN and DDM.

Table 5.4: Runtimes (in seconds) of ADWIN and DDM.

Algorithm	**ADWIN**	**DDM**
Training Times	139	55
Runtimes (in seconds)	272	545

5.5 Summary

This section presents a robust framework for incremental learning in data streams with emerging new classes. It integrates the ADWIN algorithm for concept drift detection, K-means clustering, and Gaussian Naive Bayes (GNB) within an ensemble stratified bagging approach. ADWIN enables rapid drift detection and adaptation to new classes, while GNB serves as the base classifier for enhanced performance. The ensemble stratified bagging method significantly improves predictive performance, adaptability, and scalability, ensuring real-time relevance by dynamically updating the model based on incoming data.

CHAPTER

6

Dynamic Classification Ensembles for Handling Imbalanced Multiclass Drifted Data Streams

Transfer learning is a powerful approach for addressing the challenges of dynamic data streams and concept drift. Its primary objective is to improve learning performance in a target domain by utilizing knowledge from one or more source domains. The source domain typically comprises data or tasks with abundant labeled information, whereas the target domain often faces constraints such as limited or unlabeled data. This distinction underscores the significance of transfer learning in bridging knowledge gaps and enhancing learning in resource-constrained environments [4, 85]. The field of transfer learning focuses on maximizing positive knowledge transfer while minimizing negative transfer. To achieve this, researchers have developed strategies such as instance re-weighting and feature matching to reduce the domain gap [70, 86–88]. Additionally, mitigating negative knowledge transfer often involves down-weighting irrelevant data sources to prevent the incorporation of misleading information [85]. While much of the research has been dedicated to static environments where data distributions remain stable, real-world scenarios like financial forecasting, energy demand prediction, and climate analysis often feature dynamic environments. These scenarios present the concept drift

problem, where evolving data distributions challenge traditional models [89, 90]. In dynamic environments, transfer learning must adapt to changing data. Dynamic Ensemble Selection (DES) is a key technique that dynamically evaluates and selects the most competent classifiers, improving performance while excluding underperforming ones to combat concept drift [15, 18, 56]. Adaptive Windowing (ADWIN) detects concept drift by monitoring statistical changes within sliding windows and reconfigures the ensemble by retraining classifiers or adding new ones suited to updated data distributions [52]. Additionally, the Streaming Ensemble Algorithm (SEA) adapts classifier ensembles to real-time data streams, maintaining robustness as data conditions and new classes emerge [31, 52, 54]. This study introduces efficient algorithms for classifying data streams in real-time, specifically addressing the challenges posed by heterogeneous transfer learning in non-stationary environments. The proposed framework integrates DES, ADWIN, and SEA to tackle dynamic data streams effectively. Its objectives include achieving high classification performance, adapting to evolving data distributions, and minimizing computational complexity. The subsequent sections of this paper adhere to a well-structured organization. Section 6.1 presents the motivations and contributions. Section 5.2 introduces the third proposed approach, providing intricate explanations of its constituents, including dynamic classifier ensembles, concept drift handling, and heterogeneous multisource transformation. Section 4.4 outlines the experimental setup and presents the results, providing details regarding the employed datasets, evaluation metrics, and procedures. Finally Section 6.5 presents a summary of this chapter.

6.1 Motivations and Contributions of this Chapter

This paper presents a novel contribution to real-time streaming scenarios by addressing the challenges of heterogeneous multisource streams, with key contributions that can be summarized as follows:

1. The primary innovation involves incorporating a concept drift detection method in conjunction with an ensemble classifier, enabling real-time adaptation and refinement of the third proposed approach in response to transfer learning in

non-stationary environments. This methodology ensures continuous evolution of the classification model in accordance with the changing data landscape.

2. The second significant advancement is the introduction of a precise weighting method to assess the significance of each local classifier within the ultimate classifier.
3. The third significant contribution is the development of an innovative approach that employs the eigenvector technique to facilitate the transfer of knowledge from heterogeneous source domains to the target domain.

6.2 Third Proposed Methodology

This section introduces the Heterogeneous Transfer Learning (HTL) algorithm tailored for non-stationary environments. HTL adeptly assimilates knowledge from both heterogeneous and homogeneous sources, operating within the framework of data streams subject to concept drift. Leveraging an online learning inductive parameter transfer strategy, HTL achieves seamless knowledge transfer. We delineate the core components and workflow of the HTL algorithm. To address the challenges intrinsic to the third approach, the focus is on three key aspects:

- **Heterogeneous sources:** Traditional transfer learning methods often assume homogeneity in the data distribution across source and target domains. However, in real-world scenarios, data sources may vary significantly in terms of feature space. HTL addresses this challenge by allowing knowledge transfer from sources with different dimensionalities. This means that the algorithm can effectively learn from diverse dimentionality sources, enabling a more comprehensive knowledge transfer process.
- **Drifted streams:** In dynamic environments, data distributions may change over time, leading to concept drift. This phenomenon poses a significant challenge for machine learning algorithms, as models trained on historical data may become obsolete as the underlying data distribution shifts. HTL incorporates a concept drift detector that continuously monitors the incoming data stream. Upon detecting a shift in the data distribution, the algorithm

adapts its classifiers accordingly to ensure continued effectiveness in handling drifting streams. This dynamic adjustment mechanism allows HTL to maintain high performance even in the presence of concept drift.

- **Classifier performance:** Classifier performance is crucial for the overall effectiveness of the transfer learning process. HTL employs Dynamic Ensemble Selection (DES) to enhance classifier performance. DES creates a diverse ensemble of classifiers, each trained on different subsets of the data. When presented with a new data point, DES dynamically selects the most suitable classifiers from the ensemble based on its performance on similar chunk. This adaptive selection process ensures that the most appropriate classifier is chosen for each data point, leading to improved classification performance and robustness. By leveraging DES, HTL maximizes the utility of available classifiers, resulting in superior performance in non-stationary environments with heterogeneous data sources.

6.2.1 HTL Description

The aim of this section is to harness knowledge from diverse dimensional multi-source domain streams. Following a similar framework to CDTL [1], this approach employs a class-wise and domain-weighted strategy. However, HTL enhances the weight function of CDTL in a class-wise manner, as depicted in Fig. 6.1, encompassing three phases:

- **Dynamic ensemble selection phase (DES Phase):**: The primary objective DES phase is to identify the optimal classifier for incoming data. This is crucial for ensuring that the selected classifier effectively aligns with the unique characteristics of the current data segment.
- **Drift detector phase:** After the DES phase, the method progresses to the drift detector phase, where it operates in real-time to continually monitor the data stream. This phase employs ADWIN and DDM techniques, which play a pivotal role in swiftly identifying any signs of concept drift. These techniques are designed to detect changes in the underlying data distribution over time, thus enabling the algorithm to adapt to evolving data patterns.

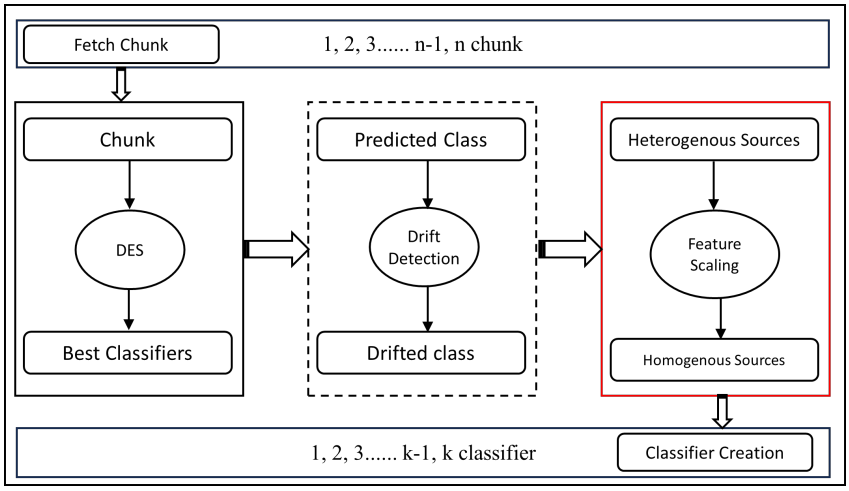


Figure 6.1: HTL Flow Diagram.

- **Feature scaling** The concluding phase of HTL is featuring scaling, operating in real-time to harmonize the diverse dimensionalities of the multisource streams with the target dimensionality. Leveraging the eigen vector technique, this phase facilitates the transformation of data into a unified dimensionality, essential for creating new classifiers tailored to the target and domain streams. By ensuring compatibility with the learning framework, this process enhances the algorithm's effectiveness in handling diverse dimensionality streams.

6.2.2 Classifier Creation Phase Description

Figure 6.2 provides a comprehensive overview of the classifier creation phase, a crucial component responsible for generating new classifiers based on the current chunk of the target domain, previous classifiers, and source domain classifiers. This phase offers several advantages and perform three tasks.

- **Source projection:** This step involves projecting the source domain classifiers onto the current chunk of the target domain. The projection likely employs the source weight function, as described in CDTL [1], to assign a weight to each classifier based on its relevance to the current chunk of data. This weighting mechanism ensures that classifiers from the source domains

contribute appropriately to the creation of the new classifier for the target domain.

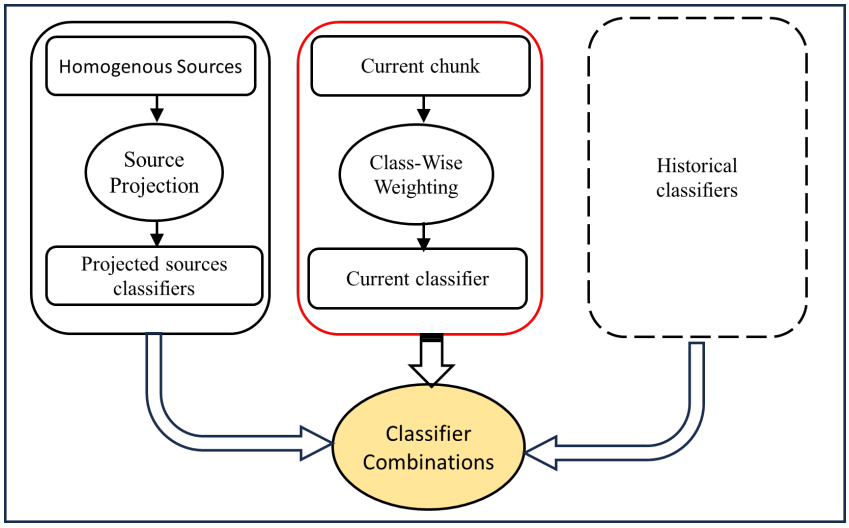


Figure 6.2: Classifier Creation Phase Flow Diagram.

- **Class-wise weights:** This block computes class-specific weights for each classifier using Equations 6.1 and 6.2 . The process involves analyzing the current chunk of the target stream (denoted as D) and considering both correct and incorrect predictions made by each classifier where $prediction_i$ refers to the predicted class of the current instance and c to the income class. These weights are crucial for determining the significance of individual classifiers across different classes. The equations facilitate the calculation of weights that reflect the classifiers' performance on specific classes, enabling the creation of a well-balanced ensemble classifier.
- **Classifier combination:** In this phase, the class-wise weights, along with the current chunk of data and the projected source domain classifiers, are combined to derive the final classifier. The combination process follows the details outlined in Eq. 6.3 , where historical classifiers (denoted as H), projected data classifiers (denoted as P), and the classifier of the current chunk (denoted as K) are integrated. This integration ensures that the resulting classifier incorporates contributions from both historical and projected data sources, leveraging the strengths of each to enhance predictive performance.

The resulting classifier is then utilized to make predictions based on the data chunk, effectively leveraging the insights gleaned from both historical and current data sources.

6.3 Mathematical Foundations of the HTL

The proposed Heterogeneous Transfer Learning (HTL) framework incorporates several mathematical components to enhance classification performance in dynamic data streams. Below, an analysis of the key equations is provided, used to define the methodology.

The first step in the framework is calculating the **class weight** for each class c predicted by a base classifier k over a dataset $\mathcal{D}$:

$$classWeight_{k,c,\mathcal{D}} = \sum_{i \in \mathcal{D}} \frac{|prediction_i = c|}{|\mathcal{D}|} \times \frac{|prediction_i \neq c|}{|\mathcal{D}|}. \tag{6.1}$$

This formula measures the relative contribution of a classifier k to the classification of class c, balancing correct and incorrect predictions within the dataset $\mathcal{D}$.

Next, the **classifier weight** is determined by aggregating the class weights across all classes C and classifiers K for the dataset:

$$classifierWeight = \sum_{k \in K} \sum_{c \in C} classWeight_{k,c,\mathcal{D}}, \quad \text{where } \mathcal{D} = 1, 2, 3, \ldots, N. \tag{6.2}$$

This aggregation quantifies the overall performance of a classifier k in the context of all classes in the dataset.

Finally, the **prediction ensemble** combines predictions from primary sources P, heterogeneous sources H, and the current classifier k for a given data chunk:

$$Prediction_{p,H,k,chunk} = \sum_{p \in P} prediction^{p}_{chunk} + \sum_{h \in H} prediction^{h}_{chunk} + prediction^{k}_{chunk}. \tag{6.3}$$

This equation ensures that the final prediction leverages diverse knowledge sources, including heterogeneous and primary streams, to achieve robust and accurate predictions for the data chunk.

These mathematical formulations underline the HTL framework's capability to dynamically adjust to evolving data distributions and heterogeneities in multisource streaming environments.

6.3.1 HTL Algorithm Description

As illustrated in Algorithm 5, the HTL algorithm involves several stages, beginning with training a classifier for the target stream and proceeding through various steps related to heterogeneous multisource preprocessing, classifier weighting, concept drift detection, and classifier management to ensure the performance and adaptability of the final classifier. In this section, detailed explanations of each individual step within the HTL algorithm are provided.

- **Converting heterogeneous multisource to homogeneous multisource:** In the initial step, the algorithm unifies the various data sources that might have different characteristics (heterogeneous multisource). This is achieved using eigenvectors and the feature count of the current chunk (line 7).
- **Training a new classifier for the first target chunk:** In line 8, the new classifier is trained specifically for the first chunk of the target stream. This classifier was used to predict the initial chunk of the target stream.
- **Calculate heterogeneous multisource weights:** The algorithm computes the weights for each heterogeneous data source. These weights are determined based on the characteristics of the data from each source and the target classifier. This weighting process helps prioritize more relevant data sources (line 9).
- **Calculating class-wise weights:** In line 27, the algorithm calculates class-wise weights using Equations 1 and 2. These weights were essential for determining the contribution of each class to the final classifier.
- **Converting homogeneous multisource to projected data source:** Line 11 involves using the weights calculated in the third step to transform the homo-

geneous multisource representation to a projected data source. This transformation likely uses the source weight function of CDTL [1].

- **Prediction of the current chunk:** In line 16, the algorithm determines the output class using Equation 3.
- **Monitoring for concept drift:** The algorithm continuously monitors the performance of its predictions to detect any concept drift, a situation in which the underlying data distribution changes, potentially leading to a degradation in model performance. Line 19 has been used for this purpose.
- **Updating the projected multisource classifiers:**Lines 29–32 are dedicated to updating the classifiers associated with the projected multisource. It is necessary to adapt to changes in the data or to maintain the performance and relevance of the classifiers over time.
- **Managing the pool of classifiers:** If the pool of classifiers exceeds a predefined maximum size threshold, lines 23 and 24 indicate that a new classifier is trained, and the worst-performing classifier is removed. This process helps to maintain a manageable and effective set of classifiers.

The heterogeneous Transfer Learning (HTL) algorithm represents a significant advancement in addressing the complexities of non-stationary environments prone to concept drift. By integrating heterogeneous and homogeneous sources within dynamic data streams, HTL demonstrates remarkable adaptability and effectiveness. Through its meticulously designed workflow, HTL can efficiently handle the challenges posed by evolving data landscapes. From the initial reception of environmental data to the nuanced processing of new data chunks and vigilant monitoring of concept drift, HTL embodies a comprehensive approach to knowledge transfer and adaptation. Owing to its ability to compute class-specific weights and leverage vital classifiers, HTL offers a robust solution for navigating non-stationary environments with confidence and precision.

6.4 Experimental Results

In this section, the proposed HTL, CORAL [70], and CDTL [1], and Melanie [2] algorithms are compared. Three subsections are introduced: the first section

focuses on the experimental setup; second, a discussion on data streams to apply all experiments on heterogeneous and homogeneous source domains, including a comparison to evaluate the runtime factor between all methods; and finally, a comparison is made between online learning and chunk-based concept drift.

6.4.1 Experimental setup

The evaluation of Heterogeneous Taxonomy Learning (HTL) involves a comparison with CORAL [70], CDTL [1], and Melanie [2] which employs multiple metrics such as precision, recall, F1 score [80], BAC [81], and G-mean [82]. The experimental protocol employed for evaluation followed the test-then-train approach [83], where the classification model is trained on a specific data chunk and subsequently evaluated on the next chunk. The chunk size was standardized to 2000 instances. Four different classification models were used as base estimators: k-Neighbors (KNN) algorithm, Support Vector Machine (abbreviated as SVM), Gaussian Naive Bayes abbreviated as (GNB), and Hoeffding Tree (abbreviated as HT), as implemented in scikit-learn [34]. We establish an ensemble classifier pool with a set limit of L = 8, wherein each ensemble consists of N = 4 base models. While these constraints remained fixed across all experiments, the threshold for the pool classifier in each approach was maintained at eight. Consequently, if the threshold is surpassed, the least-performing classifier is systematically eliminated. This configuration was applied consistently across all approaches to ensure fair engagement. The experiments were carried out using the Python programming language, with the source code publicly accessible on GitHub[1] . When dealing with heterogeneous multisource streams, it is often impractical or not applicable to truly heterogeneous experiences. Consequently, the eigenvector technique was utilized in all related approaches. This decision stems from the fact that the algorithms in related work primarily focus on homogeneous source domains and do not address the challenges posed by heterogeneous multisource data. Therefore, heterogeneous experiments were conducted.

[1] `https://github.com/Amadkour/transfer_learning_with_concept_drift.git`

6.4.1.1 Data Streams

In this study, the performance of the third proposed approach was evaluated using various datasets, including synthetic data streams, a real application stream, and dataset benchmark datasets. To conduct the evaluations, the stream-learn Python library [52, 84] was used. As detailed in Table 1, the benchmark dataset employed in the study is the Covertype dataset. This dataset comprises 52 features, 7 classes, and a total of 581,010 instances. It serves as a standard benchmark dataset widely used in stream mining research. For the evaluation of real application streams, the Sensor Stream dataset was utilized. This dataset includes 5 features, 58 classes, and a total of 392,600 instances, representing a real-world application scenario. Synthetic datasets were generated using the scikit-learn Python package. This synthetic dataset was created to simulate data streams and evaluate the performance of the framework. The synthetic datasets consisted of 10 features and four classes and were divided into 200 chunks. Each chunk had a size of 2,000. The performance of the third proposed approach was systematically evaluated using these datasets and a stream-learn library. These evaluations provided valuable insights into the effectiveness of the framework in handling different types of data streams, including benchmark datasets, real application streams, and synthetic data streams.

Table 6.1: Summary of Dataset Characteristics Utilized in the HTL.

Dataset	Number of Features	Number of Classes	Number of Instances
Covertype dataset[1]	40	7	581,010
Sensor Stream dataset[2]	5	58	392,600
Synthetic stream-52	52	5	200,000
Synthetic stream-8	8	4	200,000

6.4.1.2 Compared Approaches

In the subsequent section,the established benchmark techniques is introduced as outlined below:

- **AW- CORAL [70]:** The AW- CORAL (Adaptive Weighted CORrelation ALignment) method is a simple domain adaptation technique designed to align the distributions of both source and target features without supervision.

This is accomplished by aligning second-order statistics, focusing specifically on covariance, to match the distributions between the domains.

- **HE-CDTL [70]:** The HE-CDTL approach tackles Concept Drift Transfer Learning (CDTL) by integrating knowledge from source domains and historical time steps in the target domain to improve learning performance. HE-CDTL features class-wise weighted ensemble for independent selection of historical knowledge by each class, and AW-CORAL to mitigate domain discrepancy and reduce negative knowledge transfer.

- **Melanie [2]:** Multisource Online Transfer Learning for Non-stationary Environments, known as Melanie, represents the inaugural method capable of transferring knowledge across various data streaming sources in non-stationary environments. Melanie constructs numerous sub-classifiers to grasp diverse facets from distinct source and target concepts dynamically. It identifies sub-classifiers that align closely with the prevailing target concept, assembling them into an ensemble for predicting instances originating from the target concept.

While Melanie extends online learning through chunk-based learning akin to CDTL, it exhibits two limitations:

- utilizing a global weight for each classifier, disregarding performance variation across different locations of a data chunk

- combining learned classifiers from these domains directly with the target classifier, which may impede effective knowledge transfer due to domain discrepancies. Hence, the third proposed approach (HTL) is compared with AW- CORAL [70] and HE-CDTL [1].

6.4.2 Analysis of Experimental Results

This section presents a comprehensive evaluation of the performance of the proposed approach across multiple data streams. To guarantee a thorough assessment, five performance metrics—F1 score, precision, recall, G-mean, and BAG—were presented using two visualization diagrams: radar and line. A radar chart was

cleverly utilized to provide a comprehensive summary, precisely demonstrating the performance of each algorithm across the six key metrics. By calculating the average value of each metric, the overall performance of each approach (HTL, CORAL, and CDTL) was determined. On the other hand, the line diagram utilizes the G-mean metric to compare these methods for each chunk across 100 chunks. A line diagram was used in the first five experiments, whereas a combination of radar and line diagrams was employed in the last two experiments. This approach ensured a detailed and nuanced evaluation of the performance of the third proposed approach in diverse experimental scenarios.

6.4.2.1 **Results on Target Domains Dataset without Source Domain**

Figure 6.3 showcases the results of applying the aforementioned methods to the Covertype dataset, where no stream is utilized as a source domain. The line diagram specifically highlights the classification performance measured by the G-mean metric across 100 data chunks for each method. Notably, all methods display suboptimal performance during the initial 10 chunks. However, a significant improvement is evident in subsequent phases. This improvement can be attributed to the expansion of the classifier pool, encompassing an increasing number of classifiers. This expansion allows the DES technique to become more adept at selecting the most appropriate classifier for each incoming chunk. Consequently, performance experiences a notable upsurge in later chunks, underscoring the adaptability and efficacy of the ensemble approach. From chunk 20 to the final chunk, HTL consistently achieves the highest performance across most chunks. In contrast, CORAL shows lower performance in certain chunks, while CDTL displays lower performance in others. The superior performance of HTL can be credited to its utilization of the proposed weighting method, which assigns a weight to each historical classifier. This approach contributes to the effective training of the pool classifiers, thereby enhancing the overall performance of the approach.

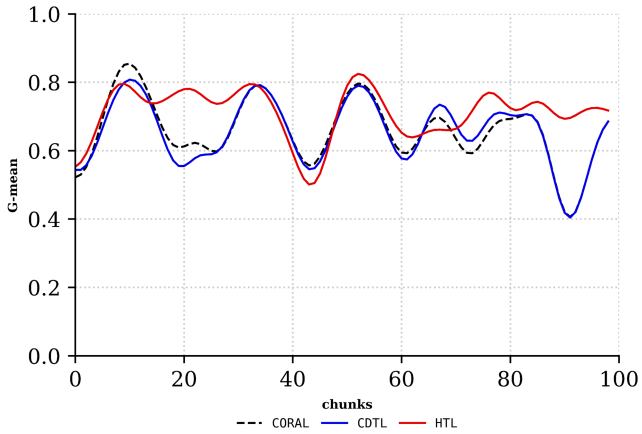


Figure 6.3: Results of the Covertype Stream as the Target Domain without the Source Domain.

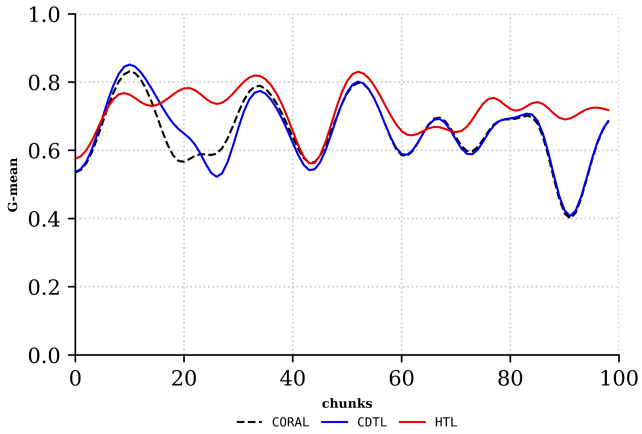


Figure 6.4: Performance results of the Covertype Stream as the target domain using homogeneous source domains.

6.4.2.2 **Results on the Homogeneous Source Domains Dataset**

Figure 6.4 depicts the results of applying the compared methods to the Covertype data stream as the target domain and Synthetic-52 as a homogenous source domain, which has the same dimensionality as the Covertype stream (52 features and seven classes). Upon examining the line diagram, it becomes evident that HTL's performance in the first eight chunks may be suboptimal due to the insufficient number of classifiers available. However, following the initial eight chunks,

the addition of more classifiers results in an improvement in HTL's performance. In contrast, CDTL maintains satisfactory performance throughout the chunks, while MLSMOTE consistently displays lower performance. It is important to note that HTL consistently achieves the highest performance across all chunks in the diagram. Notably, the CDTL and HTL methods in Fig. 6.4 demonstrated superiority over the same methods in Fig. 6.3. This is attributed to the experiment shown in Fig. 6.4, which incorporates Synthetic-52 as a source domain, thereby enhancing the performance of the methods. In contrast to the experiment in Fig. 6.3, which lacks a source domain.

6.4.2.3 **Results on One Heterogeneous Source Domain**

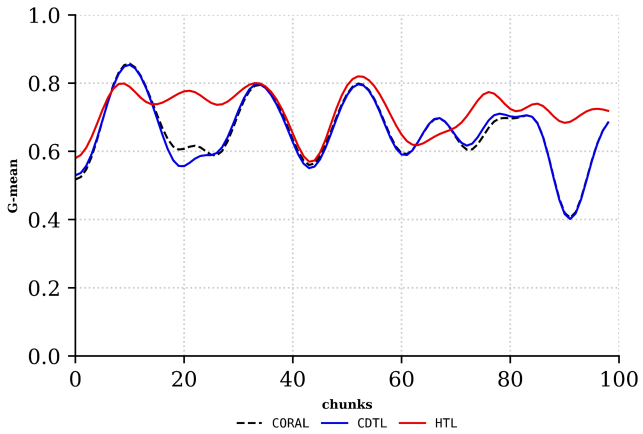


Figure 6.5: Performance results of the Covertype Stream as the target domain with a single heterogeneous source domain.

In this experiment, the Synthetic-8 stream was utilized as a heterogeneous source domain. Considering the nature of the HTL algorithm, which can operate with heterogeneous sources, specific adjustments were implemented to enable CORAL and CDTL, which typically do not operate with such sources, to function in a heterogeneous domain environment. The eigenvector technique was applied to CORAL and CDTL to facilitate compatibility with the heterogeneous sources. Figure 6.5 shows the results of applying the compared methods to the Covertype data stream as the target domain, with Synthetic-8 acting as the heterogeneous source domain. Synthetic-8 has a different dimensionality (eight features and four

classes) compared with the Covertype stream. An analysis of the line diagram reveals that the performance of the compared approaches may vary across different chunks depending on the source domain and the positive knowledge it contributes. HTL consistently outperforms, achieving the highest performance across almost all chunks, whereas CDTL and CORAL show lower performance.

6.4.2.4 **Results on All Heterogeneous and Covertype as the Target Domain**

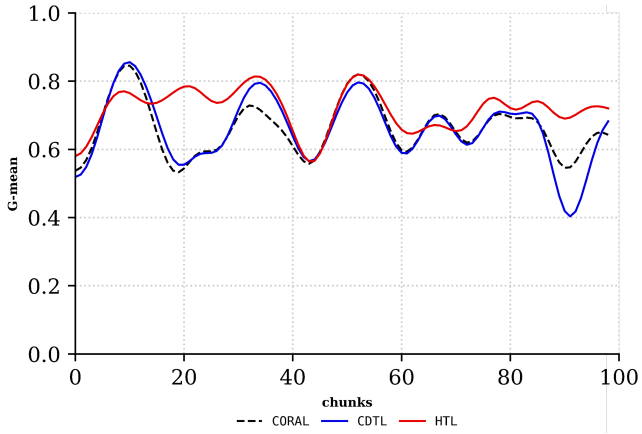


Figure 6.6: Performance results of the Covertype Stream as the target domain with multiple heterogeneous source domains.

Figure 6.6 presents the results of applying the compared methods to the Covertype data stream as the target domain, with the Synthetic-8, Synthetic-52, and Sensor streams serving as heterogeneous source domains. From the analysis, the performance in this figure exceeds that in Fig. 6.6. This improvement is likely due to the larger number of source domains in this experiment, which leads to an incremental increase in positive knowledge with each added source domain, consequently improving method performance. In Fig. 6.7, the same experimental setup was employed, but the target domain was switched from the cover type to the sensor stream. Additionally, the Covertype stream was included as one of the source domains in this configuration. This modification was intended to evaluate the performance of the HTL across the various target streams. Figure 6.7 shows that the performance of all methods might be lower than that of the Covertype stream, mainly because of the increased noise in the sensor stream and numerous

drifts. However, HTL consistently outperforms, achieving the highest performance across all chunks, whereas CORAL and CDTL display nearly identical values in most chunks.

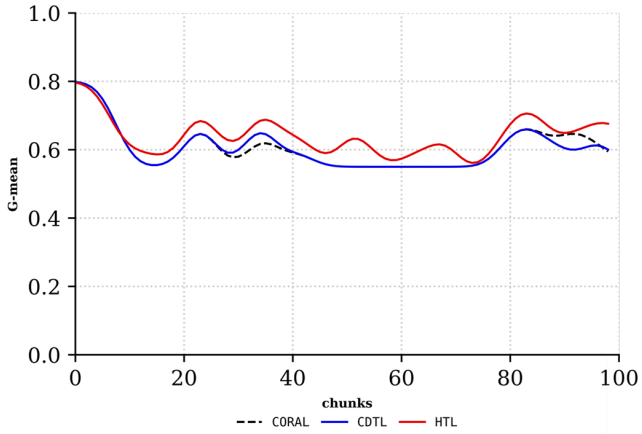


Figure 6.7: Results of the Sensor Stream as the Target Domain with Multiple Heterogeneous Source Domains.

6.4.2.5 **Results on One Heterogeneous Source Domain**

This section presents the overall performances of the compared methods (CORAL, CDTL, and HTL) across five key performance metrics: F1 score, precision, recall, G-mean, and BAG. Table 6.2 showcases the performance of each method across different experiments for the five metrics. The table illustrates the average value of each metric, which represents the overall performance of each method for 100 chunks. The emphasis on the bold highlighting in the table underscores the superiority of the HTL algorithm in various source domain configurations. In the first experiment (without the source domain), CORAL showed the best performance in terms of precision, while HTL demonstrated the best performance in the other metrics. Furthermore, in most experiments, HTL outperformed the other methods across most metrics, except for the recall metric in the third experiment, in which CDTL achieved the best performance. Notably, each subsequent experiment demonstrated improved performance compared with the previous experiment. This trend arose from the incremental nature of the source domain in each experiment. The incremental addition of the source domain enhances the positive knowledge of

each experiment, leading to an increased performance of the compared methods in most experiments.

Table 6.2: Comprehensive performance comparison of CORAL, CDTL, and HTL across all previous experiments.

Experiment	Metric	CORAL	CDTL	HTL
Without source domain	BAC	0.723	0.723	**0.725**
	G-mean	0.652	0.654	**0.717**
	F1 score	0.875	0.874	**0.890**
	Precision	0.827	0.823	**0.851**
	Recall	**0.950**	0.944	0.939
Homogenous source domain	BAC	0.731	0.730	**0.755**
	G-mean	0.658	0.653	**0.717**
	F1 score	0.876	0.875	**0.858**
	Precision	0.830	0.829	**0.851**
	Recall	0.950	0.950	**0.953**
Single heterogeneous source domain	BAC	0.732	0.732	**0.757**
	G-mean	0.661	0.657	**0.717**
	F1 score	0.875	0.875	**0.859**
	Precision	0.835	0.835	**0.851**
	Recall	0.947	0.950	**0.953**
Multisource heterogeneous domain (Covertype stream)	BAC	0.732	0.732	**0.757**
	G-mean	0.661	0.661	**0.717**
	F1 score	0.875	0.875	**0.859**
	Precision	0.835	0.835	**0.851**
	Recall	0.947	0.950	**0.953**
Multisource heterogeneous domain (Sensor stream)	BAC	0.998	0.998	**0.998**
	G-mean	0.971	0.971	**0.992**
	F1 score	0.997	0.997	**0.997**
	Precision	0.998	0.998	**0.998**
	Recall	0.998	0.998	**0.998**

6.4.3 Analysis Runtime between CORAL, CDTL, and HTL Techniques

Our experimental results highlight that the selection of the optimal algorithm for handling heterogeneous transfer learning is influenced by various factors. These factors include the dataset characteristics and runtime requirements of the algo-

rithm. In the experiments, the focus was on analyzing the runtimes of different algorithms, and the results were insightful. Notably, the HTL algorithm was exceptionally efficient. For example, when considering the training of the first experience (homogeneous source domain), the HTL algorithm completed 100 iterations within 304 seconds. In contrast, the CORAL and CDTL algorithms require more time to achieve the same number of training iterations. Furthermore, the HTL algorithm consistently demonstrated remarkable efficiency, particularly when the source domain settings were absent, single heterogeneous, or multisource heterogeneous. These findings highlight the superior runtime performance of the HTL algorithm, positioning it as an attractive choice for scenarios with stringent time constraints. The bold highlighting in Table 6.3 further accentuates the efficiency of the HTL algorithm across various source domain settings, including homogeneous, absent, single heterogeneous, and multi-source heterogeneous settings, making it a compelling option for applications where time efficiency is of paramount importance.

Table 6.3: Runtimes (in seconds) for CORAL, CDTL, and HTL.

Experience	Target domain	Source Domain	CORAL	CDTL	HTL
without source domain	Covertype	None	312	312	**304**
Homogenous source domain	Covertype	Synthetic-52	6165	614	**606**
Single heterogeneous source domain	Covertype	Synthetic-8	6060	4703	**4475**
Multi-source heterogeneous domain	Covertype	Sensor, Synthetic-52, and Synthetic-8	23011	14043	**13824**
Multi-source heterogeneous domain	Sensor	Covertype, Synthetic-52, and Synthetic-8	14524	3052	**3005**

6.4.4 Analysis performance and runtime of HTL for online learning and chunk-based concept drift

In this experiment, the performance durations of various learning flow techniques is investigated, specifically online learning and chunk-based concept drift, aiming to provide a thorough comparison between them. As shown in Table 6.4, the EDDM drift detector demonstrated significant time savings by successfully detecting 40 drifted chunks out of 100 chunks, resulting in 40 learning times. Conversely, the ADWIN drift detector achieved time savings superior to the online technique, having detected 63 drifted chunks and requiring 63 learning times. On the other hand, online learning exhibited the least efficient time savings, as it necessitated 100 times for 100 chunks. Moreover, in figures 6.8,6.9, each figure includes two

diagrams (radar and line diagram). As seen in the radar and line diagrams of Fig. 6.8, used to compare online learning and the ADWIN detector, online learning emerges as the most effective technique for achieving optimal performance, albeit with an increase in time consumption.

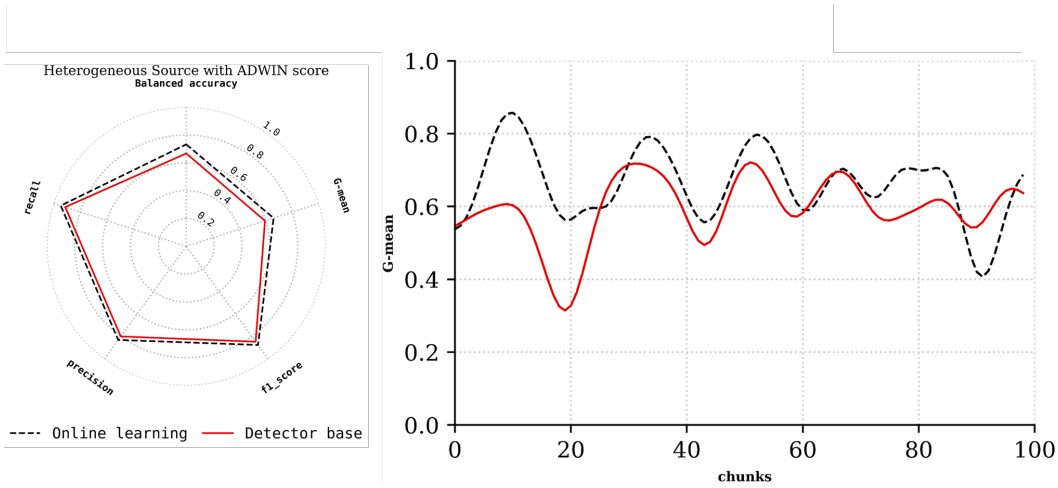


Figure 6.8: Online learning and chunk-based results for the Covertype stream using the ADWIN detector.

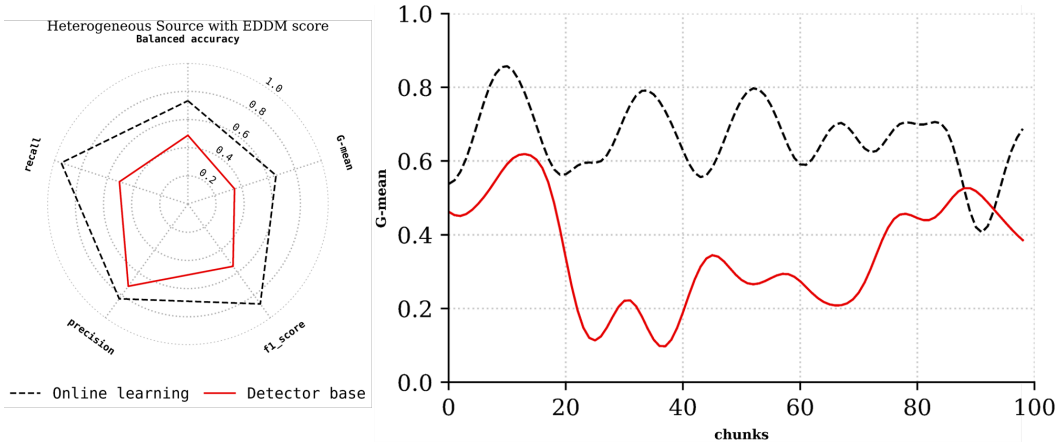


Figure 6.9: Results of online learning and chunk-based detection in the Covertype stream using the ADWIN detector.

Similarly, Fig. 6.9 is employed to compare online learning with the EDDM detector, demonstrating that online learning is the most effective method for opti-

mal performance. Consequently, online learning stands out as the most effective approach. Hence, by examining the radar diagrams of both figures, it becomes evident that ADWIN is the most similar alternative to online learning. As a result, ADWIN proves to be a favorable compromise, striking a balance between online learning and EDDM detectors in terms of performance. In contrast, EDDM exhibits the lowest performance. Therefore, in the context of chunk-based concept drift, particularly when utilizing the ADWIN detector, this approach is ideal for environments that demand strong performance while minimizing time expenditure.

Table 6.4: Runtimes (in seconds) for the Online Learning, ADWIN, and EDMM.

Technique	**Runtime (in seconds)**	**Learning Times**
Online Learning	3102	100
ADWIN detector	2257	63
EDMM detector	1948	40

6.5 Summary

This study presents a novel framework for incremental learning in data streams, with a focus on heterogeneous transfer learning. The approach effectively integrates concept drift detection via the ADWIN algorithm, eigenvectors for knowledge transfer, and ensemble classifiers. Extensive experiments on benchmark, real-world, and synthetic datasets validate the framework's efficacy in adapting to evolving data patterns. The use of Dynamic Ensemble Selection (DES) optimizes classifier performance, enhancing predictive performance and robustness. The framework excels in handling heterogeneous multi-source domains, providing high adaptability, efficiency, and scalability to real-time data streams while maintaining reliable and relevant performance.

Algorithm 5: Flow Diagram of the Heterogeneous Transfer Learning.

Input: Target domain stream *stream*, heterogeneous multisource domain Ψ, pool of classifiers, threshold ℓ

Data: Current chunk *a*, classifiers *k*, source classifiers ψ, projected source domain *S*, target domain weights ω, source domain weights λ

Output: Prediction

```
for stream have chunk do
    if a is the First chunk then
        Ψ ← convertSourcesToTargetDim(Ψ, a)
        k ← trainingNewClassifier(a)
        λ ← SourcesDomainWeights(Ψ, k)
        ω ← classWiseWeights(K, a)
        S ← projectedSourceDomain(Ψ, λ)
        for source in S do
            newClassifier ← trainingNewClassifier(source)
            ψ ← ψ ∪ newClassifier
        prediction ← getPrediction(a, k, ψ, λ, ω)
    else
        prediction ← getPrediction(a, k, ψ)
        driftResult ← conceptDriftDetector(prediction)
        if driftResult have drift then
            newClassifier ← trainingNewClassifier(a)
            k ← k ∪ newClassifier
            if size(K) ≥ ℓ then
                removeWorstClasssifier(k)
            λ ← SourcesDomainWeights(Ψ, k)
            ω ← classWiseWeights(K, a)
            S ← projectedSourceDomain(Ψ, λ)
            for source in S do
                newClassifier ← trainingNewClassifier(source)
                ψ ← ψ ∪ newClassifier
            prediction ← getPrediction(a, k, ψ, λ, ω)
return prediction
```

CHAPTER

7

Conclusions and Future Work

This chapter summarizes the key findings of this study on incremental learning in data streams and presents potential directions for future research to further improve and expand the methodology.

7.1 Conclusions

- This study introduces a novel approach integrating oversampling techniques, concept drift detection, and Dynamic Ensemble Selection (DES) to identify the most suitable ensemble classifiers.
- We demonstrated the effectiveness of the methodology through extensive experiments across various datasets, including benchmarks, real-world streams, and synthetic data.
- A key feature is the rapid detection of concept drift, enabling the system to adapt quickly, ensuring continued relevance and performance in real-time scenarios.
- We addressed the minority class problem using advanced oversampling and KNN algorithms to prevent overlapping class instances.
- The DES technique was pivotal in selecting the most effective classifiers, optimizing overall performance.

- Our methodology excels in handling multiclass imbalanced stream challenges, maintaining high performance as class distributions evolve.
- The approach is marked by its adaptability, efficiency, and scalability, offering dynamic model updates to ensure real-time reliability.
- Despite its advantages, the methodology has limitations such as the significant time required to generate non-overlapping synthetic instances, and its reliance on the performance of MLSMOTE and MLSOL techniques.

7.2 Future Work

- Future research should focus on developing more advanced oversampling techniques that avoid overlapping synthetic instances, thus reducing the computational burden.
- Exploring meta-learning methods to better assess imbalanced multiclass ratios and improve minority class identification could enhance the effectiveness of the approach.
- Further development of advanced concept drift detection algorithms is needed to improve the timely recognition of drift and better handle the evolving data distributions.
- The incorporation of alternative ensemble strategies and deep learning techniques could lead to further improvements in both drift detection and classifier performance.
- Expanding the methodology to address a broader range of real-world applications, including streaming from diverse domains, will enhance its applicability and robustness.
- Future work should also focus on refining classifier selection and prediction methods, especially for handling multisource domains in real-time environments.
- Investigating the use of deep learning methods to predict future drifts and optimize classifier performance is a promising direction for improving model adaptation and performance.

Bibliography

[1] C. Yang, Y.-m. Cheung, J. Ding, and K. C. Tan, "Concept drift-tolerant transfer learning in dynamic environments," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 8, pp. 3857–3871, 2021.

[2] B. Dong, Y. Gao, S. Chandra, and L. Khan, "Multistream classification with relative density ratio estimation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3478–3485.

[3] J. Shan, H. Zhang, W. Liu, and Q. Liu, "Online active learning ensemble framework for drifted data streams," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 2, pp. 486–498, 2018.

[4] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[5] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[6] S. Wang, L. L. Minku, and X. Yao, "A systematic study of online class imbalance learning with concept drift," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 10, pp. 4802–4821, 2018.

[7] Y. Sun, A. K. Wong, and M. S. Kamel, "Classification of imbalanced data: A review," *International journal of pattern recognition and artificial intelligence*, vol. 23, no. 04, pp. 687–719, 2009.

[8] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Addressing imbalance in multilabel classification: Measures and random resampling algorithms," *Neurocomputing*, vol. 163, pp. 3–16, 2015.

[9] ——, "Mlsmote: Approaching imbalanced multilabel learning through synthetic instance generation," *Knowledge-Based Systems*, vol. 89, pp. 385–397, 2015.

[10] Z. Daniels and D. Metaxas, "Addressing imbalance in multi-label classification using structured hellinger forests," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[11] B. Liu and G. Tsoumakas, "Making classifier chains resilient to class imbalance," in *Asian Conference on Machine Learning*. PMLR, 2018, pp. 280–295.

[12] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Evolving diverse ensembles using genetic programming for classification with unbalanced data," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 3, pp. 368–386, 2012.

[13] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 463–484, 2011.

[14] R. M. Cruz, R. Sabourin, and G. D. Cavalcanti, "Dynamic classifier selection: Recent advances and perspectives," *Information Fusion*, vol. 41, pp. 195–216, 2018.

[15] L. I. Kuncheva, "Clustering-and-selection model for classifier combination," in *KES'2000. Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies. Proceedings (Cat. No. 00TH8516)*, vol. 1. IEEE, 2000, pp. 185–188.

[16] T. Woloszynski and M. Kurzynski, "A probabilistic model of classifier competence for dynamic ensemble selection," *Pattern Recognition*, vol. 44, no. 10-11, pp. 2656–2668, 2011.

[17] R. Lysiak, M. Kurzynski, and T. Woloszynski, "Optimal selection of ensemble classifiers using measures of competence and diversity of base classifiers," *Neurocomputing*, vol. 126, pp. 29–35, 2014.

[18] R. M. Cruz, R. Sabourin, and G. D. Cavalcanti, "Meta-des. oracle: Meta-learning and feature selection for dynamic ensemble selection," *Information fusion*, vol. 38, pp. 84–103, 2017.

[19] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "Smoteboost: Improving prediction of the minority class in boosting," in *Knowledge Discovery in Databases: PKDD 2003: 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Cavtat-Dubrovnik, Croatia, September 22-26, 2003. Proceedings 7*. Springer, 2003, pp. 107–119.

[20] S. Wang, H. Chen, and X. Yao, "Negative correlation learning for classification ensembles," in *The 2010 international joint conference on neural networks (IJCNN)*. IEEE, 2010, pp. 1–8.

[21] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine learning*, vol. 23, pp. 69–101, 1996.

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

[23] N. Lu, J. Lu, G. Zhang, and R. L. De Mantaras, "A concept drift-tolerant case-base editing technique," *Artificial Intelligence*, vol. 230, pp. 108–133, 2016.

[24] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[25] V. Losing, B. Hammer, and H. Wersing, "Knn classifier with self adjusting memory for heterogeneous concept drift," in *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 2016, pp. 291–300.

[26] A. Storkey, "When training and test sets are different: characterizing learning transfer," 2008.

[27] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, and F. Herrera, "A survey on data preprocessing for data stream mining: Current status and future directions," *Neurocomputing*, vol. 239, pp. 39–57, 2017.

[28] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. d. Carvalho, and J. Gama, "Data stream clustering: A survey," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, pp. 1–31, 2013.

[29] A. Dries and U. Rückert, "Adaptive concept drift detection," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 2, no. 5-6, pp. 311–327, 2009.

[30] C. Alippi and M. Roveri, "Just-in-time adaptive classifiers—part i: Detecting nonstationary changes," *IEEE Transactions on Neural Networks*, vol. 19, no. 7, pp. 1145–1153, 2008.

[31] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17*. Springer, 2004, pp. 286–295.

[32] L. Bu, C. Alippi, and D. Zhao, "A pdf-free change detection test based on density difference estimation," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 2, pp. 324–334, 2016.

[33] T. D. S. K. S. Venkatasubramanian and K. Yi, "An information-theoretic approach to detecting changes in multi-dimensional data streams."

[34] I. Frias-Blanco, J. del Campo-Ávila, G. Ramos-Jimenez, R. Morales-Bueno, A. Ortiz-Diaz, and Y. Caballero-Mota, "Online and non-parametric drift detection methods based on hoeffding's bounds," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 810–823, 2014.

[35] M. Yamada, A. Kimura, F. Naya, and H. Sawada, "Change-point detection with feature selection in high-dimensional time-series data," in *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[36] S. H. Bach and M. A. Maloof, "Paired learners for concept drift," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 23–32.

[37] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 2007, pp. 443–448.

[38] Y. Sun, K. Tang, Z. Zhu, and X. Yao, "Concept drift adaptation by exploiting historical knowledge," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 10, pp. 4822–4832, 2018.

[39] N. C. Oza and S. Russell, "Experimental comparisons of online and batch versions of bagging and boosting," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 359–364.

[40] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavalda, "New ensemble methods for evolving data streams," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 139–148.

[41] F. Chu and C. Zaniolo, "Fast and light boosting for adaptive mining of data streams," in *Advances in Knowledge Discovery and Data Mining: 8th Pacific-Asia Conference, PAKDD 2004, Sydney, Australia, May 26-28, 2004. Proceedings 8*. Springer, 2004, pp. 282–292.

[42] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, vol. 106, pp. 1469–1495, 2017.

[43] M. Pratama, J. Lu, and G. Zhang, "Evolving type-2 fuzzy classifier," *IEEE Transactions on Fuzzy Systems*, vol. 24, no. 3, pp. 574–589, 2015.

[44] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 71–80.

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

[46] H. Yang and S. Fong, "Incrementally optimized decision tree for noisy big data," in *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, 2012, pp. 36–44.

[47] ——, "Countering the concept-drift problems in big data by an incrementally optimized stream mining model," *Journal of Systems and Software*, vol. 102, pp. 158–166, 2015.

[48] L. Rutkowski, L. Pietruczuk, P. Duda, and M. Jaworski, "Decision trees for mining data streams based on the mcdiarmid's bound," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1272–1279, 2012.

[49] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, "A new method for data stream mining based on the misclassification error," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 5, pp. 1048–1059, 2014.

[50] J. Zhang, T. Wang, W. W. Ng, and W. Pedrycz, "Knnens: A k-nearest neighbor ensemble-based method for incremental learning under data stream with emerging new classes," *IEEE transactions on neural networks and learning systems*, vol. 34, no. 11, pp. 9520–9527, 2022.

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

[52] A. H. Madkour, A. Elsayed, and H. Abdel-Kader, "Historical isolated forest for detecting and adaptation concept drifts in nonstationary data streaming," *IJCI. International Journal of Computers and Information*, vol. 10, no. 2, pp. 16–27, 2023.

[53] C. H. Tan, V. C. Lee, and M. Salehi, "Information resources estimation for accurate distribution-based concept drift detection," *Information Processing & Management*, vol. 59, no. 3, p. 102911, 2022.

[54] J. N. Adams, S. J. van Zelst, T. Rose, and W. M. van der Aalst, "Explainable concept drift in process mining," *Information Systems*, vol. 114, p. 102177, 2023.

[55] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[56] K. Jackowski, B. Krawczyk, and M. Woźniak, "Improved adaptive splitting and selection: the hybrid training method of a classifier based on a feature space partitioning," *International journal of neural systems*, vol. 24, no. 03, p. 1430007, 2014.

[57] T. Yin, C. Liu, F. Ding, Z. Feng, B. Yuan, and N. Zhang, "Graph-based stock correlation and prediction for high-frequency trading systems," *Pattern Recognition*, vol. 122, p. 108209, 2022.

[58] J. Ren, Y. Wang, Y.-m. Cheung, X.-Z. Gao, and X. Guo, "Grouping-based oversampling in kernel space for imbalanced data classification," *Pattern Recognition*, vol. 133, p. 108992, 2023.

[59] V. C. Nitesh, "Smote: synthetic minority over-sampling technique," *J Artif Intell Res*, vol. 16, no. 1, p. 321, 2002.

[60] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *International conference on intelligent computing*. Springer, 2005, pp. 878–887.

[61] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem," in *Advances in knowledge discovery and data mining: 13th Pacific-Asia conference, PAKDD 2009 Bangkok, Thailand, April 27-30, 2009 proceedings 13*. Springer, 2009, pp. 475–482.

[62] T. Maciejewski and J. Stefanowski, "Local neighbourhood extension of smote for mining imbalanced data," in *2011 IEEE symposium on computational intelligence and data mining (CIDM)*. IEEE, 2011, pp. 104–111.

[63] X. Mu, K. M. Ting, and Z.-H. Zhou, "Classification under streaming emerging new classes: A solution using completely-random trees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 8, pp. 1605–1618, 2017.

[64] Y.-N. Zhu and Y.-F. Li, "Semi-supervised streaming learning with emerging new labels," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 7015–7022.

[65] X.-Q. Cai, P. Zhao, K.-M. Ting, X. Mu, and Y. Jiang, "Nearest neighbor ensembles: An effective method for difficult problems in streaming classification with emerging new classes," in *2019 IEEE international conference on data mining (ICDM)*. IEEE, 2019, pp. 970–975.

[66] M. Long, J. Wang, G. Ding, J. Sun, and P. S. Yu, "Transfer feature learning with joint distribution adaptation," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 2200–2207.

[67] ——, "Transfer joint matching for unsupervised domain adaptation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1410–1417.

[68] Q. Sun, R. Chattopadhyay, S. Panchanathan, and J. Ye, "A two-stage weighting framework for multi-source domain adaptation," *Advances in neural information processing systems*, vol. 24, 2011.

[69] Y. Freund, R. E. Schapire *et al.*, "Experiments with a new boosting algorithm," in *icml*, vol. 96. Citeseer, 1996, pp. 148–156.

[70] B. Sun, J. Feng, and K. Saenko, "Return of frustratingly easy domain adaptation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[71] M. M. Rahman, C. Fookes, M. Baktashmotlagh, and S. Sridharan, "Correlation-aware adversarial domain adaptation and generalization," *Pattern Recognition*, vol. 100, p. 107124, 2020.

[72] E. Zhong, W. Fan, J. Peng, K. Zhang, J. Ren, D. Turaga, and O. Verscheure, "Cross domain distribution adaptation via kernel mapping," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 1027–1036.

[73] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," *arXiv preprint arXiv:2010.16061*, 2020.

[74] B. Liu, K. Blekas, and G. Tsoumakas, "Multi-label sampling based on local label imbalance," *Pattern Recognition*, vol. 122, p. 108294, 2022.

[75] V. López, A. Fernández, J. G. Moreno-Torres, and F. Herrera, "Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. open problems on intrinsic data characteristics," *Expert Systems with Applications*, vol. 39, no. 7, pp. 6585–6608, 2012.

[76] M.-L. Zhang, Y.-K. Li, H. Yang, and X.-Y. Liu, "Towards class-imbalance aware multi-label learning," *IEEE Transactions on Cybernetics*, vol. 52, no. 6, pp. 4459–4471, 2020.

[77] A. Liu, Y. Song, G. Zhang, and J. Lu, "Regional concept drift detection and density synchronized drift adaptation," in *IJCAI International Joint Conference on Artificial Intelligence*, 2017.

[78] Q. Da, Y. Yu, and Z.-H. Zhou, "Learning with augmented class by exploiting unlabeled data," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 28, no. 1, 2014.

[79] X. Mu, F. Zhu, J. Du, E.-P. Lim, and Z.-H. Zhou, "Streaming classification with emerging new class by class matrix sketching," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[80] Y. Sasaki *et al.*, "The truth of the f-measure. teach tutor mater, 1 (5), 1–5," 2007.

[81] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *2010 20th international conference on pattern recognition*. IEEE, 2010, pp. 3121–3124.

[82] M. Kubat, S. Matwin *et al.*, "Addressing the curse of imbalanced training sets: one-sided selection," in *Icml*, vol. 97, no. 1. Citeseer, 1997, p. 179.

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

[84] P. Ksieniewicz and P. Zyblewski, "Stream-learn—open-source python library for difficult data stream batch analysis," *Neurocomputing*, vol. 478, pp. 11–21, 2022.

[85] Z. Wang, Z. Dai, B. Póczos, and J. Carbonell, "Characterizing and avoiding negative transfer," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 293–11 302.

[86] B. Zadrozny, "Learning and evaluating classifiers under sample selection bias," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 114.

[87] C. Cortes, M. Mohri, M. Riley, and A. Rostamizadeh, "Sample selection bias correction theory," in *International conference on algorithmic learning theory*. Springer, 2008, pp. 38–53.

[88] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE transactions on neural networks*, vol. 22, no. 2, pp. 199–210, 2010.

[89] P. Li, X. Wu, X. Hu, and H. Wang, "Learning concept-drifting data streams with random ensemble decision trees," *Neurocomputing*, vol. 166, pp. 68–83, 2015.

[90] Z. Cao, K. You, M. Long, J. Wang, and Q. Yang, "Learning to transfer examples for partial domain adaptation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2985–2994.