1

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 [**?** **?**]. 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 [**?**]. To address this issue, concept drift detectors are used to identify changes in data stream distributions by leveraging informa-

tion 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 [**?** **?**], which is characterized by uneven class distribution, poses a challenge for traditional classifiers [**?**], particularly in multiclass scenarios where minority class samples are at risk of misclassification owing to their limited representation [**?**]. Addressing imbalanced data classification requires specialized techniques to ensure accurate minority class classification without compromising the performance of the majority class [**?** **?** **?**]. 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 [**?**]. 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) [**?** **?**]. The second and third groups encompass adaptive algorithms and hybrid methods, respectively. Adaptive algorithms include one-class and cost-sensitive classification [**?**]. Hybrid methods merge data preprocessing with classification techniques, often utilizing ensemble classifiers to effectively mitigate class imbalance and enhance classifier performance [**?** **?** **?**]. Class overlap occurs when instances from different classes inhabit the same region in the data space [**?** **?**]. 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 [**?**]. 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 [**?**]. 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 [**?**]. 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 [**? ? ?**].

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 1.1,the motivations and the contributions are presented. The first proposed framework are

discussed in detail in Section 1.2. The experimental results and the discussion are presented in Sections 1.4. Section 1.5 presents a summary of this chapter.

1.1 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.

1.2 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 [**?** **?**], 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.

1.2.1 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.

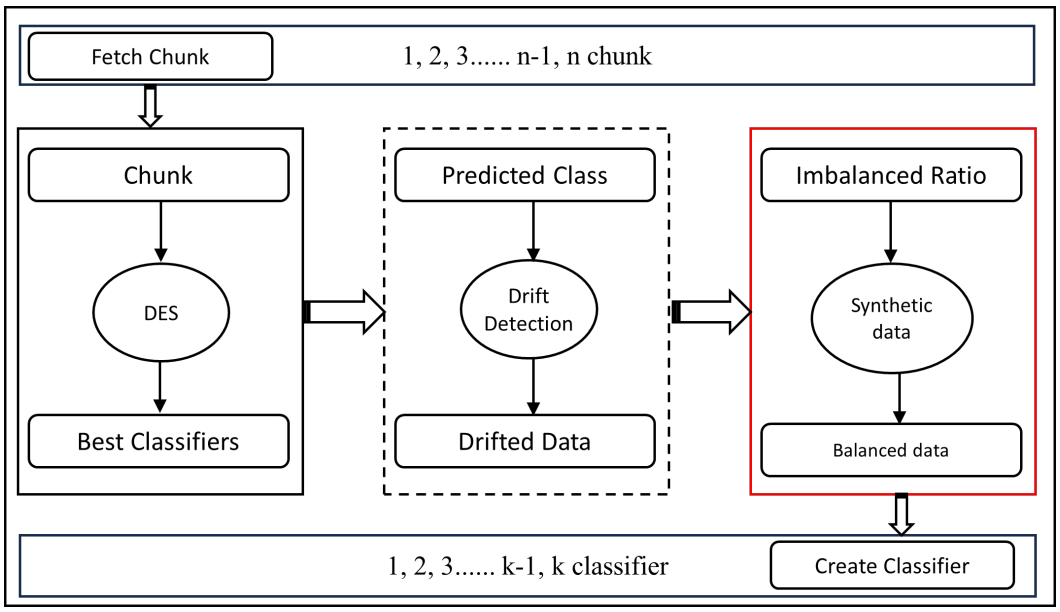

Figure 1.1: First Proposed Approach (PA1) for Imbalanced Multi-class Drifted Streams.

Specifically, as shown in Fig. 1.1, 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 [**?**], 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. 1.1. In the third phase, first proposed method uses a set of equations

1.11.21.3 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 1.2 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.

1.2.2 Synthetic Data Generator Phase Description

Figure 1.2 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.

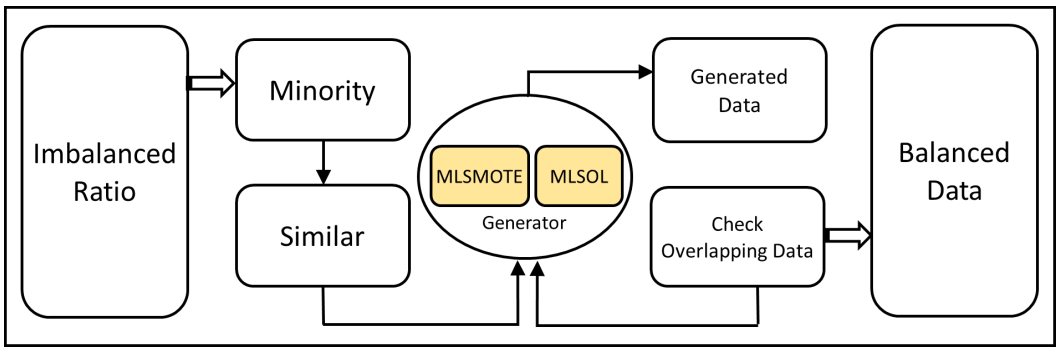

Figure 1.2: Flow Diagram of the Synthetic Data Generator.

- **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 [**?**], 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 [**?** **?**]. 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 [**?**] and MLSOL [**?**]. 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.

1.3 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 1.1 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:

$$frq_c = \sum_{i=1}^{\text{chunk size}} \begin{cases} 1, & \text{if } y_i = c \\ 0, & \text{otherwise} \end{cases}, \quad i = 1, 2, 3, \ldots \text{chunk size}. \tag{1.1}$$

Equation 1.2 defines the *best frequency* for a chunk, $freq_n$, as the proportion of instances n relative to the total number of classes C:

$$\text{best } freq_n = \frac{|n|}{|C|}. \tag{1.2}$$

Finally, Equation 1.3 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 ($freq_{\text{chunk}}$):

$$\text{classes type}_{\text{chunk}} = \sum_{c=1}^{C} \begin{cases} \text{Minority}, & \text{if } diff(sd_c - frq_i) > \text{best } freq_{\text{chunk}} \\ \text{Majority}, & \text{otherwise.} \end{cases} \tag{1.3}$$

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

1.4 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.

1.4.1 Experimental setup

The evaluation of the first proposed approach incorporated the utilization of various metrics such as recall, precision, specificity, F_1 score, balanced accuracy score (BAC), and geometric mean score (G-mean) [**?**]. The experimental protocol utilized for evaluation was the test-then-train approach [**?**], 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 [**?**]. 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

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

to demonstrate the effectiveness of the contribution. Additionally, these experiments are conducted that uses two different concept drift detectors, ADWIN [**?**] and DDM [**?**], to demonstrate the adaptability and robustness of first proposed approach across varying drift detectors.

1.4.2 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 [**?**]. Table 1.1 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 the effectiveness of the first approach in handling different types of data streams, including benchmark datasets, real application streams, and synthetic data streams.

Table 1.1: 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

1.4.3 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—F_1 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.

1.4.3.1 Results on the Benchmark Stream

Figures 1.3 and 1.4 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.

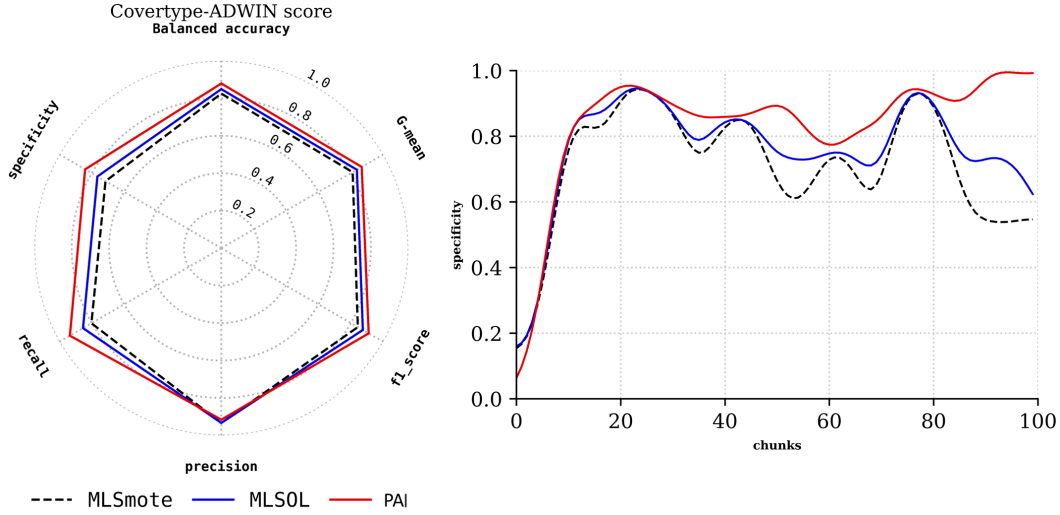

Figure 1.3: Comparison of PA1, MLSMOTE, and MLSOL on Covertype Dataset with ADWIN Concept Drift Detector.

In Figure 1.3, 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, F_1 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 1.4, 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 1.4 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.

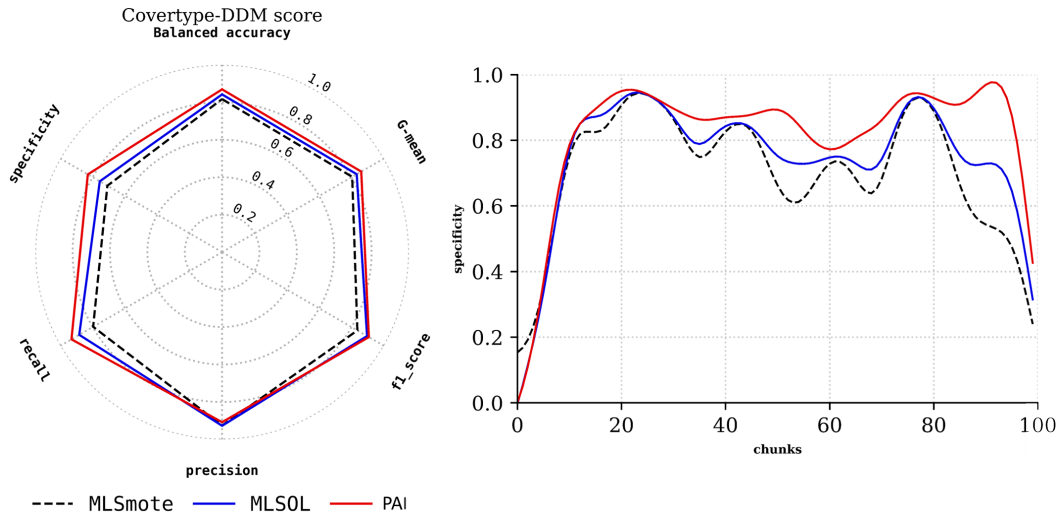

Figure 1.4: Comparison of PA1, MLSMOTE, and MLSOL on Covertype Dataset with DDM Concept Drift Detector.

1.4.3.2 Results on the Real Application Stream

Figures 1.5 and 1.6 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 1.5, 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, F_1 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 1.5 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 1.6 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 1.5 and 1.6 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.

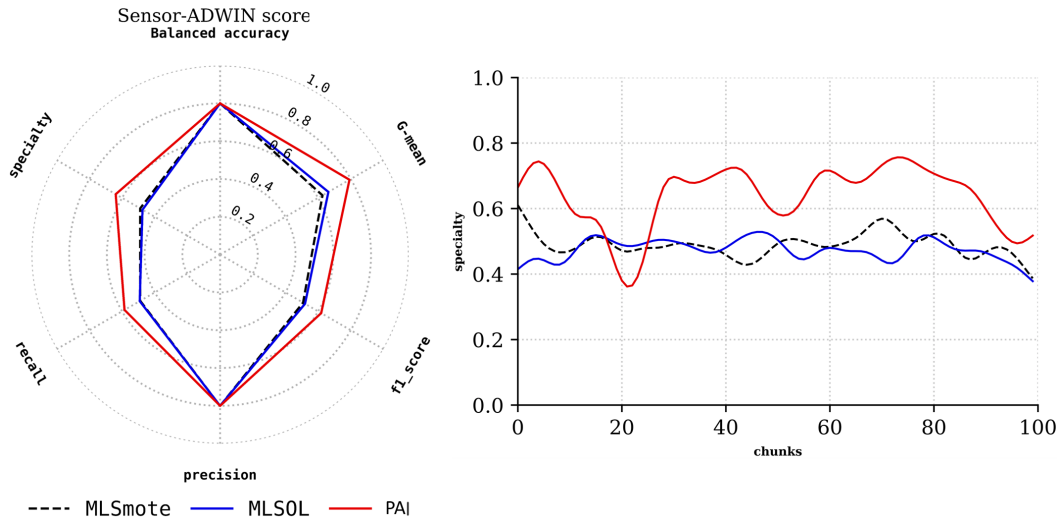

Figure 1.5: Comparison of PA1, MLSMOTE, and MLSOL on Sensor Dataset with ADWIN Concept Drift Detector.

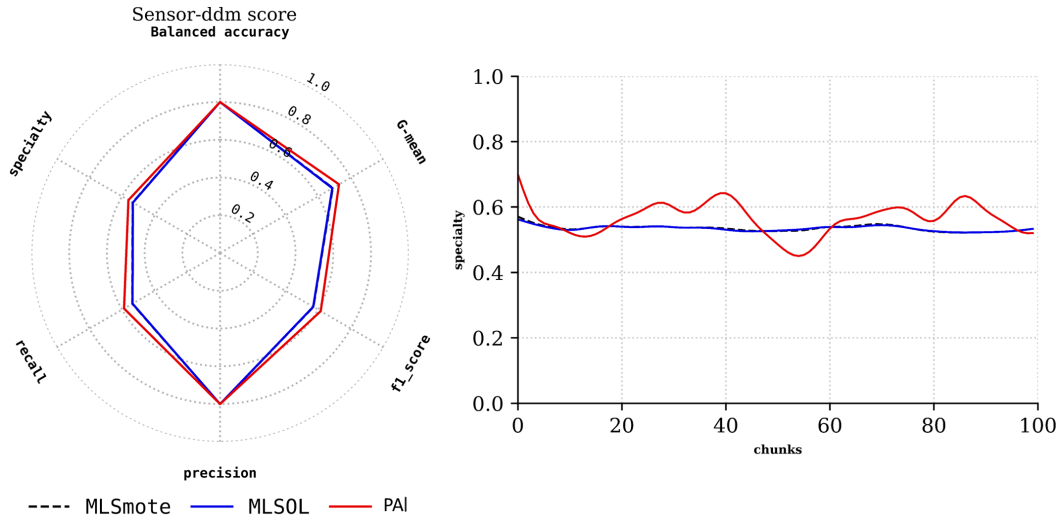

Figure 1.6: Comparison of PA1, MLSMOTE, and MLSOL on Sensor Dataset with DDM Concept Drift Detector

.

1.4.3.3 Results on the Synthetic Stream

Figures 1.7 and 1.8 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 1.7, 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 1.8 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.

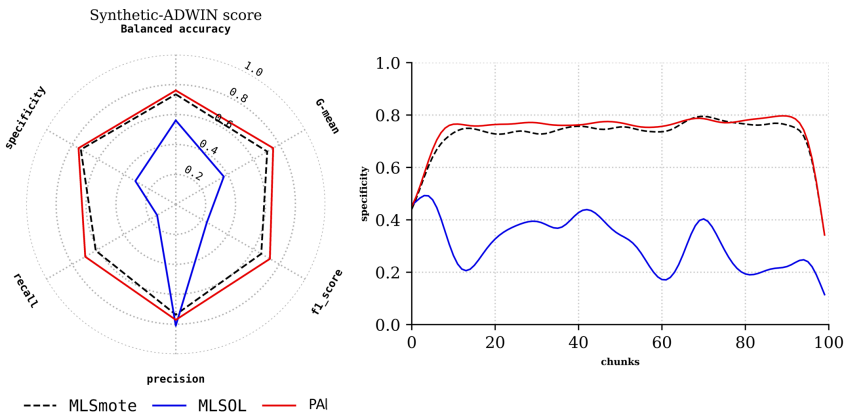

Figure 1.7: Comparison of PA1, MLSMOTE, and MLSOL on Synthetic Stream with ADWIN.

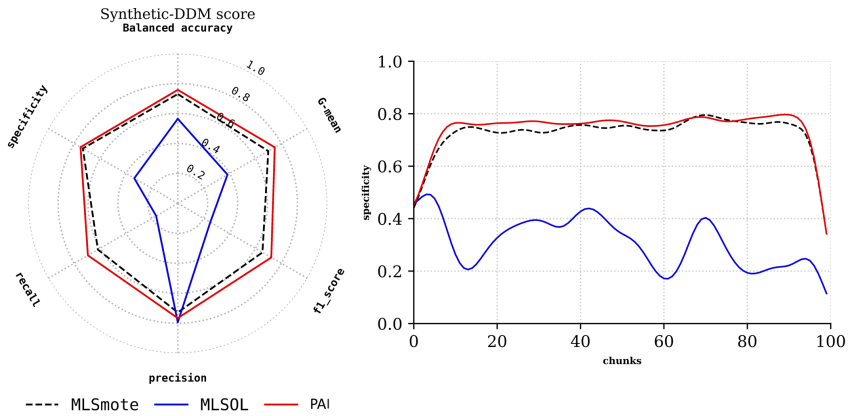

Figure 1.8: Comparison of PA1, MLSMOTE, and MLSOL on Synthetic with DDM.

1.4.4 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 1.9, 1.10, and 1.11 present the experimental outcomes. Each figure contains ten bar groups, where each group represents overlapping samples for chunks processed by MLSMOTE, MLSOL, and PA1.

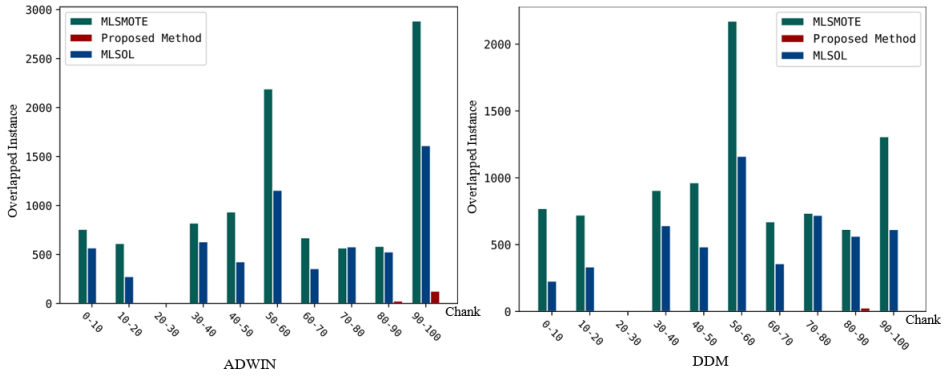

Figure 1.9: Overlapping Points of PA1, MLSMOTE, and MLSOL on Covertype Stream.

Figure 1.9: 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 1.10: 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 1.9, 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.

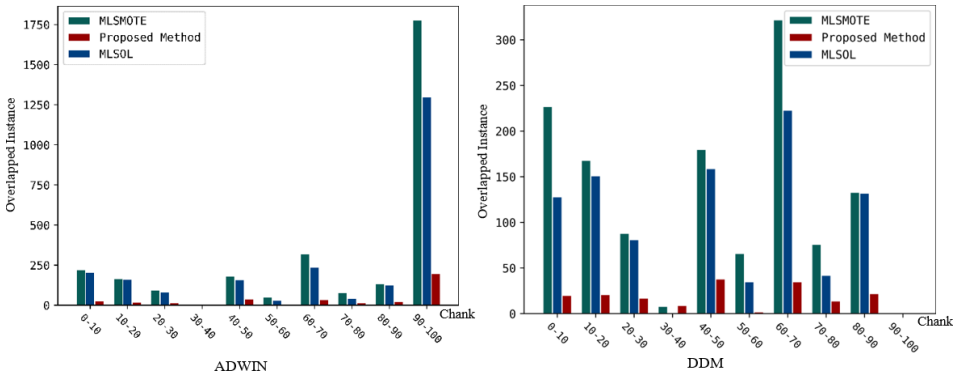

Figure 1.10: Overlapping Points of PA1, MLSMOTE, and MLSOL on Sensor Stream.

Figure 1.11: 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.

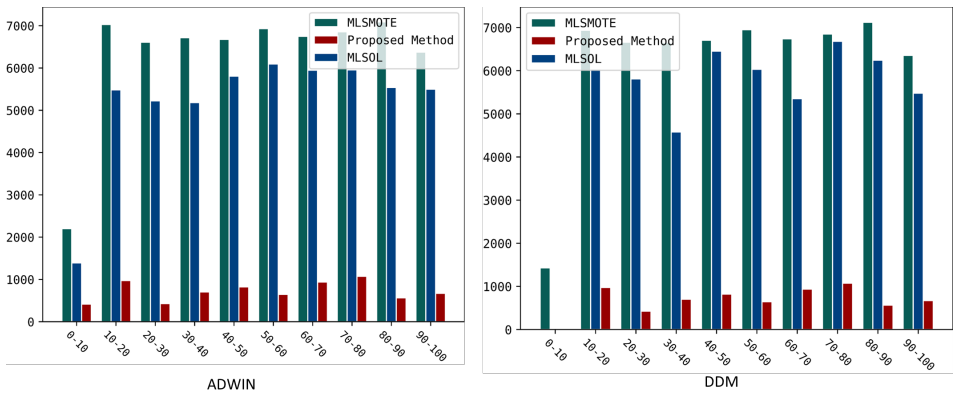

Figure 1.11: Overlapping Points of PA1, MLSMOTE, and MLSOL on Synthetic Stream.

1.4.5 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 1.2, 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 1.2: 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**

1.4.6 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 [**?**]. 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 1.3: 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 1.3 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 per-

formance, 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.

1.5 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.

2

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 [**?** **?** **?**]. 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 [**?**], [**?**], [**?**]. 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 abil-

ity 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 [**? ? ?**]. 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 [**? ? ?**]. 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 2.2 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 2.4 outlines the experimental setup and presents the results, providing details regarding the employed datasets, evaluation metrics, and procedures. Finally, Section 2.5 presents a summary of this chapter.

2.1 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.

2.2 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.

2.2.1 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 2.1 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. 2.1). These classifiers are used in the second phase to predict class labels, while drift detectors like ADWIN or DDM monitor for concept drift.

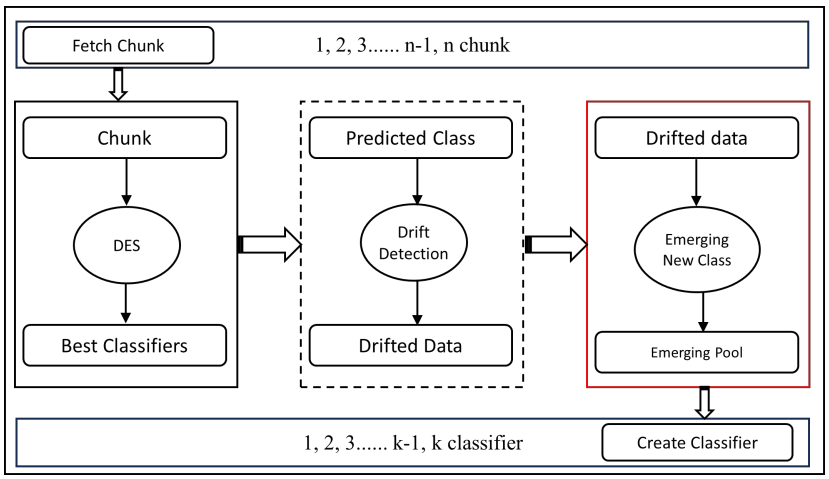

Figure 2.1: Flow Diagram of the Second Proposed Approach.

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. 2.2). 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).

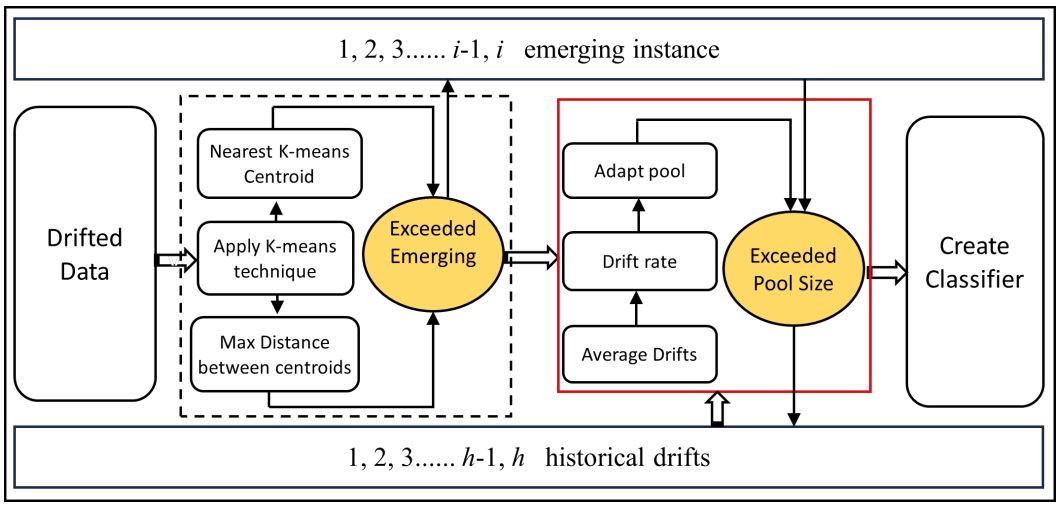

Figure 2.2: Flow Diagram of the Emerging Phase.

2.3 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 2.1, where d_c identifies the class centroid with the maximum Euclidean Distance (ED) between any two centroids c_i and c_j:

$$d_c = \arg\max_{d} \sum_{i=1}^{i} \sum_{j=i+1}^{i} ED(c_i, c_j) \tag{2.1}$$

Equation 2.2 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:

$$d_x = \arg\min_{i} \sum_{i=1}^{i} ED(x, c_i) \tag{2.2}$$

The decision-making process is governed by Equation 2.3, 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:

$$p = \begin{cases} EC & \text{if } d_x > d_c \\ P_{DES} & \text{otherwise} \end{cases} \tag{2.3}$$

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.

2.3.1 Emerging New Classes Phase Description

In this section,the Emerging new classes phase details is presented. Figure 2.2 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. 2.2) 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.2.1 and Eq.2.2, 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. 2.3; 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.

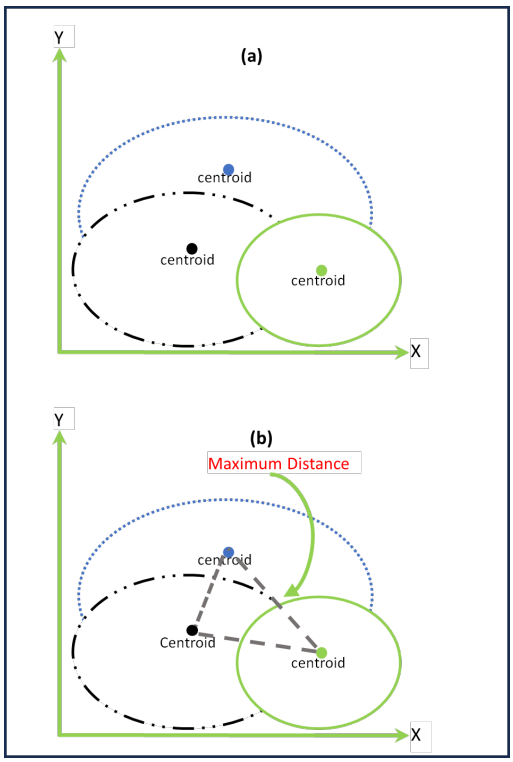

Figure 2.3: Steps for clustering a dataset with two features and three classes (Blue, Black, Green).

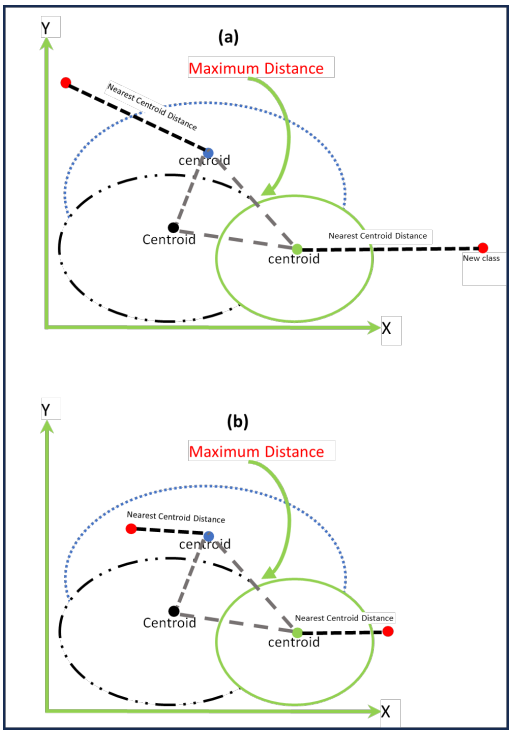

Figure 2.4: Steps for handling new points (red) with two features (x, y).

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 2.3 and 2.4. In Fig. 2.3 (a), the drifted chunk is divided into three clusters using the K-means algorithm. Figure 2.3 (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. 2.4 (a). If the distance is within the threshold, the instance is classified as a known class Fig. 2.4 (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

2.4 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.

2.4.1 Experimental Setup

The evaluation of the second proposed approach involves a comparison with SENC-Forest [**?**], SENNE [**?**], and KENNES [**?**]. The evaluation utilizes several metrics, including recall, precision, F_1 score [**?**], BAC [**?**], and G-mean [**?**]. The procedure for experimentation followed a test-then-train technique [**?**], 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 [**?**]. 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 [**?**] and DDM [**?**] 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/`

[**?** **?** **?** **?**]. 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].

2.4.2 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 [**?**] were used. As detailed in Table 2.1, 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 [**?**]. 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 2.1: 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`

2.4.3 Compared Approaches

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

1. **SENCForest [?]:** 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 [?]:** 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 [?]:** 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.

2.4.4 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, F_1 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.

2.4.4.1 Results On The Benchmark Dataset

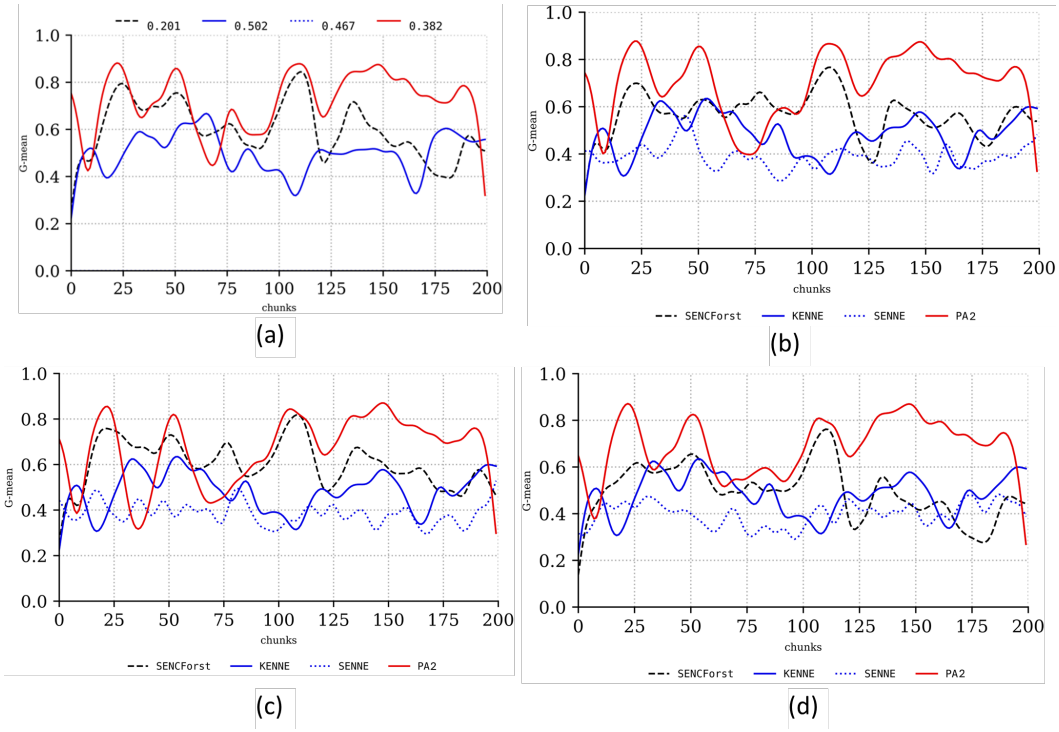

Figure 2.5: Covertype Stream Performance.

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 2.5, which depict the effects of emerging buffer sizes—5, 10, 20, and adaptive sizes—on performance. In the initial analysis (Fig. 2.5(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. 2.5(b) and Fig. 2.5(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. 2.5(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.

2.4.4.2 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 2.6, are visualized through scatter plots. In Figure 2.6(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 2.6(b) and 2.6(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 2.6(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.

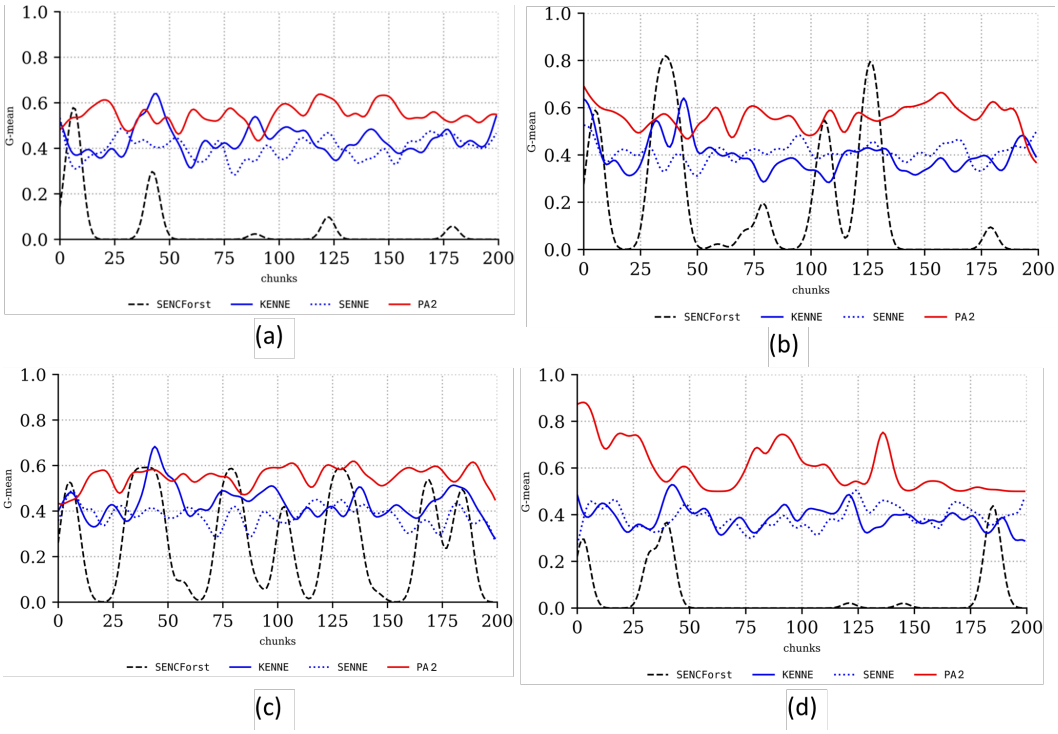

Figure 2.6: Sensor Stream Performance.

2.4.4.3 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 2.7, illustrate that the PA2 algorithm consistently achieved the highest performance across different buffer sizes of 5, 10, and 20. As shown in Figures 2.7(a), 2.7(b), and 2.7(c), scatter plots reveal PA2's superior performance in all these scenarios. Furthermore, Figure 2.7(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.

2.4.5 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 2.2. 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

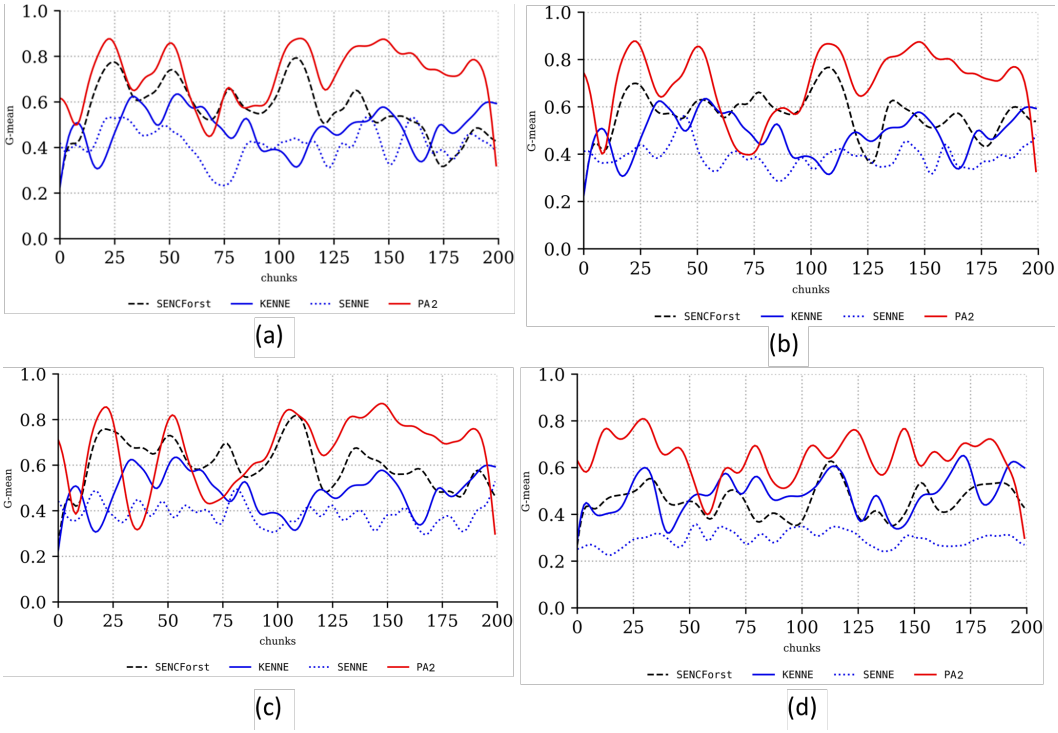

Figure 2.7: Synthetic Stream Performance.

Table 2.2. 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.

2.4.6 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 2.8, 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

Table 2.2: 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**

as balanced accuracy, precision, recall, G-mean, and F_1 score. Based on these results, GNB and HT were identified as the most suitable base classifiers for the bagging technique. Further comparisons between GNB and HT in terms of runtime and update frequency, as presented in Table 2.3, 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 2.3 and Figure 2.8. 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

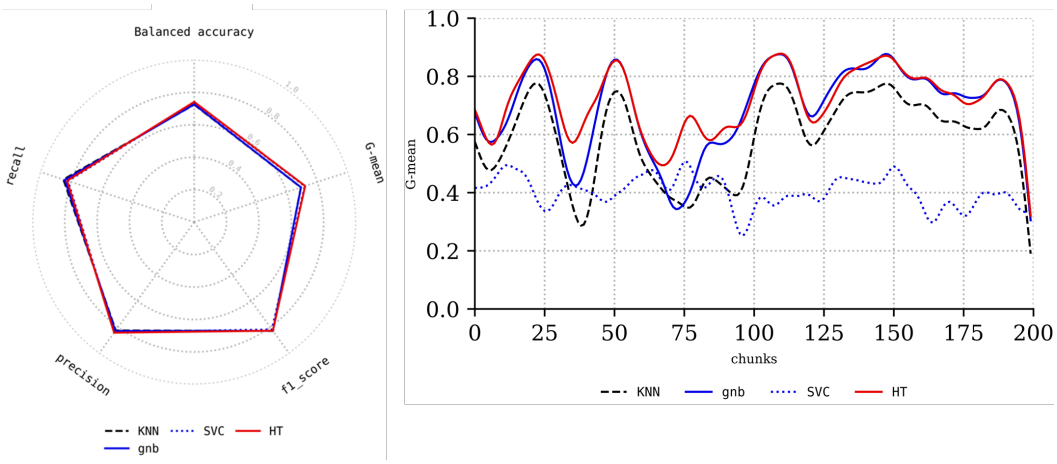

Figure 2.8: Performance Metrics for Covertype Stream with ADWIN and DDM.

Table 2.3: 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

solution often depends on the specific demands of the environment or problem at hand.

2.4.7 Comparison between ADWIN and DDM

In this section, two concept drift detection methods, the Drift Detection Method (DDM) [**?**] and the Adaptive Window (ADWIN) [**?** **?**], are compared. Both methods are highly regarded for their performance in managing incremental drifted streams [**?** **?** **?** **?**]. Figure 2.9 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 2.4, 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 reflec-

tion 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.

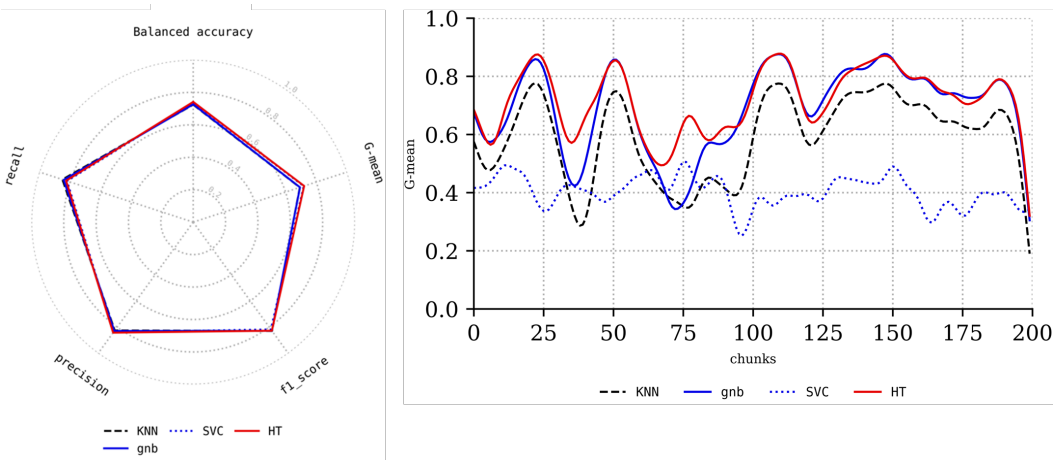

Figure 2.9: Performance Metrics for Synthetic Stream with ADWIN and DDM.

Table 2.4: Runtimes (in seconds) of ADWIN and DDM.

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

2.5 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.

3

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 [**?** **?**]. 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 [**?** **?** **?** **?**]. Additionally, mitigating negative knowl-

edge transfer often involves down-weighting irrelevant data sources to prevent the incorporation of misleading information [**?**]. 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 [**? ?**]. 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 [**? ? ?**]. 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 [**?**]. Additionally, the Streaming Ensemble Algorithm (SEA) adapts classifier ensembles to real-time data streams, maintaining robustness as data conditions and new classes emerge [**? ? ?**]. 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 3.1 presents the motivations and contributions. Section 2.2 introduces the third proposed approach, providing intricate explanations of its constituents, including dynamic classifier ensembles, concept drift handling, and heterogeneous multisource transformation. Section 1.4 outlines the experimental setup and presents the results, providing details regarding the employed datasets, evaluation metrics, and procedures. Finally Section **??** presents a summary of this chapter.

3.1 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 contribu-

tions 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.

3.2 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.

3.2.1 HTL Description

The aim of this section is to harness knowledge from diverse dimensional multi-source domain streams. Following a similar framework to CDTL [**?**], 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. 3.1, 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.

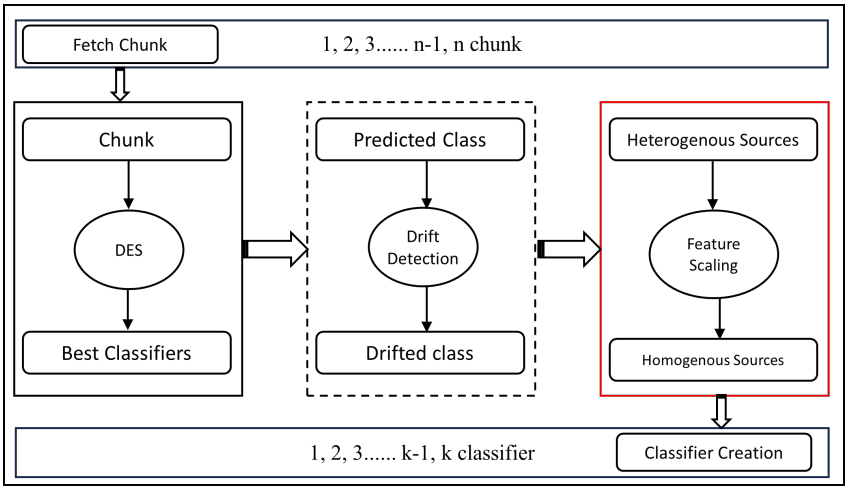

Figure 3.1: HTL Flow Diagram.

- **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.

3.2.2 Classifier Creation Phase Description

Figure 3.2 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 [**?**], 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.

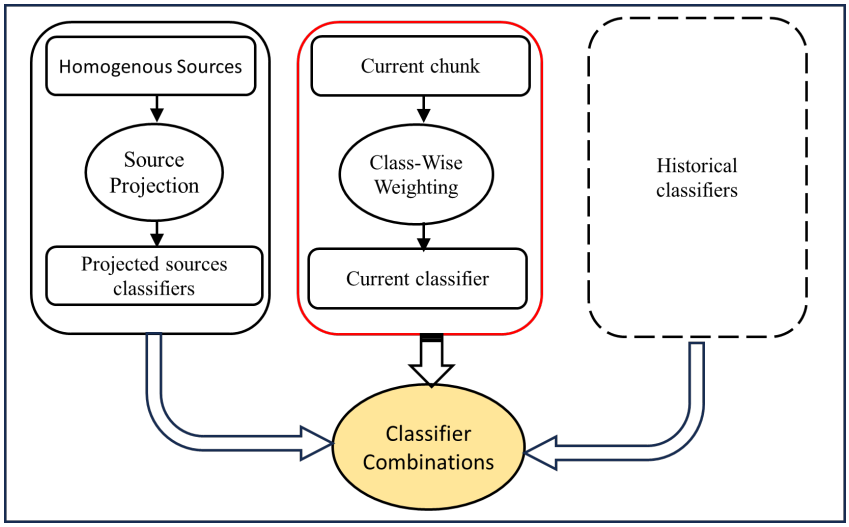

Figure 3.2: Classifier Creation Phase Flow Diagram.

- **Class-wise weights:** This block computes class-specific weights for each classifier using Equations 3.1 and 3.2 . 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. 3.3 , 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.

3.3 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 $\mathcal{D}$:

$$classWeight_{k,c,\mathcal{D}} = \sum_{i \in \mathcal{D}} \frac{|prediction_i = c|}{|\mathcal{D}|} \times \frac{|prediction_i \neq c|}{|\mathcal{D}|}. \tag{3.1}$$

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:

$$classifierWeight = \sum_{k \in K} \sum_{c \in C} classWeight_{k,c,\mathcal{D}}, \quad \text{where } \mathcal{D} = 1, 2, 3, \ldots, N. \tag{3.2}$$

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:

$$Prediction_{p,H,k,chunk} = \sum_{p \in P} prediction^{p}_{chunk} + \sum_{h \in H} prediction^{h}_{chunk} + prediction^{k}_{chunk}. \tag{3.3}$$

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.

3.3.1 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 [**?**].

- **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.

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
```

3.4 Experimental Results

In this section, the proposed HTL, CORAL [**?**], and CDTL [**?**], and Melanie [**?**] 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.

3.4.1 Experimental setup

The evaluation of Heterogeneous Taxonomy Learning (HTL) involves a comparison with CORAL [**?**], CDTL [**?**], and Melanie [**?**] which employs multiple metrics such as precision, recall, F_1 score [**?**], BAC [**?**], and G-mean [**?**]. The experimental protocol employed for evaluation followed the test-then-train approach [**?**], 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 [**?**]. 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

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

related work primarily focus on homogeneous source domains and do not address the challenges posed by heterogeneous multisource data. Therefore, heterogeneous experiments were conducted.

3.4.1.1 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 [**?** **?**] 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 3.1: 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

3.4.1.2 Compared Approaches

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

- **AW-CORAL [?]:** 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 [?]:** 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 [?]:** 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 [**?**] and HE-CDTL [**?**].

3.4.2 Analysis of Experimental Results

This section evaluates the proposed approach's performance across multiple data streams using five metrics: F_1 score, precision, recall, G-mean, and BAG. Two visualization techniques, radar and line diagrams, were employed. Radar charts summarized algorithm performance across metrics, while line diagrams focused on G-mean comparisons across 100 chunks. The first five experiments used line diagrams, and the last two combined radar and line charts, offering a detailed evaluation of the third proposed approach under diverse scenarios.

3.4.2.1 Results on Target Domains Dataset without Source Domain

Figure 3.3 presents the results of applying various classification methods to the Covertype dataset without employing a source domain stream. The line diagram reveals the classification performance, assessed using the G-mean metric, over 100 data chunks for each method. Philosophically, the observed trajectory of performance reflects an epistemological journey from initial uncertainty to eventual refinement and adaptation. The suboptimal performance during the initial 10 chunks suggests a nascent phase where the system's understanding is limited, akin to the formative stages of knowledge acquisition. This phase is marked by an incomplete pool of classifiers, signifying the constraints of limited epistemic resources. As the classifier pool expands, representing an enrichment of the system's cognitive tools, the DES technique demonstrates an enhanced ability to discern and apply the most suitable classifier for each incoming data chunk. This mirrors the philosophical principle of cumulative epistemic progress, where successive integration of knowledge resources fosters improved understanding and capability. Between chunk 20 and the final chunk, HTL consistently outperforms other methods. This superiority is attributable to its weighting mechanism, which assigns significance to historical classifiers, embodying a philosophical stance that values historical precedent and contextual relevance in constructing present knowledge. The ability of HTL to effectively leverage this weighting method facilitates the training of the classifier pool, thereby amplifying its overall efficacy. In contrast, the variable performance of CORAL and CDTL in certain chunks highlights the inherent challenges

of reconciling differing approaches to knowledge adaptation. This variability underscores the dynamic interplay of methodologies, each striving to balance generality and specificity in their engagement with a continually evolving data stream.

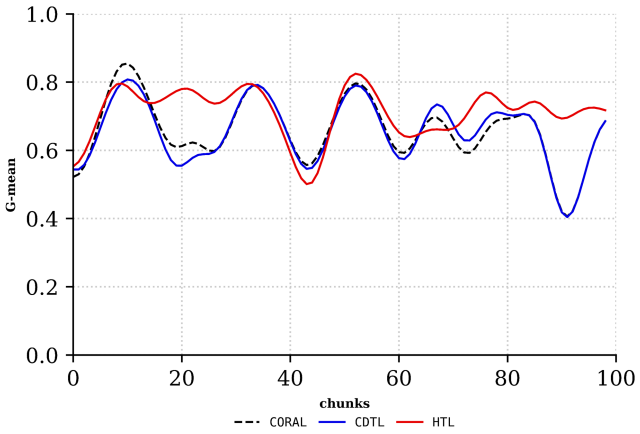

Figure 3.3: Results of the Covertype Stream as the Target Domain without the Source Domain.

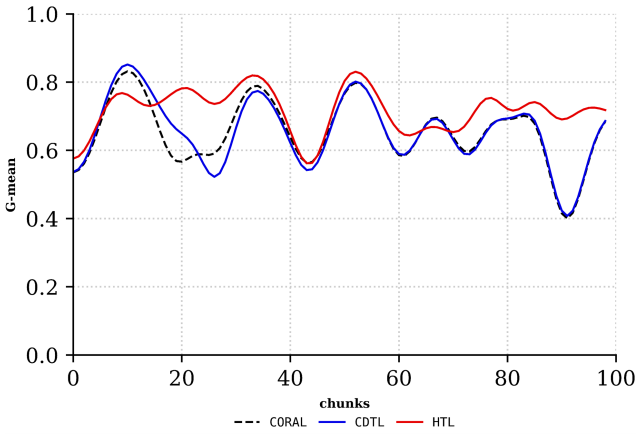

Figure 3.4: Performance results of the Covertype Stream as the target domain using homogeneous source domains.

3.4.2.2 Results on the Homogeneous Source Domains Dataset

Figure 3.4 illustrates the outcomes of applying the compared methods to the Covertype data stream as a target domain, with Synthetic-52 serving as a homogenous source domain. This scenario introduces an epistemological dimension rooted in

the interaction between prior knowledge (the source domain) and newly encountered contexts (the target domain). The line diagram reveals HTL's initially suboptimal performance across the first eight chunks, attributable to the limited pool of classifiers available in the early phase. This aligns with the philosophical notion of an incomplete epistemic framework, where insufficient resources impede initial understanding. As the classifier pool expands beyond the eighth chunk, HTL's performance improves, underscoring the significance of cumulative resource enrichment in advancing predictive accuracy. CDTL maintains consistent performance throughout the chunks, reflecting an equilibrium-based adaptation strategy that prioritizes stability over dynamic optimization. Meanwhile, CORAL's consistently lower performance highlights the potential limitations of certain methodologies in transferring knowledge effectively across domains, pointing to the philosophical challenge of achieving universality in adaptive systems.

Notably, HTL achieves the highest performance across all chunks, emphasizing the utility of its weighting mechanism and its capacity to harness the source domain knowledge effectively. The superior performance of both CDTL and HTL in Figure 3.4 compared to Figure 3.3 reflects the epistemological advantage conferred by incorporating Synthetic-52 as a source domain. This source-target synergy illustrates the philosophical principle of grounded learning, where external, homogenous knowledge sources enhance the capacity to navigate and adapt to complex, evolving data streams.

The contrasting scenarios in Figures 3.3 and 3.4 underscore the critical role of prior knowledge in shaping learning trajectories. In the absence of a source domain, as in Figure 3.3, the methods must independently construct their epistemic framework, leading to comparatively constrained performance. Conversely, the inclusion of a source domain, as shown in Figure 3.4, facilitates a richer epistemic interplay, enhancing adaptability and robustness in the face of dynamic data challenges.

3.4.2.3 Results on One Heterogeneous Source Domain

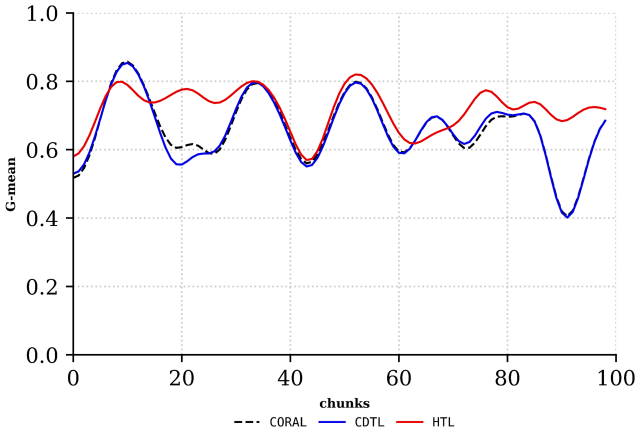

Figure 3.5: Performance results of the Covertype Stream as the target domain with a single heterogeneous source domain.

Figure 3.5 presents the results of applying the compared methods to the Covertype data stream as the target domain, with Synthetic-8 as a heterogeneous source domain. This experiment explores the epistemological dynamics of knowledge transfer across domains with differing structures, reflecting the philosophical challenges of reconciling heterogeneity in adaptive systems. The Synthetic-8 stream, with its distinct dimensionality (eight features and four classes), contrasts with the Covertype stream. While HTL is inherently designed to handle heterogeneous sources, specific adjustments, such as the application of the eigenvector technique, were required to enable CORAL and CDTL to operate within this domain. This adaptation highlights the importance of methodological flexibility in addressing epistemic diversity and bridging structural disparities between domains. The line diagram demonstrates variability in the performance of the approaches across chunks, suggesting that the contribution of the source domain's knowledge is contingent on its alignment with the target domain's requirements. HTL consistently achieves superior performance across almost all chunks, emphasizing its robust capacity for extracting and applying positive knowledge from heterogeneous sources. This superior adaptability reflects a philosophical commitment to leveraging epistemic pluralism, where diverse knowledge sources enrich the learning system's overall efficacy.

In contrast, the comparatively lower performance of CORAL and CDTL underscores the inherent challenges these methods face when transferring knowledge across domains with significant structural differences. The need for eigenvector-based adjustments further emphasizes their limited native capacity for heterogeneous adaptation, revealing the philosophical tension between static methodologies and dynamic, cross-domain requirements. This experiment exemplifies the critical role of epistemological adaptability in heterogeneous environments. The consistent success of HTL underscores the potential of methodologies that embrace diversity in knowledge sources, whereas the relative struggles of CORAL and CDTL highlight the constraints of approaches rooted in homogeneity.

3.4.2.4 Results on All Heterogeneous and Covertype as the Target Domain

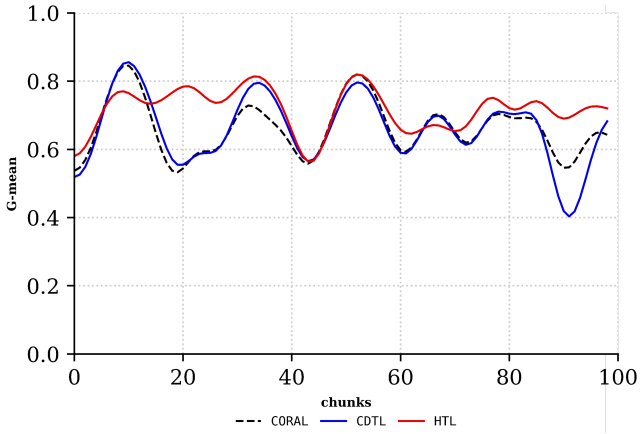

Figure 3.6: Performance results of the Covertype Stream as the target domain with multiple heterogeneous source domains.

Figures 3.6 and 3.7 delve into the impact of incorporating multiple heterogeneous source domains on the classification performance of various methods, offering a nuanced philosophical reflection on the interplay between diversity, adaptability, and resilience in dynamic environments. In Figure 3.6, the Covertype data stream serves as the target domain, while Synthetic-8, Synthetic-52, and Sensor streams act as heterogeneous source domains. The results demonstrate a notable improvement over previous experiments, underscoring the epistemic value of increasing

the number of source domains. This enhancement reflects the philosophical principle of cumulative epistemic diversity: as the pool of source domains expands, the system gains access to a broader spectrum of positive knowledge. Each additional domain contributes incrementally to the refinement of classifiers, facilitating more accurate and adaptive performance.

Figure 3.7, employing the same experimental setup but switching the target domain to the Sensor stream, highlights the influence of contextual factors on performance. The Sensor stream introduces higher noise levels and frequent concept drifts, presenting significant challenges to all methods. Despite these adversities, HTL consistently achieves superior performance across all chunks, reinforcing its adaptability and capacity to navigate complex and unstable environments. This robustness aligns with a philosophical framework that values resilience and context-sensitive adaptation in epistemic systems. In contrast, CORAL and CDTL exhibit nearly identical performance values across most chunks, indicating a more static approach to adaptation. Their inability to capitalize fully on the heterogeneity of source domains or mitigate the challenges posed by the Sensor stream underscores the philosophical tension between generality and specificity in dynamic learning systems. While they may perform adequately in stable or homogenous contexts, their relative rigidity limits their efficacy in environments marked by noise and drift.

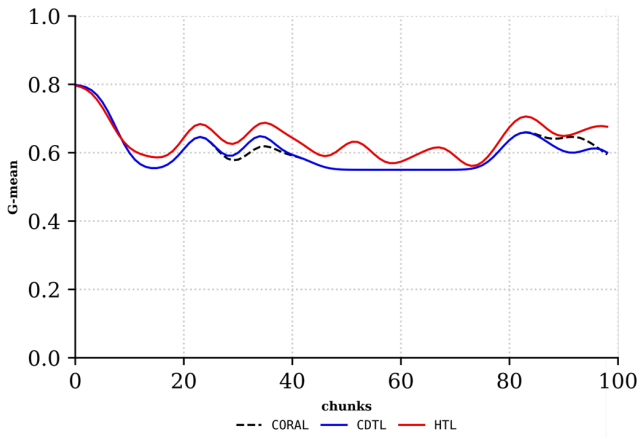

Figure 3.7: Results of the Sensor Stream as the Target Domain with Multiple Heterogeneous Source Domains.

3.4.2.5 Results on One Heterogeneous Source Domain

This section evaluates the overall performances of CORAL, CDTL, and HTL across five performance metrics: F_1 score, precision, recall, G-mean, and BAG. Table 3.2 summarizes the average performance across 100 data chunks for each method and highlights HTL's consistent superiority in most configurations.

In the first experiment, which lacked a source domain, CORAL achieved the highest precision, while HTL excelled across the other metrics. In subsequent experiments, HTL consistently outperformed other methods in most metrics, except for recall in the third experiment, where CDTL achieved the best results. The performance improvement across experiments reflects the incremental addition of source domains, which enhances the positive knowledge available to the classifiers, thereby improving overall performance.

Table 3.2: 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**
	F_1 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**
	F_1 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**
	F_1 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**
	F_1 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**
	F_1 score	0.997	0.997	**0.997**
	Precision	0.998	0.998	**0.998**
	Recall	0.998	0.998	**0.998**

3.4.3 Analysis Runtime between CORAL, CDTL, and HTL Techniques

From a philosophical perspective, the experimental results underscore the nuanced interplay between technological efficiency and contextual suitability in heterogeneous transfer learning. The choice of an optimal algorithm emerges not as an absolute truth but as a contextual decision shaped by the characteristics of the dataset and the temporal demands of the application. The study's findings illuminate the inherent efficiency of the HTL algorithm, particularly in scenarios demanding rapid

iteration. For instance, the HTL algorithm's completion of 100 iterations within 304 seconds in the homogeneous source domain setting starkly contrasts with the prolonged runtimes of CORAL and CDTL algorithms. This efficiency is not merely a quantitative advantage but symbolizes the evolving imperative of time-conscious adaptability in machine learning [**?**]. The philosophical significance deepens when considering the diverse source domain configurations—homogeneous, absent, single heterogeneous, and multi-source heterogeneous—each representing varying degrees of complexity and unpredictability. The HTL algorithm's consistent efficiency across these scenarios reflects a harmonious balance between universality and specificity. It challenges the idea of algorithmic rigidity, suggesting that adaptability and context-awareness are the cornerstones of effective machine learning paradigms [**?**]. Thus, the findings, accentuated by the runtime data in Table 3.3, reveal the HTL algorithm as not merely a technical tool but a conceptual embodiment of efficiency tailored to the exigencies of dynamic, real-world applications. In this light, its superiority is not only functional but philosophical, aligning with the broader pursuit of practical excellence in algorithmic design and implementation.

Table 3.3: 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**

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

This experiment explores the temporal dynamics of various learning techniques, comparing the efficiency of online learning with chunk-based concept drift methods. The time savings provided by drift detectors like EDDM and ADWIN offer insights into the relationship between learning time and performance. The EDDM drift detector is notably efficient, detecting 40 drifted chunks out of 100, which highlights a pragmatic approach to reducing computational overhead while maintaining accuracy. This efficiency aligns with the concept of "pragmatic optimiza-

tion," aiming to balance time and performance within real-world constraints [**?**]. In contrast, the ADWIN drift detector, which detects 63 drifted chunks and requires 63 learning times, demonstrates a trade-off between improved drift detection and time efficiency. This suggests that optimal performance is often context-dependent, with the most efficient algorithm not necessarily being the one that minimizes time, but one that balances all relevant factors. On the other hand, online learning, requiring 100 learning times for 100 chunks, proves to be the least time-efficient. However, its design may reflect a commitment to consistency and robustness, even at the expense of time. Figures 3.8 and 3.9, which include radar and line diagrams, visually capture the tension between time efficiency and performance. While online learning achieves optimal performance, its higher time consumption highlights the conflict between pursuing perfection and managing resource limitations. This analysis emphasizes that efficiency is not solely about minimizing time but involves balancing multiple factors in real-world applications.

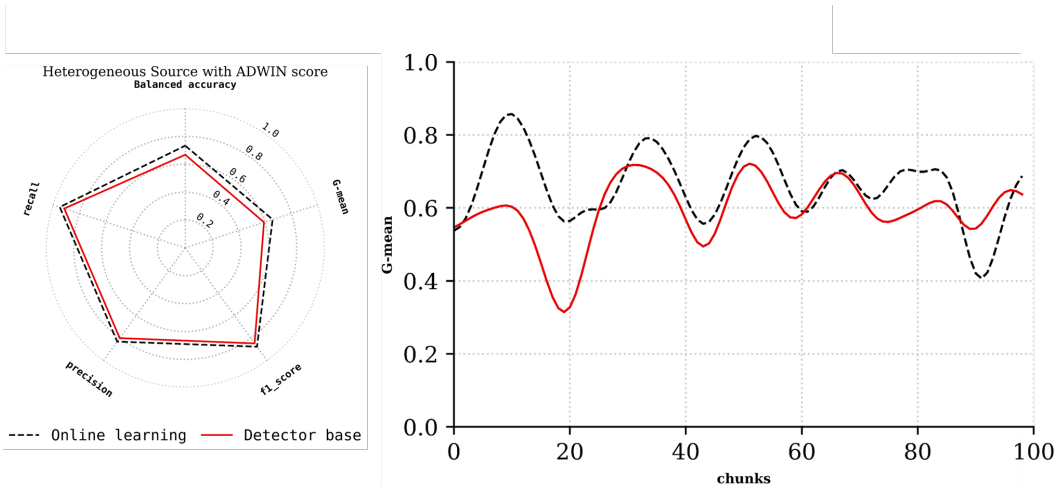

Figure 3.8: Online learning and chunk-based results for the Covertype stream using the ADWIN detector.

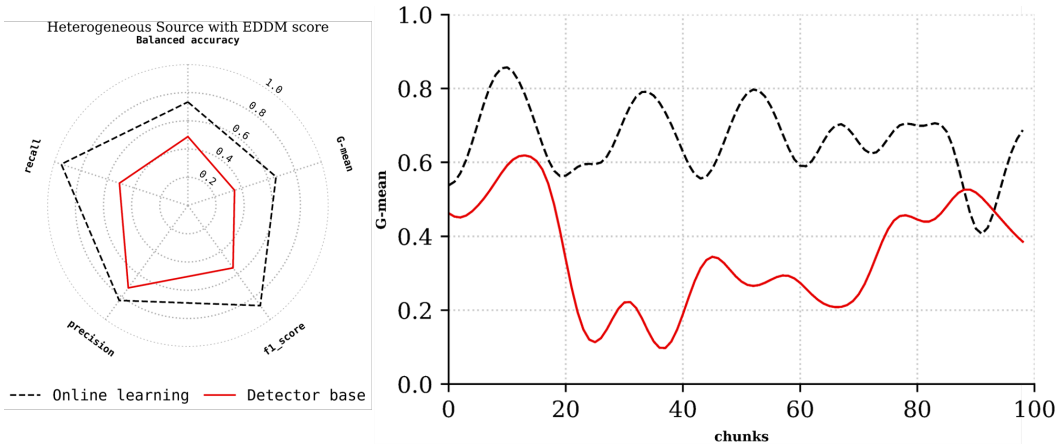

Figure 3.9: Results of online learning and chunk-based detection in the Covertype stream using the ADWIN detector.

The comparison between online learning, the EDDM detector, and ADWIN reflects a broader philosophical consideration of efficiency, adaptability, and the balance between idealism and practicality. Online learning, with its ability to achieve optimal performance, represents an ideal approach—constantly evolving, adapting to real-time data without reliance on prior knowledge. This continuous adaptation mirrors the notion of becoming, where the system remains in a state of perpetual growth and refinement. The radar diagrams in Figure 3.9 further emphasize the superiority of online learning, positioning it as the most effective method. ADWIN, in this context, stands as a favorable compromise, balancing the strengths of online learning with the practicality of reduced time expenditure. ADWIN thus exemplifies the concept of harmony, where competing forces of performance and efficiency are reconciled, offering an adaptable solution that doesn't strive for perfection but achieves a satisfactory balance. In contrast, the EDDM detector, with its lower performance, reflects the limitations of rigidity—remaining static in the face of evolving data, it fails to respond effectively to dynamic changes. This serves as a reminder of the consequences of inflexibility in systems, where an inability to adapt leads to diminished overall effectiveness. Ultimately, in environments characterized by chunk-based concept drift, especially with the ADWIN detector, the approach that ensures high performance while minimizing computational costs reveals itself as ideal. This balance speaks to the philosophical principle of achieving

optimal results through continuous yet measured adaptation [**?** **?**].

Table 3.4: 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

4

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.

4.1 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.

4.2 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.