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 offer the unique ability to adapt their composition based on data characteristics, making them valuable in situations with evolving data conditions [**?**]. The aim of classifier ensemble selection is to identify the optimal subset of classifiers from a larger ensemble. A prominent approach in classifier ensemble selection is the overproduce-and-select strategy. This selection process is guided by diverse criteria, including individual performance measures, diversity metrics, meta-learning techniques, and performance-estimation approaches. Such optimization is particularly crucial in scenarios in which striking a balance between accuracy and computational resource constraints is paramount. There are two distinct approaches: static and dynamic approaches. Static selection involves assigning classifiers to predefined feature partitions, whereas dynamic selection adaptively selects classifiers based on their competency [**?**]. Dynamic selection offers two choices: Dynamic Classifier Selection (DCS) and Dynamic Ensemble Selection (DES). DCS algorithms enable the selection of the most appropriate classifier for each data point, based on its local competencies. In contrast, DES focuses on selecting the optimal classifiers for each instance based on their competence within localized regions [**? ? ?**]. Competency assessment relies on a Dynamic SELection dataset (DSEL) containing labeled samples. Moreover, innovative techniques, such as the randomized reference classifier, introduce randomness into class supports to enhance adaptability in addressing the challenges related to imbalanced data. 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, we employed dynamic classifier ensembles. These ensembles utilize oversampling techniques, implemented either on a global or local scale, as a preprocessing step to address class imbalance. Furthermore, we enhanced multiclass learning techniques to counteract class imbalance through method adaptation.

The remainder of this chapter is organized as follows: In Section 1.1, we present the motivations and the contributions. The first proposed framework are discussed in detail in Section 1.2. The experimental results and the discussion are presented in Sections 3.3. Finally, the conclusions of this study and future research are discussed in Section **??**.

1.1 Motivations and Contributions

1. We introduce a classification approach 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 accuracy, specifically for multiclass imbalanced non-stationary streams.
2. Additionally, we propose an adaptive method 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

In this section, we present the primary phases of our study, comprising three distinct stages. Following this introduction, we delve into the synthetic data-generator method, providing 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. In pursuit of this goal, our proposal addresses the four primary challenges inherent in constructing our approach.

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

- **Overlapping class:** We also address a critical challenge involving class overlapping, a factor known to substantially affect model performance [**? ?**]. To address this issue, we introduce an adaptive method that generates nonoverlapping 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, our 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 accuracy and robustness.

1.2.1 Approach Overall Details

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 our 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 our 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.

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. 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). 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. As shown in Fig. 1.2, 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 our 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

Fig. 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.

- **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 accuracy.

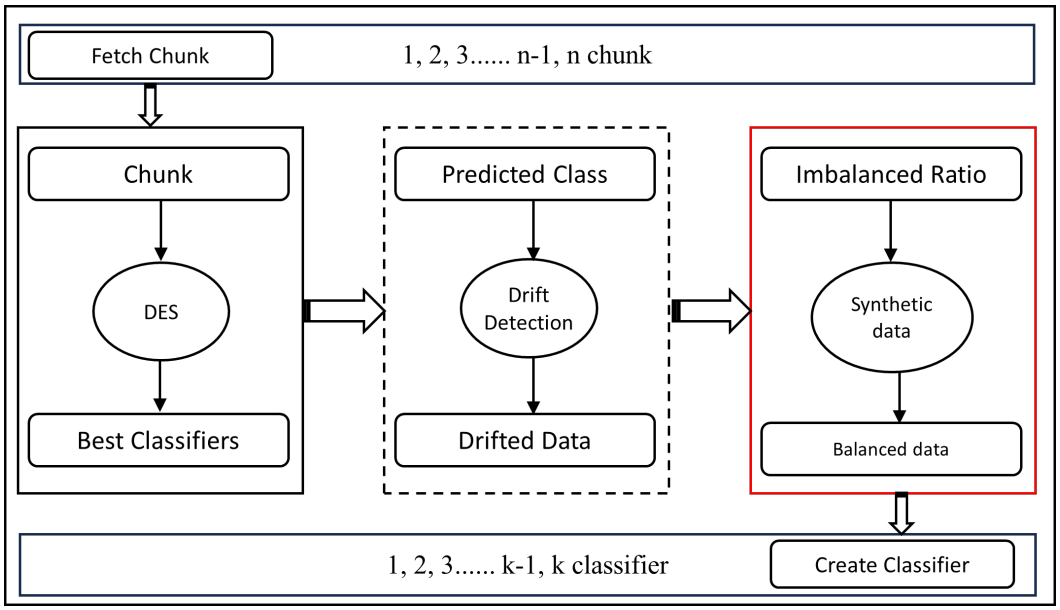

Figure 1.1: First Proposed Approach (PA1) for imbalanced multi-class drifted data streams.

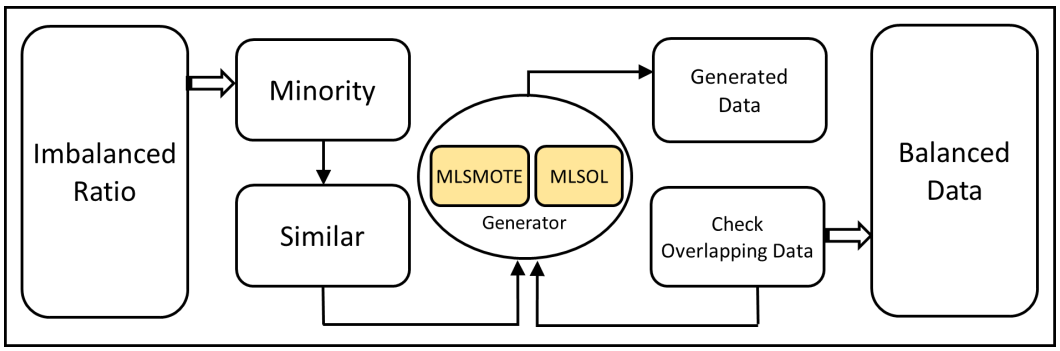

Figure 1.2: Flow of the synthetic data generator.

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

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

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

Algorithm 1: First proposed approach algorithm for imbalanced multi-class drifted data 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

In Algorithm 2, also known as the Synthetic Data Generator, we input three essential elements: the minority class samples, the current data chunk, and the de-

sired size for generating synthetic data. The primary objective of this algorithm is to produce synthetic data samples. To achieve this, we employ the KNN algorithm 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 accuracy 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.

Algorithm 2: Synthetic data generator.

Input: Minority classes μ, current chunk a, sample size η, historical chunks h
Output: Generated data b

```
b ← ∅;
f ← MLSMSOTE;
knn ← kNearestNeighbor(a);
chunk ← similarChunk(a, h);
f ← similarChunkOverSamplingMethod(chunk);
if f = MLSMSOTE then
    f ← MLSOL;
else
    f ← MLSMSOTE;
while |b| < η do
    p ← generateSyntheticPoint(μ, f);
    similarPointsClass ← KNN.getKneighbor(b);
    if similarPointsClass = μ then
        b ← b ∪ {p};
return b;
```

1.3 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 accuracy 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. The two main questions to be answered are:

- Q_1: What is the impact of reduced and consistent data on the performance of ensemble learning?
- Q_2: Is it possible with the search capability of swarm intelligence to enhance the combination of classifiers?

1.3.1 Experimental setup

The evaluation of the first proposed approach incorporated the utilization of various metrics such as recall, precision, specificity, f1 score, balanced accuracy score (BAC), and geometric mean score (G-mean) [**?**]. 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. We employed four classification classifiers 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] . We conducted a comparison between multiclass oversampling techniques (MLSMOTE and MLSOL) and first proposed approach to demonstrate the effectiveness of our contribution. Additionally, we conducted these experiments using two different concept drift detectors, ADWIN [**?**] and DDM [**?**], to demonstrate the adaptability and robustness of first proposed approach across varying drift detectors.

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

1.3.2 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: Characteristics of the datasets used in the experiments.

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

1.3.3 Analysis of Experimental Results

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

was calculated to present the overall performance of each method (MLSMOTE, MLSOL, PA1). It is important to note that the PA1 is represented by red lines.

1.3.3.1 Results on the Benchmark Stream

The results of the mentioned methods (MLSMOTE, MLSOL, PA1) applied to the Covertype dataset are presented in Fig. 1.3 , utilizing ADWIN as the drift detector. The radar diagram shows that the metric values were between 0.8 and 1.0. MLSOL exhibited the highest precision, whereas MLSOTE and PA1 had nearly identical values. However, PA1 excels in other metrics, whereas MLSMOTE has the lowest values. The line diagram in Fig. 1.3 shows the classification accuracy across 100 data chunks using the specificity metric for the methods in each chunk. Notably, all methods exhibit suboptimal accuracy during the first 20 chunks. However, a noticeable improvement was observed beyond this initial phase. The key factor driving this improvement was the expansion of the classifier pool, which now encompasses a growing number of classifiers. This expansion enables the Dynamic Ensemble Selection (DES) technique to become more proficient in selecting the most suitable classifier for each incoming chunk. Consequently, accuracy experienced a significant boost in later chunks, reflecting the adaptability and effectiveness of the ensemble approach. From chunk 20 to the last chunk, PA1 achieved the highest accuracy, whereas MLSMOTE recorded the lowest accuracy. PA1's superior performance of the PA1 is credited to its use of historical chunks to generate optimal nonoverlapping samples, thereby effectively training the pool classifiers. In Fig. 1.4 , the same dataset is employed with the DDM functioning as the drift detector. The radar plot illustrates results comparable to those of the prior experiment, whereas the line diagram underscores PA1's supremacy in most chunks. However, it also reveals that all approaches demonstrate diminished performance, in contrast to Fig. 1.3 (ADWIN), suggesting that ADWIN surpasses DDM when utilized in the Covertype data stream.

1.3.3.2 Results on the Real Application Stream

Fig. 1.5 illustrates the outcomes of employing the three methods on the Sensor data stream using ADWIN as the drift detector. The radar graph depicts metric val-

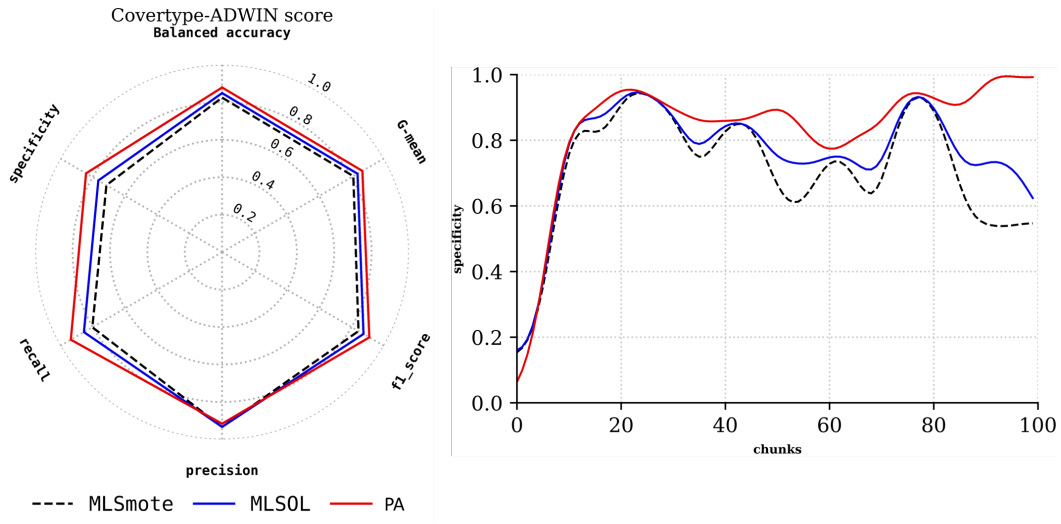

Figure 1.3: Performance of the first Proposed Approach (PA1), MLSMOTE, and MLSOL techniques on the Covertype Benchmark dataset using the ADWIN concept drift detector.

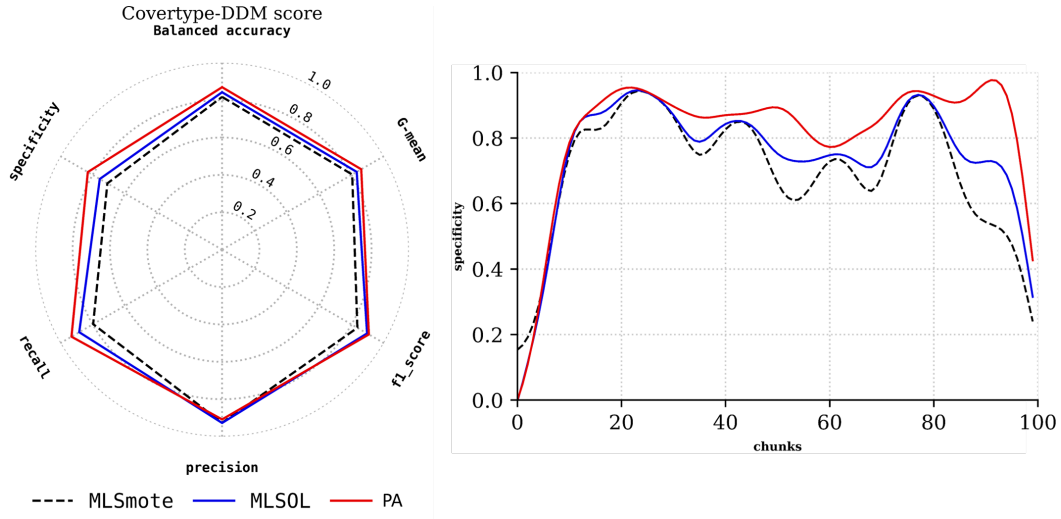

Figure 1.4: Performance of the first Proposed Approach (PA1), MLSMOTE, and MLSOL techniques on the Covertype Benchmark stream using the DDM concept drift detector.

ues ranging from 0.6 to 0.8, which indicate the pronounced drift and imbalance of the Sensor stream. In terms of the precision and recall metrics, the three methods exhibited almost identical values, whereas PA1 stood out in the other metrics. In

contrast, MLSMOTE and MLSOL displayed similar values. By examining the line graph in Fig. 1.5 , it is evident that during the initial 30 chunks, PA1's performance of PA1 might be suboptimal in some instances because of the limited number of classifiers in the pool. However, after the first 30 chunks, the performance significantly improved with the addition of more classifiers. In the same graph, PA1 consistently achieved the highest performance across all chunks, whereas MLSMOTE exhibited lower performance in certain chunks, and MLSOL exhibited lower performance in the other chunks. In Fig. 1.6, using the same dataset with the DDM as the drift detector, the radar plot metrics indicate lower results compared to the previous experiment. Nonetheless, the line graph underscores PA1's dominance of PA1 in most chunks, although all methods exhibit nearly identical performance across the entire set of chunks. Overall, the performance of all the methods in Fig. 1.6 is inferior to that in Fig. 1.5 (ADWIN), which highlights ADWIN's superiority of ADWIN over DDM when applied to the Sensor data stream.

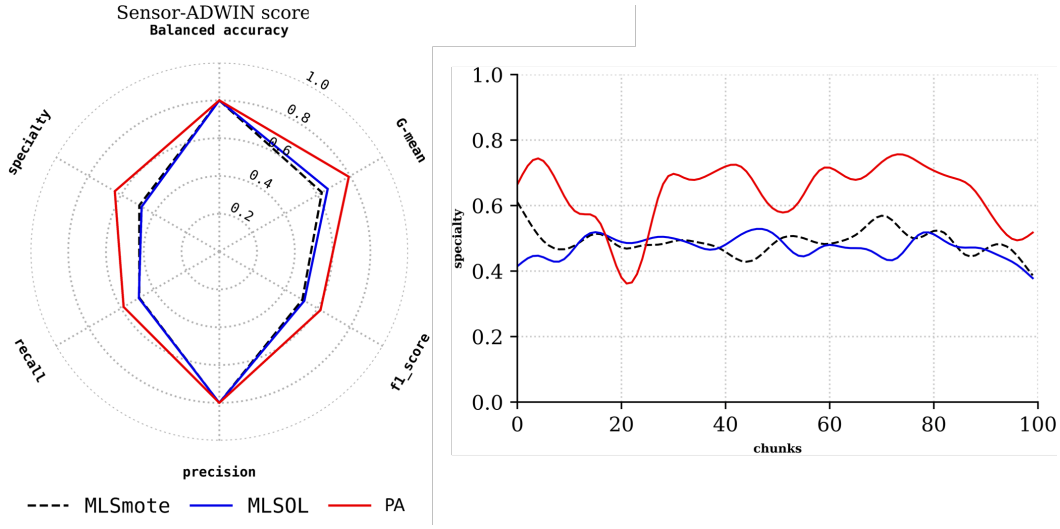

Figure 1.5: Performance of the first Proposed Approach (PA1), MLSMOTE, and MLSOL techniques on the Sensor real application stream using the ADWIN concept drift detector.

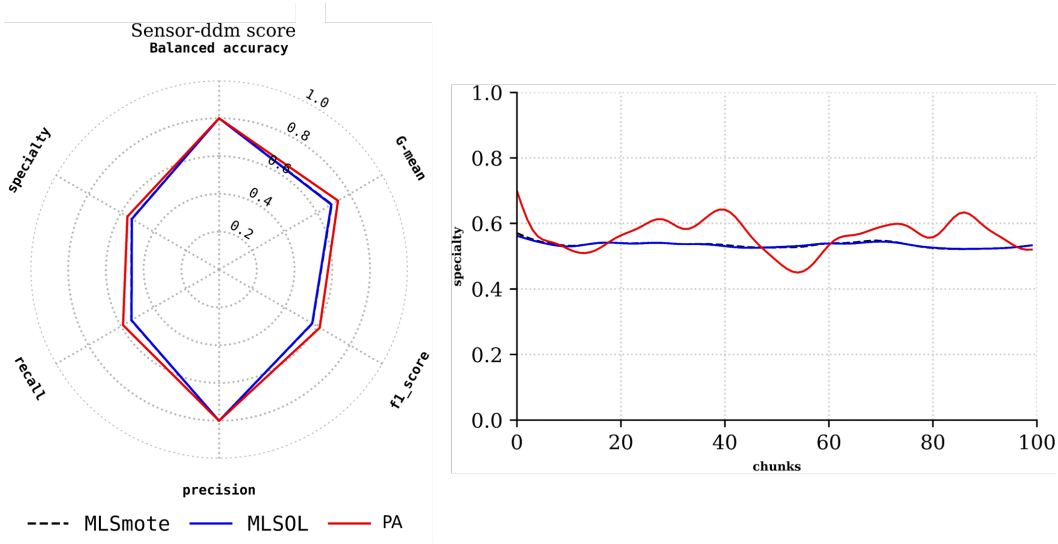

Figure 1.6: Performance of the first Proposed Approach (PA1), MLSMOTE, and MLSOL techniques on the Sensor real application stream using the DDM concept drift detector.

1.3.3.3 Results on the Synthetic Stream

The results of applying the same methods on the synthetic data stream using ADWIN as the drift detector are presented in Fig. 1.7. The radar diagram indicates metric values ranging from 0.6 to 0.8, suggesting that the synthetic stream is prone to frequent drifts. While MLSOL exhibited the highest precision, MLSOTE exhibited the lowest values. Conversely, PA1 performed well in other metrics, and MLSMOTE demonstrated the least favorable values. Upon examining the line diagram in Fig. 1.7, it becomes evident that during the initial ten chunks, the PA1's performance might be suboptimal because of the limited number of classifiers in the pool. However, after the first ten chunks, the performance improved significantly with the inclusion of more classifiers. In the same diagram, PA1 consistently achieves the highest performance across all chunks, whereas MLSOL maintains satisfactory performance, and MLSMOTE exhibits lower performance throughout. In Fig. 1.8, using the same synthetic data stream but with the DDM as the drift detector, the radar plot metrics produce similar results to the previous experiment. However, the line diagram emphasizes the same outcomes as those in the previous experiment. Nevertheless, MLSOL and MLSMOTE achieved lower values than

the previous experiment. These findings confirmed ADWIN's superiority of ADWIN over DDM when applied to synthetic data streams. These results highlight the versatility and robustness of the algorithm, demonstrating its effectiveness in handling concept drift across diverse datasets, including the challenging synthetic dataset, Covertype dataset, and Sensor dataset, regardless of whether ADWIN or DDM is used as the concept drift detector.

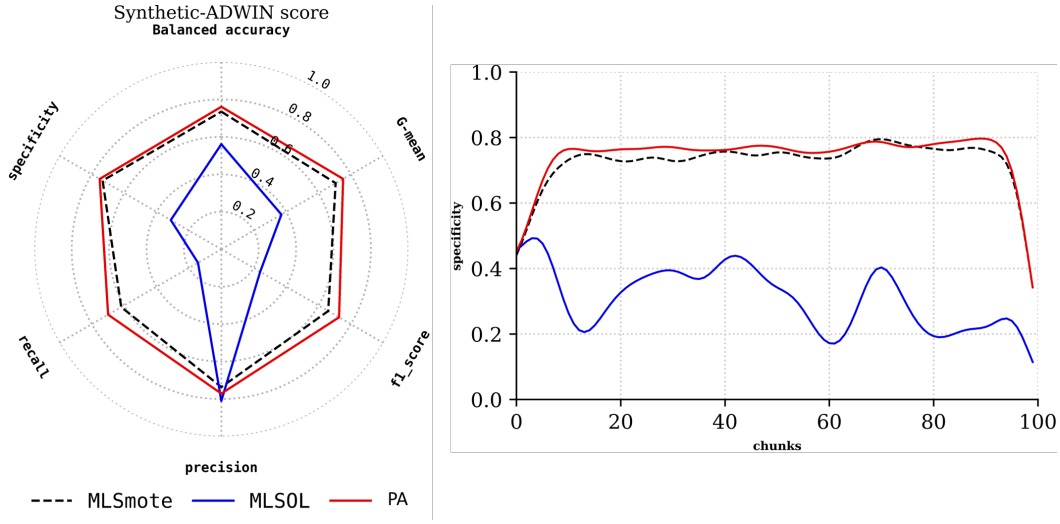

Figure 1.7: Performance of the firstProposed Approach (PA1), MLSMOTE, and MLSOL techniques on the Synthetic stream using the ADWIN concept drift detector.

1.3.4 Analysis of Class Overlap Factor between Proposed Approach (PA1), MLSMOTE, and MLSOL Techniques.

In our experimental study, we aimed to identify the critical factors that influence the selection of an optimal approach for minority classes in imbalanced drifted streams. These factors include various aspects, such as dataset characteristics, the choice of concept drift detectors, and the effectiveness of the algorithm in handling class overlap. To compare the overlapping class behavior of first proposed approach (PA1) with MLSMOTE and MLSOL, we systematically designed experiments organized into groups of ten chunks. The results were visualized in bar diagrams, contrasting PA1 with other methods (MLSMOTE and MLSOL). Figures 1.91.101.11 present ten groups, each with three bars representing MLSOTE,

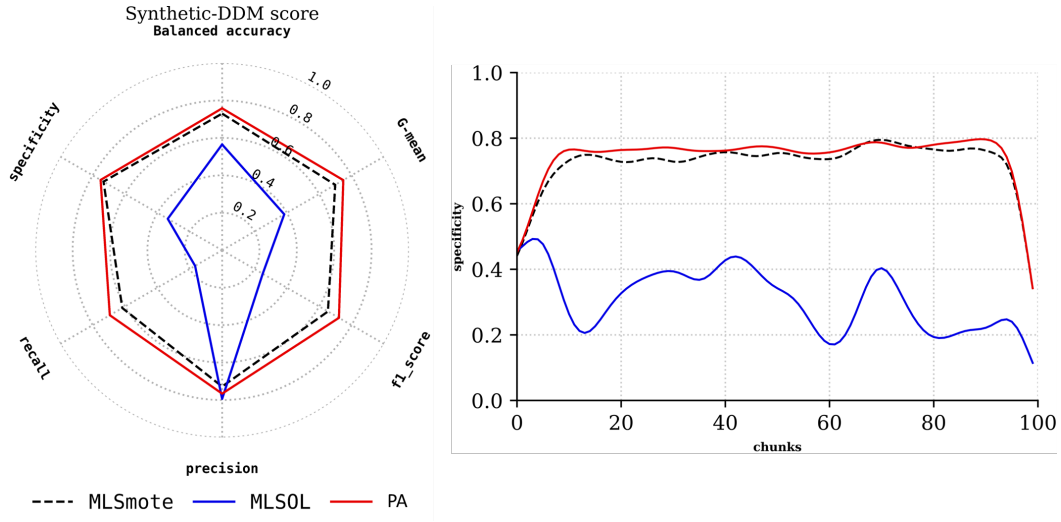

Figure 1.8: Performance of the first Proposed Approach (PA1), MLSMOTE, and MLSOL techniques on the Synthetic stream using the DDM concept drift detector.

MLSOL, and PA1, respectively. Each bar visually represents overlapped samples for ten chunks of several data streams, considering distinct concept drift detectors, specifically ADWIN and DDM. Fig. 1.9 shows two diagrams for the ADWIN and DDM detectors. In the ADWIN diagram, the third bar group (chunks 20-30) lacks overlapping samples because this chunk range does not have drifts, and no synthetic samples are generated for the training step. The sixth and tenth groups have the highest overlapped samples because these groups experience many drifts, leading to the generation of several samples and, consequently, the data indicates that MLSMOTE displays the highest number of overlapping samples across all groups, while PA1 exhibits very few, except for the last group (90-100), which has a small number of overlapping samples. However, the DDM diagram has more overlapping samples than the ADWIN diagram because the DDM detector detects fewer drifts than ADWIN. Specifically, the last value on the Y-axis of the ADWIN diagram is 3,000 samples, while the last value on the Y-axis of the DDM diagram is 2,000 samples. Fig. 1.10 and Fig. 1.11 display the overlapping samples of the three methods on the Sensor and synthetic data streams, respectively. In Fig. 1.10, the ADWIN diagram demonstrates fewer overlapped samples than in Fig. 1.10, indicating a lower number of drifts in the Sensor stream. The overall diagram indicates that first proposed approach achieves fewer overlapped samples than MLSOL and

MLSMOTE, with MLSMOTE having the highest number of overlapped samples. In the DDM diagram of Fig. 10, the number of overlapped samples is lower than in Fig. 1.9, with first proposed approach consistently achieving the lowest number of overlapped samples across most group bars. In Fig. 1.11, the ADWIN and DDM diagrams exhibit the greatest number of overlapped samples compared to the earlier figures. This suggests that the synthetic stream underwent frequent drifts and was more susceptible to noise. Consequently, the last value on the Y-axis in both the ADWIN and DDM diagrams was recorded for 7,000 samples. First proposed approach consistently achieves the lowest number of overlapped samples, whereas MLSMOTE consistently attains the highest number of overlapped samples across most group bars in both the ADWIN and DDM diagrams. In conclusion, our thorough examination of overlapping class instances across MLSMOTE, MLSOL, and first approach consistently reveals minimal overlapped samples compared to other methods, regardless of whether DDM or ADWIN is employed as drift detectors

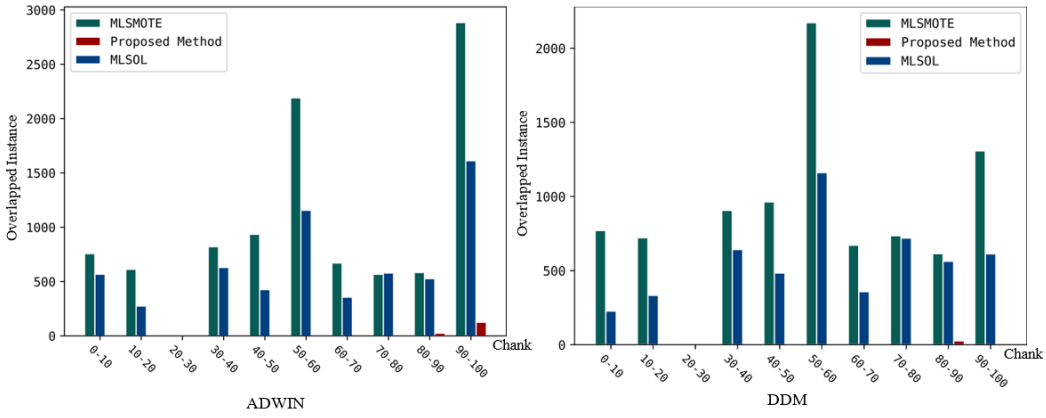

Figure 1.9: Overlapping generated points of the first Proposed Approach (PA1), MLSMOTE, and MLSOL techniques on the Covertype benchmark stream.

1.3.5 Analyzing Runtime Factor Between the First Proposed Approach, MLSMOTE, and MLSOL Techniques

The results of our experiments indicate that the choice of the best algorithm for multiclass imbalanced streams depends on various factors, including dataset characteristics, the concept drift detector used, the presence of an overlapping class

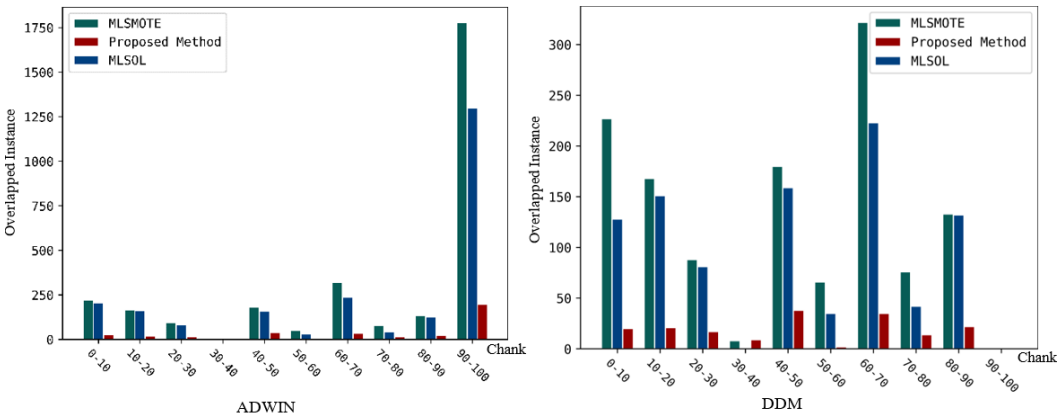

Figure 1.10: Overlapping generated points of the first Proposed Approach (PA1), MLSMOTE, and MLSOL techniques on the Sensor real application stream.

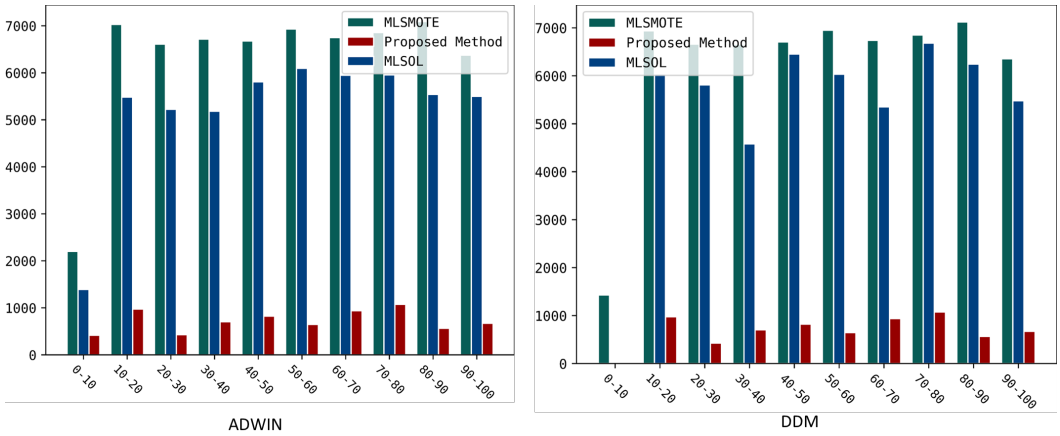

Figure 1.11: Overlapping generated instances of the first Proposed Approach (PA1), MLSMOTE, and MLSOL techniques on the synthetic stream.

problem, and the algorithm's runtime demands. To investigate the runtimes of MLSMOTE, MLSOL, and first approach, we conducted experiments, as shown in Table 1.2. Our findings reveal that first proposed approach is highly efficient, regardless of whether ADWIN or DDM is used as the concept drift detector. Specifically, when ADWIN was used, the first proposed approach algorithm took 8344 s to train and predict the Covertype stream for ensemble classifiers, whereas it took 8019 s when using the DDM detector. In contrast, MLSMOTE and MLSOL require more time for the same task. Notably, first approach maintains efficiency,

even when using the DDM concept detector. Additionally, first approach demonstrates shorter processing times in the Sensor stream and synthetic data compared with other methods. Consistently, the DDM detector achieved less time across all experiments owing to its lower detection of drifts compared to the ADWIN detector. This is because there are fewer instances that trigger training for pool classifiers when using the DDM. The bold highlighting in Table 1.2 further emphasizes the efficiency of first approach with both the ADWIN and DDM detectors across all dataset streams. Because our proposal generates fewer overlapped samples, leading to an overall decrease in the running time.

Table 1.2: Runtime of MLSMOTE, MLSOL, and the first Proposed Approach (PA1).

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**

1.3.6 Analyzing Non-parametric Tests between the First Proposed Approach, MLSMOTE, and MLSOL Techniques

We conducted a thorough series of statistical analyses encompassing 12 comparisons across three diverse datasets, three methods, and two drift detectors. These analyses were rigorously evaluated using a non-parametric test, specifically the Kruskal-Wallis test. The results of this test were striking, revealing substantial variations in the G-mean measurements across most experiments. Importantly, these differences were not due to random chance [**?**]. Upon closer examination of the assessments for the three methods, PA1, MLSMOTE, and MLSOTE, as detailed in Table 1.3, we found compelling evidence supporting the acceptance of the null hypothesis (H0). This implies that the expected and observed data exhibited statistically significant disparities. H0 is rejected when significant differences are not

observed and H0 is accepted if the P-value falls below the critical value. Underlining the significance of our analysis, the Kruskal-Wallis test was conducted with a 95% confidence level, and the P-value was rounded to the first three digits after the decimal point. Nevertheless, it is important to note that a similarity in the performances of the methods emerged in the third and fourth experiments, resulting in the rejection of H0. This similarity can be attributed to the fact that the DDM detected fewer drifts in these experiments.

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

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 ability

to effectively recognize and adapt to these novel classes in real-time is crucial for maintaining accurate and up-to-date models. Furthermore, the inherent limitations of data streams, such as limited memory and storage constraints, impose additional complexities in the learning process. Learning algorithms must operate efficiently within these resource constraints to ensure real-time processing and to avoid overwhelming computational overhead. To address the challenges presented by data streams containing emerging new classes, dynamic ensemble selection (DES) [**? ? ?**]. Dynamic ensemble selection involves utilizing multiple classifiers in machine learning to make collective predictions or classifications of data. Dynamic ensemble selection is distinguished by its ability to dynamically adapt the ensemble based on the characteristics of the data. Instead of relying on a fixed ensemble of classifiers, dynamic ensemble selection continuously assesses the performance of individual classifiers and selects the subset that demonstrates the highest competence for the current data. This adaptability enables the ensemble to enhance its performance over time by incorporating the most suitable classifiers for prevailing data conditions. Furthermore, if a classifier becomes ineffective owing to concept drift or the emergence of new classes, it can be excluded from the ensemble to prevent it from negatively affecting the overall performance. Adaptive Windowing (ADWIN) is another widely employed method to address concept drift. Concept drift refers to the phenomenon in which the statistical properties of data change over time, leading to a decline in the learning algorithm performance [**? ? ?**]. ADWIN continuously monitors incoming data and detects changes or drifts in data distribution. It achieves this by maintaining sliding windows of variable sizes and monitoring statistical measures such as the mean or variance within these windows. Upon detecting a significant change or drift, the ADWIN triggers an update in the ensemble or classifier configuration. This update may involve retraining the classifiers with new data or incorporating new classifiers that are better suited to the updated data distribution. By adapting the ensemble to changing data conditions, ADWIN ensures that the classifier system remains accurate and up-to-date even in the presence of concept drift or the emergence of new classes. The primary objective of utilizing these approaches, such as dynamic ensemble selection and ADWIN, is to establish a flexible and effective classification system that is capable

of handling data streams containing emerging new classes. By dynamically adjusting the ensemble based on the data characteristics and detecting and adapting to concept drift, these approaches enable the system to maintain accurate predictions over time. Adaptability and responsiveness are crucial for addressing the unique challenges posed by the dynamic nature of data streams and the emergence of new classes within them.

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 2.3 outlines the experimental setup and presents the results, providing details regarding the employed datasets, evaluation metrics, and procedures. Finally, in Section **??**, we offer concluding remarks, Highlighting the main discoveries, examining their significance, and suggesting possible directions for future investigations.

2.1 Motivations and Contributions

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. We aim to develop novel algorithms that can effectively handle the emergence of new classes issue in dynamic data streams, achieve high accuracy of classification, and reduce the complexity of computations. 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

In this section, we outline the key stages of our research, which consist of three distinct steps designed to address the challenges of handling emerging new classes in changeable 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, our 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 accuracy and robustness.

2.2.1 Second Proposed Approach Flow

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.

As depicted in Fig. 2.1, 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. 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).

2.2.2 Emerging New Classes Phase Details

In this section we present the Emerging new classes phase details. As shown in Fig.2, 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.

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. Fig. 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).

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

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

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

2.3 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

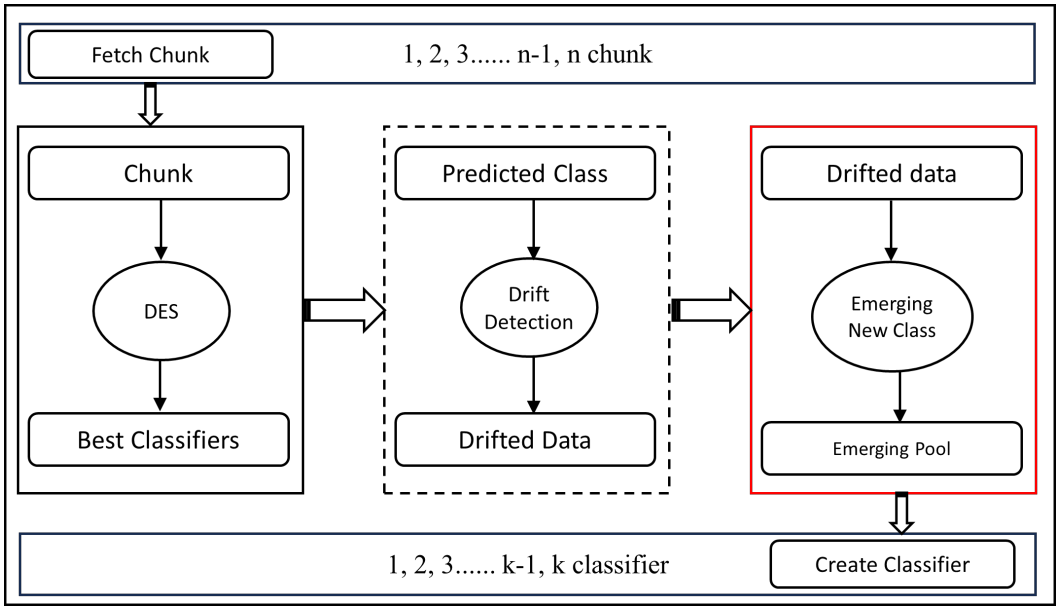

Figure 2.1: Flow of the second proposed approach for emerging drifted data.

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 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 accuracy and robustness. The experiments detailed in this study offer crucial insights into improving the second 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. The two main questions to be answered are:

- Q_1.How does the emergence of new classes in data streams affect the stability

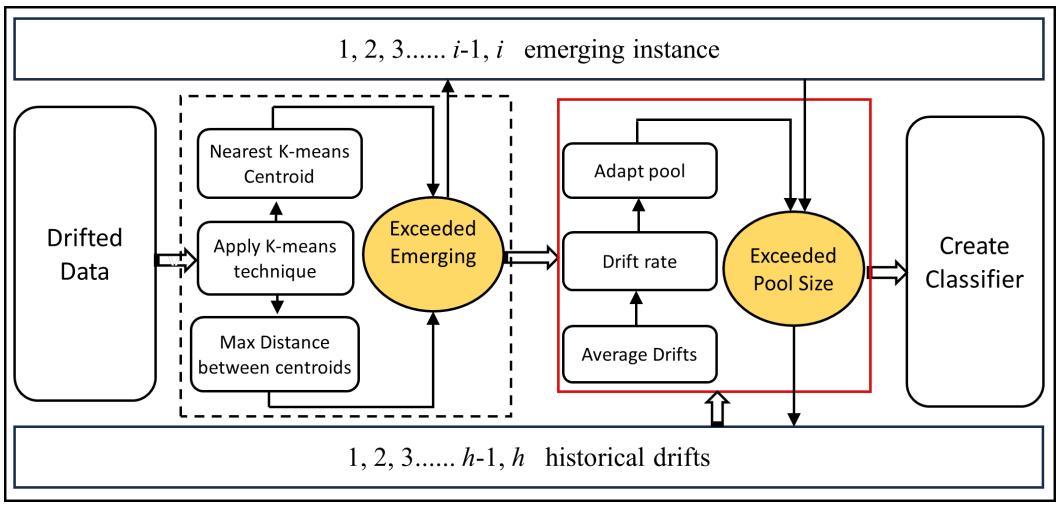

Figure 2.2: Flow of the Emerging Phase.

and performance of ML models?

- Q_2. how to employee concept drift to solve the emerging new classes problem?

2.3.1 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, F1 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 our 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

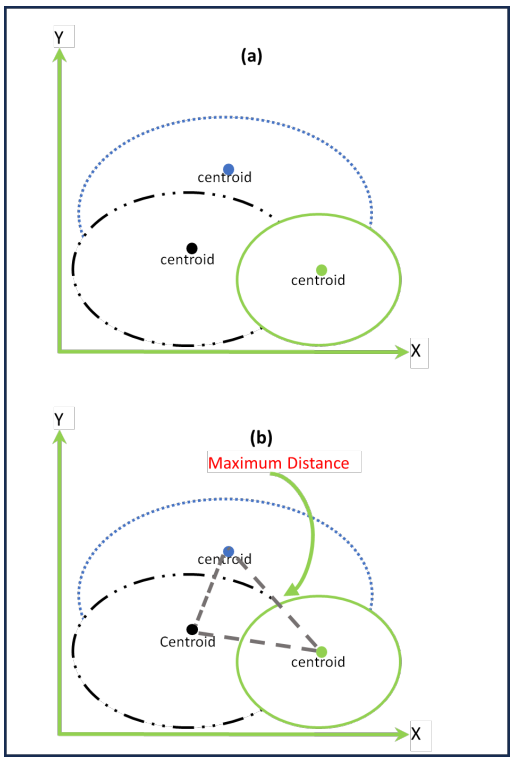

Figure 2.3: Example steps for a dataset with two features (x, y) and three classes (Blue, Black, Green): (a) Use the K-means algorithm to cluster the current chunk. (b) Determine the maximum distance between class centroids.

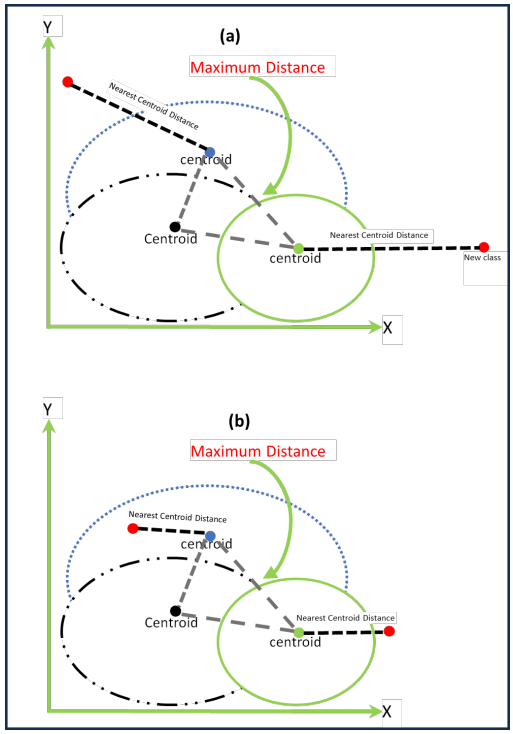

Figure 2.4: Example steps for new points (red points) with two features (x, y): (a) Identify new incoming instances (represented by red points) as potential new classes. (b) Determine whether the new instance is part of a nearby existing class or belongs to an emerging new class.

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 [**? ? ? ?**]. 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 [2].

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

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

2.3.2 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.

2.3.3 Compared Approaches

In the following section, we present a comparative analysis between our proposed GNB methodology and three established benchmark techniques, 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.

2.3.4 Examination of Experimental Findings

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

2.3.4.1 Results On The Benchmark Dataset

This experiment aimed to evaluate the performance of the second approach on the Covertype data stream using different buffer sizes with ADWIN as the concept drift detector. The findings are visually represented using scatter plots in Fig. 2.5, which contains four subplots for emerging buffer sizes of 5, 10, 20, and adaptive sizes, respectively. In Fig. 2.5(b), the G-mean accuracy of SENCForest, SENNE, KENNES, and PA2 is shown across 200 chunks, specifically focusing on a buffer size of 5 instances for emerging new classes. At first, all methods exhibit less-than-ideal accuracy in the initial 20 chunks. However, a marked improvement is observed after this period, as the classifier pool grows. This growth bolsters the

DES, dynamic ensemble selection, technique, allowing it to choose the most suitable classifiers for subsequent chunks. Consequently, there are significant gains in accuracy from chunk 20 onward, PA2 consistently achieved the highest accuracy, while KENNES and SENNE recorded the lowest. Additional experiments with buffer sizes of 10 and 20, shown in Fig. 2.5(b) and Fig. 2.5(c), revealed slightly lower accuracy compared to the buffer size of 5, likely due to fewer updates when using larger buffer sizes. Finally, the adaptive buffer size experiment in Fig. 2.5(d) highlights the PA2 algorithm's superior performance across all chunks, with the adaptive emerging pool size delivering the best results.

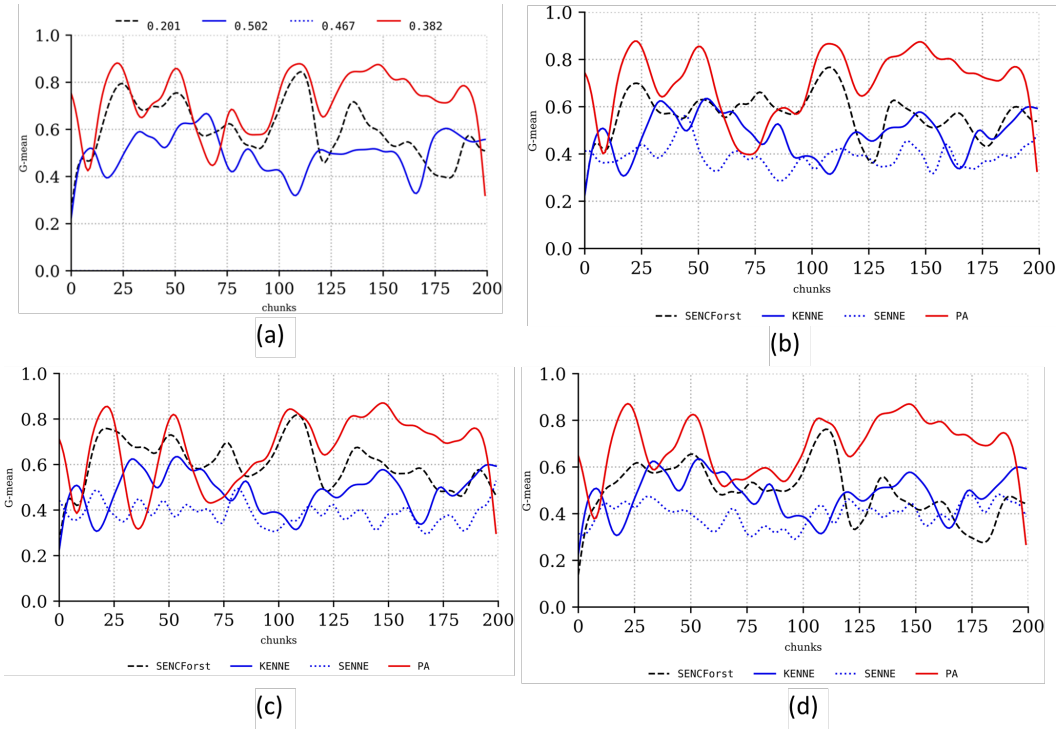

Figure 2.5: Covertype stream for various emerging buffer sizes: (a) buffer size of 5, (b) buffer size of 10, (c) buffer size of 20, and (d) adaptive buffer size.

2.3.4.2 Results On Real Application Stream

This experiment focused on assessing the accuracy of SENCForest, SENNE, KENNES, and PA2 across 200 chunks of a sensor data stream, utilizing varying buffer sizes similar to the previous experiment. The results, shown in Fig. 2.6, were visualized

using scatter plots. Fig. 2.6 (a) displays the classification accuracy for each algorithm with a buffer size of 5. The PA2 algorithm achieved the highest performance overall, except for chunks 43 to 48, where some instances were incorrectly assumed as outliers rather than emergent. In contrast, SENCForest and SENNE had the lowest accuracy. Additional experiments with buffer sizes of 10 and 20 (Fig. 2.6(b) and Fig. 2.6(c)) demonstrated reduced accuracy compared to the buffer size of 5, attributed to fewer updates with larger buffer sizes. The adaptive buffer size experiment in Fig. 2.6(d) reaffirmed the effectiveness of the PA2 algorithm, which outperformed others across all chunks, with the adaptive emerging pool size providing optimal results.

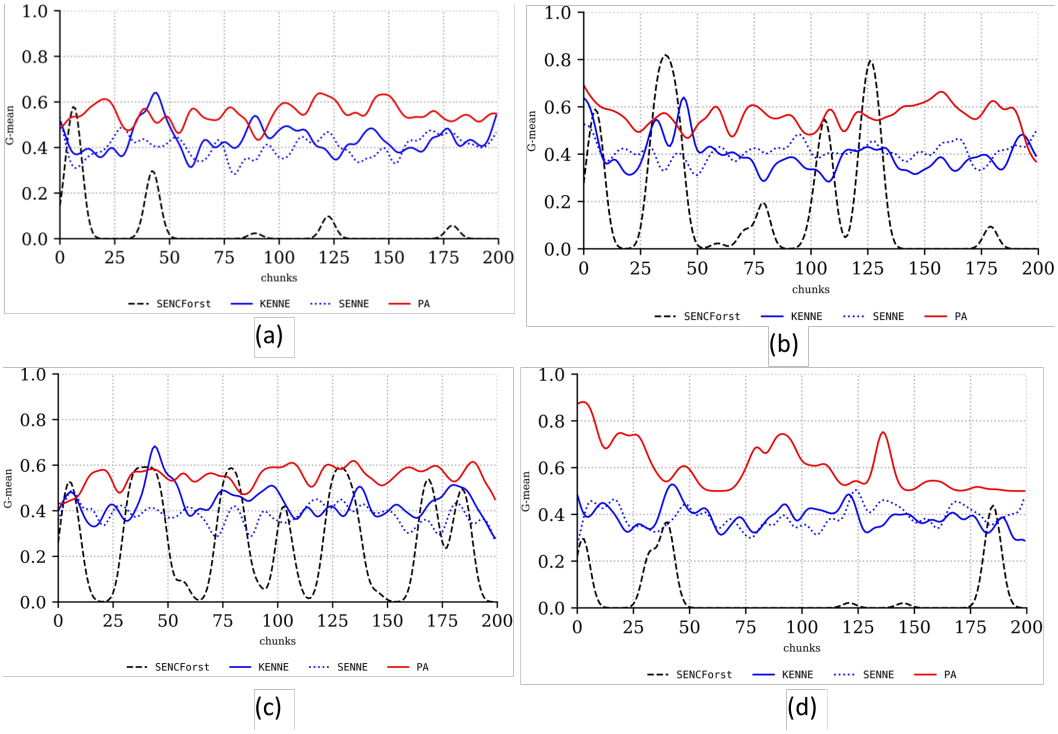

Figure 2.6: Sensor stream for various emerging buffer sizes: (a) buffer size of 5, (b) buffer size of 10, (c) buffer size of 20, and (d) adaptive buffer size.

2.3.4.3 Results On Synthetic Stream

This experiment aimed to assess the performance of the previously tested methods on a synthetic data stream. Results, depicted in Fig. 2.7, were analyzed using radar

and scatter plots. Fig. 2.7(a) shows the performance of various algorithms using a buffer size of 5, with the radar plot highlighting PA2's superior performance. Scatter plots illustrate that PA2 consistently delivered the highest accuracy when applied to the synthetic stream with buffer sizes of 5, 10, and 20, as shown in Fig. 2.7(a), Fig. 2.7(b), and Fig. 2.7(c), respectively. Fig. 2.7(d) demonstrates that the adaptive pool size yielded the best accuracy on the synthetic stream, underscoring the adaptability and efficiency of the PA2 algorithm.

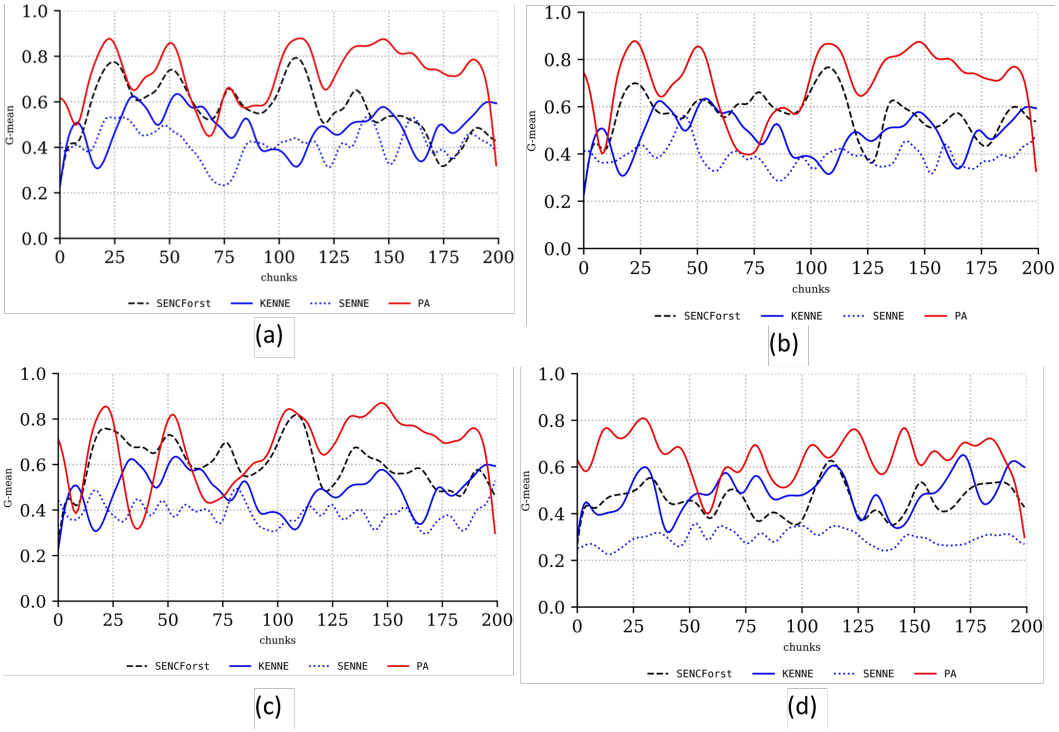

Figure 2.7: Synthetic stream for various emerging buffer sizes: (a) buffer size of 5, (b) buffer size of 10, (c) buffer size of 20, and (d) adaptive buffer size.

2.3.5 Runtime Analysis Of The Best Algorithms

Our experimental results revealed that selecting the optimal algorithm for detecting emerging new classes is influenced by various factors, including dataset characteristics, buffer length, and the algorithm's runtime requirements. We conducted a series of experiments to evaluate the runtime performance of different algorithms and

observed that the PA2 algorithm demonstrated outstanding efficiency. Specifically, with the adaptive pool size approach, the PA2 algorithm trained bagging classifiers 150 times within 2131 seconds when using 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 Table 2.2.

2.3.6 Comparison between GNB, KNN, and HT, SVC

In this section, we compare various algorithms (KNN, SVC, GNB, and HT) as base classifiers for the bagging technique. The comparison, illustrated in Fig.2.8, focuses on the Covertype dataset stream, where GNB (blue line) and HT (red line) consistently achieve the highest scores across most chunks. Both classifiers also excel in radar plots across key performance metrics, including accuracy, precision, recall, and F1-score. Based on these results, GNB and HT emerge as the most suitable base classifiers for the bagging technique. We further compared GNB and HT in terms of runtime and update frequency, as shown in Table 2.3. The results indicate that GNB has the best runtime performance, while HT demonstrates superior accuracy, likely due to its higher number of updates compared to GNB, as reflected in Table 2.3 and Fig. 2.8.

2.3.7 Comparison between ADWIN and DDM

In this section, we compare two concept drift detection methods: the Drift Detection Method (DDM) [**?**] and the Adaptive Window (ADWIN) [**?** **?**]. These methods are widely recognized for their excellent performance in handling incremental drifted streams [**?** **?** **?** **?**]. As shown in Fig. 2.9, 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 accuracy.

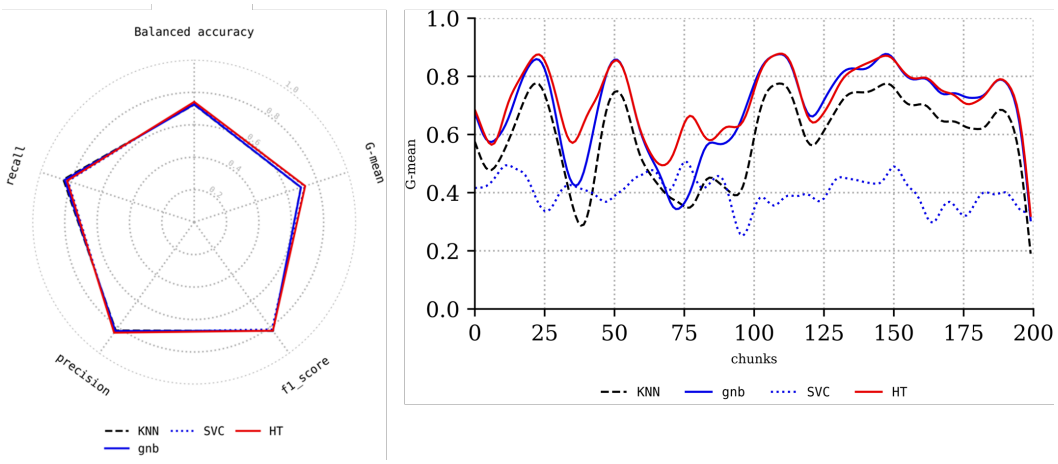

Figure 2.8: Covertype stream for concept drift detectors (ADWIN, DDM) evaluated using several metrics: recall, precision, balanced accuracy, G-mean, and F1 score.

Furthermore, when comparing their runtime and update frequency, as presented in Table 2.4, ADWIN proves to be the superior detector for drifted streams. It identifies the highest number of drifts (139) in the least amount of time (272 seconds), demonstrating its efficiency and effectiveness.

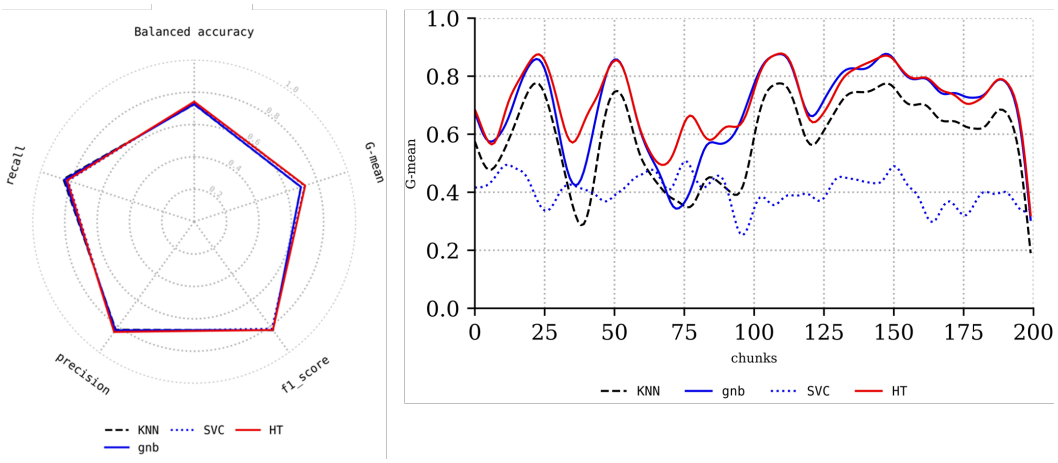

Figure 2.9: Synthetic stream for concept drift detectors (ADWIN, DDM) evaluated using several metrics: recall, precision, balanced accuracy, mean, and F1 score.

Algorithm 3: Second proposed approach algorithm for emerging drifted data streams.

Input: data stream, classifier threshold κ
Data: current chunk a, classifier k, classifiers pool Ψ, drifted pool ψ
Output: prediction p
$\psi \leftarrow \phi$
$\Psi \leftarrow \phi$
for *stream has 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**
      $k \leftarrow$
       utilize a and ψ to get new classes according to Algorithm 4
      $\Psi \leftarrow \Psi + k$
      if $|\Psi| > \kappa$ **then**
        removeWorstClassifier(a)
      end
      $p \leftarrow$ getPrediction(a, k)
    end
  end
end
return p

Algorithm 4: Flow of 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

Table 2.1: The datasets utilized in the experiments exhibited diverse characteristics.

Data stream	Features	Classes	Instances	Chunk Size
Sensor stream [1]	5	58	392600	2000
Covertype stream[2]	52	7	581010	2000
Synthetic stream	10	4	200000	2000

Table 2.2: Running time of SENCForest, SENNE, KENNE, and the second Proposed Approach (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**

Table 2.3: Running time of KNN, SVC, GNB, and HT as base classifiers.

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

Table 2.4: Running time of ADWIN and DDM as drift detectors.

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

3

Dynamic Classification Ensembles for Handling Imbalanced Multiclass Drifted Data Streams

Transfer learning plays a pivotal role in addressing the intricate challenges posed by dynamic data streams and inherent concept drifts. Transfer learning aims to enhance a model's learning performance within a target domain by leveraging the knowledge gleaned from the source domains [**?** **?**]. This research domain focuses extensively on addressing target tasks that grapple with limited or unlabeled data, with a dual emphasis on amplifying positive knowledge transfer and alleviating negative knowledge transfer [**?**]. To facilitate the transfer of valuable knowledge, researchers have developed diverse techniques, including methodologies aimed at reducing the domain gap, such as instance re-weighting [**?** **?** **?**] and feature matching [**?** **?**]. Conversely, strategies for mitigating negative knowledge transfer often involve down weighting irrelevant data sources [**?**]. Although a substan-

tial body of transfer learning research has focused on static environments, where data in each domain are assumed to conform to the same distribution, real-world scenarios, including financial data analysis, energy demand prediction, and climate data analysis, often involve dynamic environments. Within these dynamic contexts, the concept drift problem [**?** **?**] emerges as data distributions evolve over time. Dynamic environment transfer learning requires continuous adaptation to concept drift and adaptive utilization of valuable knowledge from source domains within each distinct environment. These new challenges in transfer learning, particularly within dynamic environments, represent a profound departure from traditional transfer learning paradigms that do not inherently address concept drift and the dynamic nature of evolving data. To overcome these challenges, Dynamic Ensemble Selection (DES) [**?** **?** **?**], ADWIN, and Streaming Ensemble Algorithm (SEA) have become prominent in the scientific literature [**?** **?** **?**] Dynamic ensemble selection employs multiple classifiers within the realm of machine learning to make collective predictions or classifications, and leverage data characteristics. Its key feature is its dynamic adaptability, as it continuously evaluates the performance of individual classifiers and selects a subset that demonstrates the highest competence within the prevailing data conditions. This adaptability empowers the ensemble to progressively enhance its performance by incorporating best-fitting classifiers for the existing data context. Moreover, ineffective classifiers can be excluded from the ensemble in scenarios marked by concept drift, thereby protecting the overall performance from detrimental effects. Adaptive Windowing (ADWIN) is another widely used method designed to address the challenges of concept drift [**?**]. Concept drift refers to the phenomenon in which the statistical properties of data change over time, leading to a decline in the learning algorithm performance. ADWIN monitors incoming data by maintaining sliding windows of variable sizes and evaluating statistical attributes, such as the mean or variance within these windows. Upon detecting substantial shifts or drifts, the ADWIN initiates adjustments to the ensemble or classifier configuration. These adjustments may involve retraining classifiers with fresh data, or incorporating new classifiers that are better suited to the updated data distribution. Additionally, the Streaming Ensemble Algorithm (SEA) is an integral component for addressing these dynamic challenges [**?** **?** **?**]. SEA is specifically designed to manage data streams and inherent concept drifts.

This is achieved by adapting the ensemble of classifiers in real time as new data arrives. SEA's adaptability of SEA ensures that the ensemble remains robust and accurate even in the face of shifting data conditions and the emergence of new classes. This study proposes efficient algorithms for classifying data streams in real-time scenarios, focusing on addressing the challenges posed by heterogeneous transfer learning in data distributions. The goal is to develop a novel algorithm (Heterogeneous Transfer Learning) that can effectively handle non-stationary data streams, achieve high classification accuracy, and minimize 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 3.3 outlines the experimental setup and presents the results, providing details regarding the employed datasets, evaluation metrics, and procedures.

3.1 Motivations and Contributions

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

1. The primary innovation involves incorporating a concept drift detection method in conjunction with an ensemble classifier, enabling real-time adaptation and refinement of the third proposed approach in response to transfer learning in non-stationary environments. This methodology ensures continuous evolution of the classification model in accordance with the changing data landscape.
2. The second significant advancement is the introduction of a precise weighting method to assess the significance of each local classifier within the ultimate classifier.
3. The third significant contribution is the development of an innovative approach that employs the eigenvector technique to facilitate the transfer of knowledge from heterogeneous source domains to the target domain.

3.2 Third Proposed Methodology

In this section, we introduce 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 our approach, we focus 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 accuracy 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 Overall Details

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 demonstrated in Equation 1. This equation factors in both correct and incorrect predictions for each classifier class, with K representing the number of classifiers in the current chunk, C denoting the number of classes in the current chunk, and i indicating the current chunk. The components of HTL operate synergistically, 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, our method advances 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.

- **Feature scaling** The concluding phase of our approach 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 Details

Fig. 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.

- **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.

3.2.3 HTL Detailed Algorithm

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 accuracy 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 homogeneous 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 accuracy 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 accuracy 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.

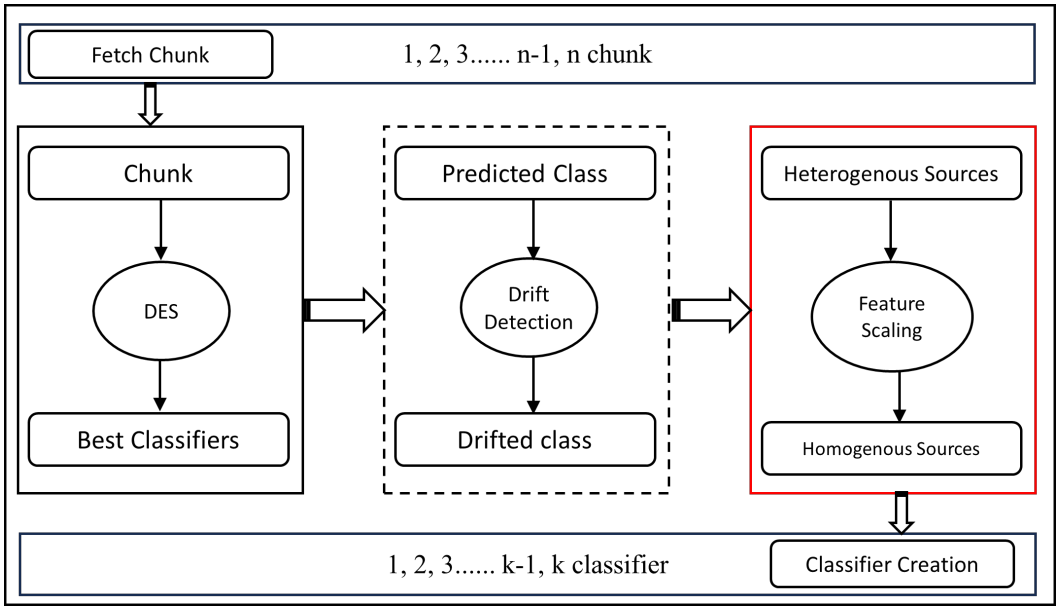

Figure 3.1: Heterogeneous Transfer Learning (HTL) Approch Flow.

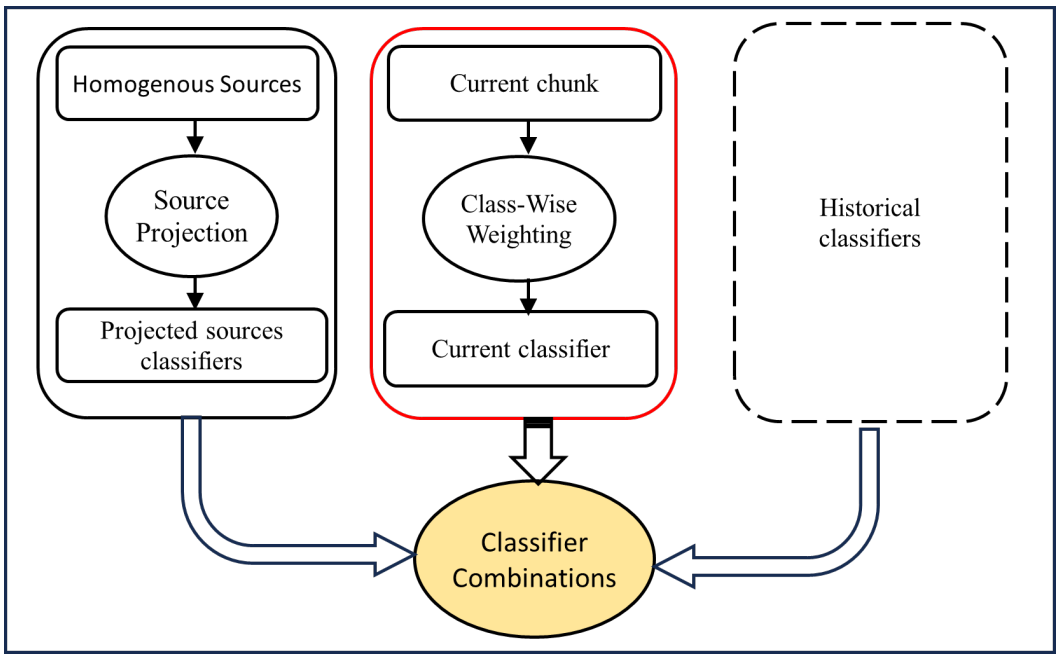

Figure 3.2: Approach for Classifier Creation.

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

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

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

Algorithm 5: Flow of the Heterogeneous Transfer Learning (HTL).

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
```

3.3 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 our 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. The two main questions to be answered are:

- Q_1. How does the Heterogeneous sources in data streams affect the stability and performance of Transfer Learning Technique?
- Q_2. What approaches or techniques can be employed in heterogeneous transfer learning to facilitate knowledge transfer across diverse sources and domains?

3.3.1 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, F1 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 our experiments, the threshold for the pool classifier in each approach was maintained at eight. Consequently, if the threshold is surpassed, the least-performing classifier is systematically eliminated. This configuration was applied consistently across all approaches to ensure

fair engagement. The experiments were carried out using the Python programming language, with the source code publicly accessible on GitHub[1] . When dealing with heterogeneous multisource streams, it is often impractical or not applicable to truly heterogeneous experiences. Consequently, the eigenvector technique was utilized in all related approaches. This decision stems from the fact that the algorithms in related work primarily focus on homogeneous source domains and do not address the challenges posed by heterogeneous multisource data. Therefore, heterogeneous experiments were conducted.

3.3.1.1 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.

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

Table 3.1: Characteristics of the datasets utilized in the experiments.

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

3.3.1.2 Compared Approaches

In the subsequent section, we introduce the established benchmark techniques 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, we compare the third proposed approach (HTL) with AW- CORAL [**?**] and HE-CDTL yang2021concept.

3.3.2 Analysis of Experimental Results

This section presents a comprehensive evaluation of the performance of the proposed approach across multiple data streams. To guarantee a thorough assessment, five performance metrics—F1 score, precision, recall, G-mean, and BAG—were presented using two visualization diagrams: radar and line. A radar chart was cleverly utilized to provide a comprehensive summary, precisely demonstrating the performance of each algorithm across the six key metrics. By calculating the average value of each metric, the overall performance of each approach (HTL, CORAL, and CDTL) was determined. On the other hand, the line diagram utilizes the G-mean metric to compare these methods for each chunk across 100 chunks. A line diagram was used in the first five experiments, whereas a combination of radar and line diagrams was employed in the last two experiments. This approach ensured a detailed and nuanced evaluation of the performance of the third proposed approach in diverse experimental scenarios.

3.3.2.1 Results on Target Domains Dataset without Source Domain

Fig. 3.3 showcases the results of applying the aforementioned methods to the Covertype dataset, where no stream is utilized as a source domain. The line diagram specifically highlights the classification accuracy measured by the G-mean metric across 100 data chunks for each method. Notably, all methods display suboptimal accuracy during the initial 10 chunks. However, a significant improvement is evident in subsequent phases. This improvement can be attributed to the expansion of the classifier pool, encompassing an increasing number of classifiers. This expansion allows the DES technique to become more adept at selecting the most

appropriate classifier for each incoming chunk. Consequently, accuracy experiences a notable upsurge in later chunks, underscoring the adaptability and efficacy of the ensemble approach. From chunk 20 to the final chunk, HTL consistently achieves the highest accuracy across most chunks. In contrast, CORAL shows lower performance in certain chunks, while CDTL displays lower performance in others. The superior performance of HTL can be credited to its utilization of the proposed weighting method, which assigns a weight to each historical classifier. This approach contributes to the effective training of the pool classifiers, thereby enhancing the overall performance of the approach.

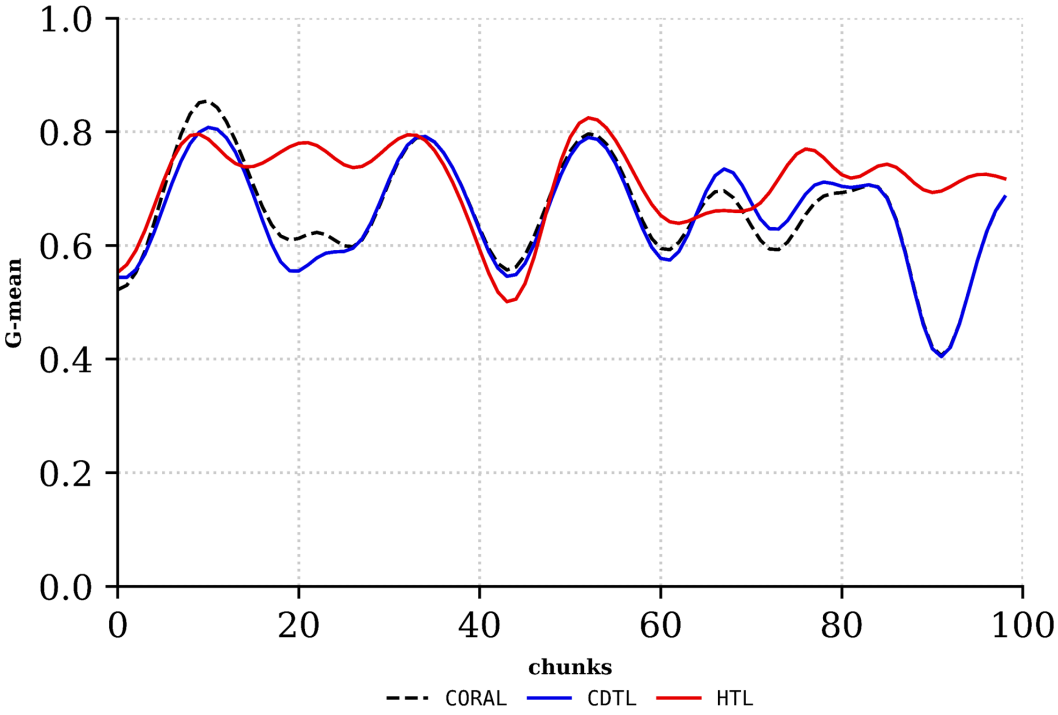

Figure 3.3: Results of the Covertype Stream as the Target Domain in the Absence of the Source Domain.

3.3.2.2 Results on the Homogeneous Source Domains Dataset

Fig. 3.4 depicts the results of applying the compared methods to the Covertype data stream as the target domain and Synthetic-52 as a homogenous source domain, which has the same dimensionality as the Covertype stream (52 features and

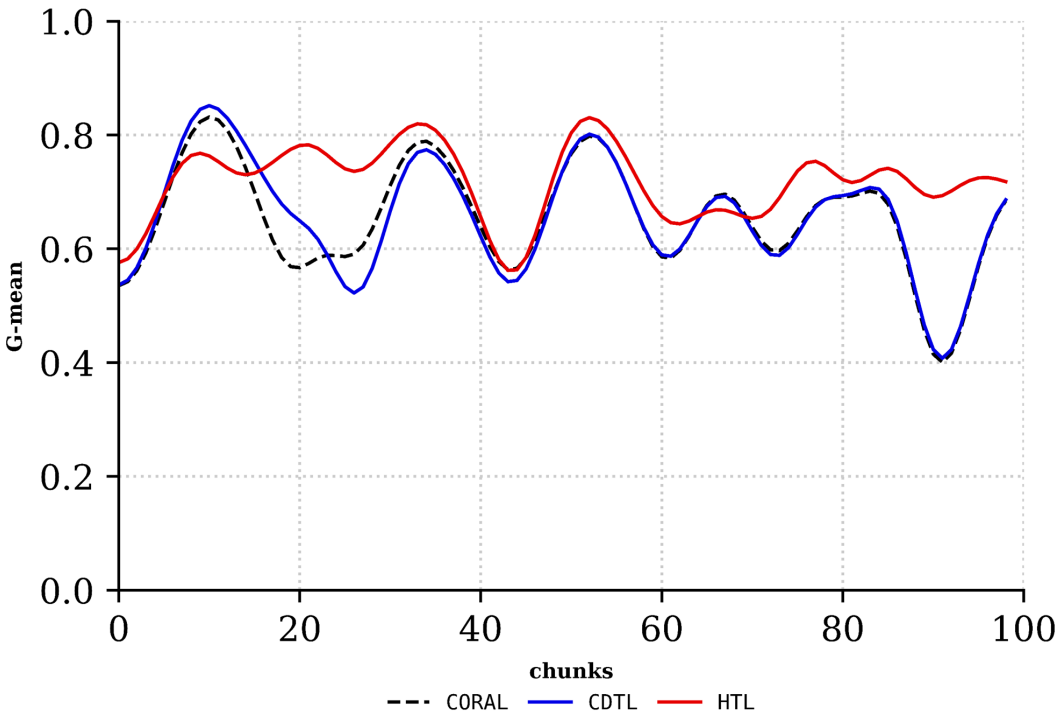

Figure 3.4: Results of the Covertype Stream as the Target Domain with Homogeneous Source Domains.

seven classes). Upon examining the line diagram, it becomes evident that HTL's performance in the first eight chunks may be suboptimal due to the insufficient number of classifiers available. However, following the initial eight chunks, the addition of more classifiers results in an improvement in HTL's performance. In contrast, CDTL maintains satisfactory performance throughout the chunks, while MLSMOTE consistently displays lower performance. It is important to note that HTL consistently achieves the highest performance across all chunks in the diagram. Notably, the CDTL and HTL methods in Fig. 3.4 demonstrated superiority over the same methods in Fig. 3.3. This is attributed to the experiment shown in Fig. 3.4, which incorporates Synthetic-52 as a source domain, thereby enhancing the performance of the methods. In contrast to the experiment in Fig. 3.3, which lacks a source domain.

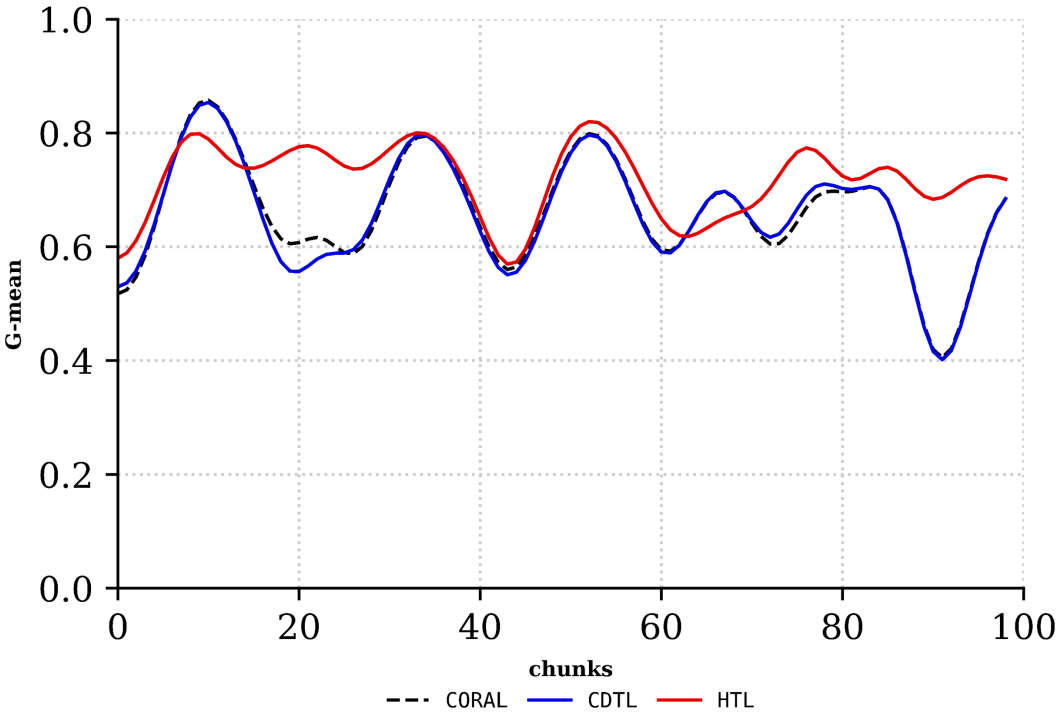

Figure 3.5: Results of the Covertype Stream as the Target Domain with One Heterogeneous Source Domain.

3.3.2.3 Results on One Heterogeneous Source Domain

In this experiment, the Synthetic-8 stream was utilized as a heterogeneous source domain. Considering the nature of the HTL algorithm, which can operate with heterogeneous sources, specific adjustments were implemented to enable CORAL and CDTL, which typically do not operate with such sources, to function in a heterogeneous domain environment. The eigenvector technique was applied to CORAL and CDTL to facilitate compatibility with the heterogeneous sources. Fig. 3.5 shows the results of applying the compared methods to the Covertype data stream as the target domain, with Synthetic-8 acting as the heterogeneous source domain. Synthetic-8 has a different dimensionality (eight features and four classes) compared with the Covertype stream. An analysis of the line diagram reveals that the performance of the compared approaches may vary across different chunks depending on the source domain and the positive knowledge it contributes. HTL consistently outperforms, achieving the highest performance across almost all chunks,

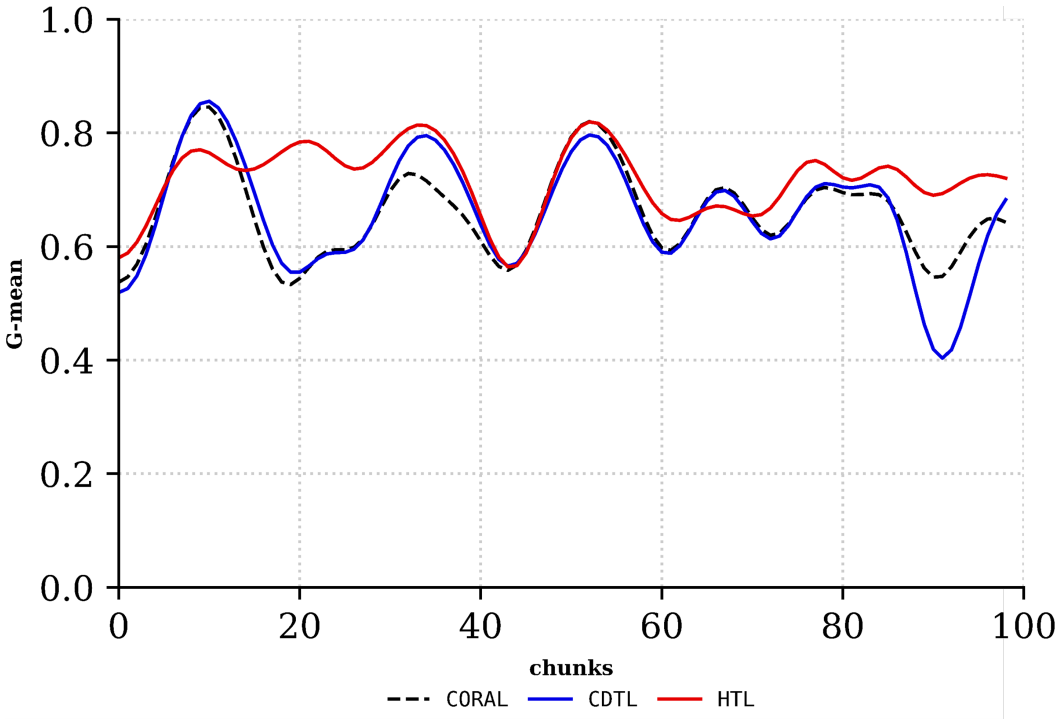

Figure 3.6: Results of the Covertype Stream as the Target Domain with Multiple Heterogeneous Source Domains.

whereas CDTL and CORAL show lower performance.

3.3.2.4 Results on All Heterogeneous Source Domains Stream and Covertype Stream as the Target Domain

Fig. 3.6 presents the results of applying the compared methods to the Covertype data stream as the target domain, with the Synthetic-8, Synthetic-52, and Sensor streams serving as heterogeneous source domains. From the analysis, the performance in this figure exceeds that in Fig. 3.6. This improvement is likely due to the larger number of source domains in this experiment, which leads to an incremental increase in positive knowledge with each added source domain, consequently improving method performance. In Fig. 3.7, the same experimental setup was employed, but the target domain was switched from the cover type to the sensor stream. Additionally, the Covertype stream was included as one of the source domains in this configuration. This modification was intended to evaluate

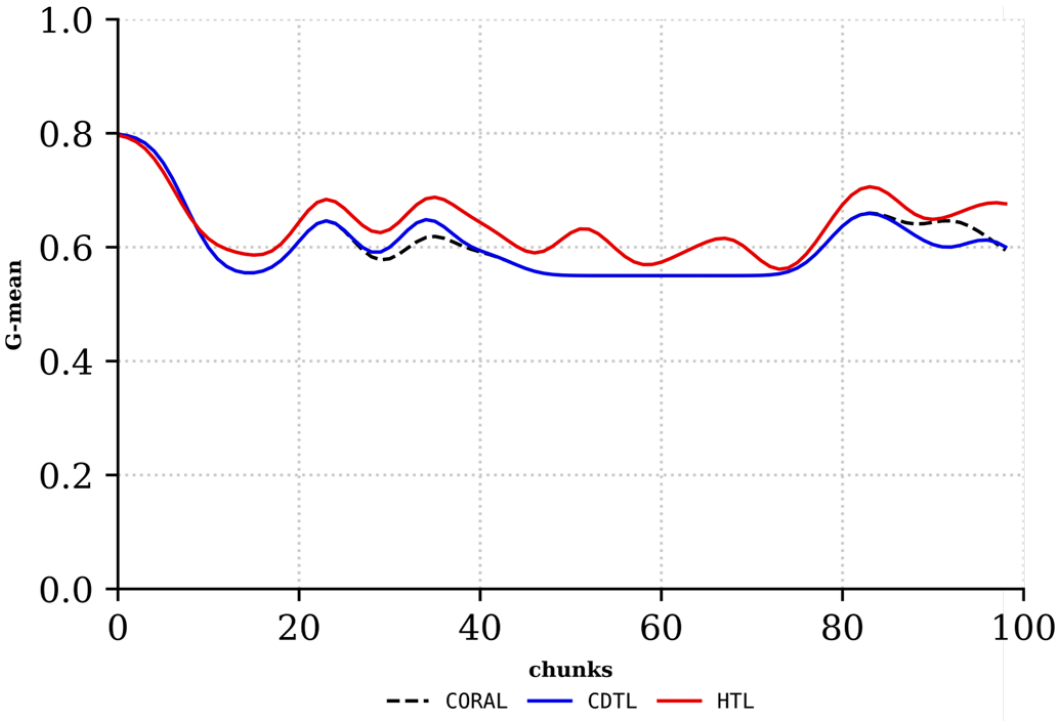

Figure 3.7: Results of the Sensor Stream as the Target Domain with Multiple Heterogeneous Source Domains.

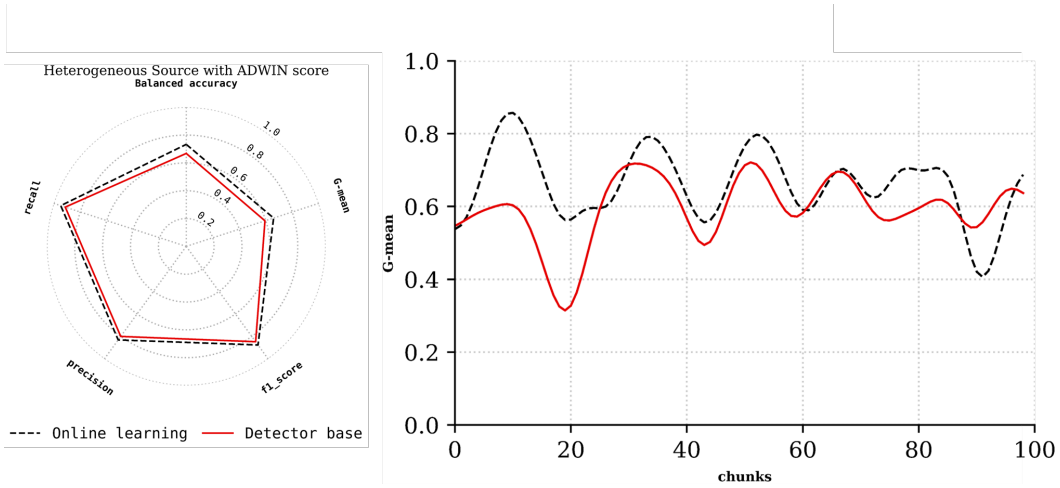

Figure 3.8: Online learning and detector chunk-based result in Covertype stream via ADWIN detector

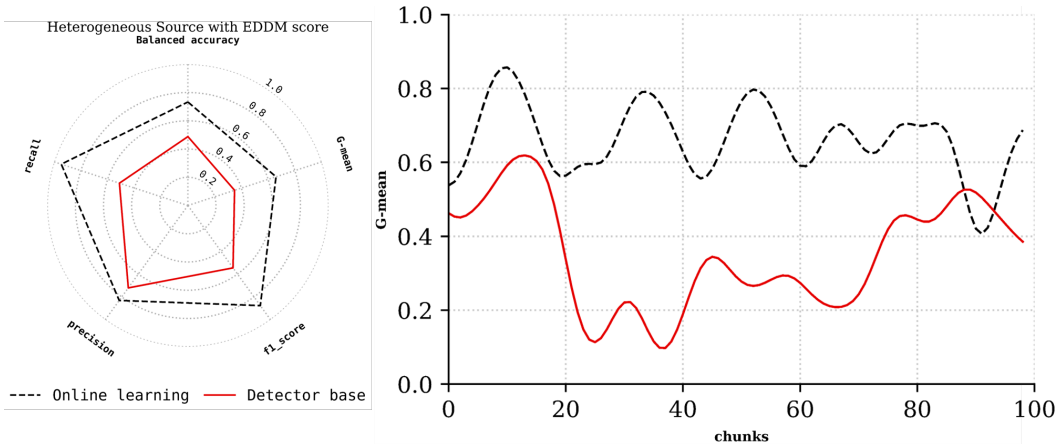

Figure 3.9: Results of Online Learning and Chunk-Based Detection in the Covertype Stream with the ADWIN Detector.

the performance of the HTL across the various target streams. Fig. 3.7 shows that the performance of all methods might be lower than that of the Covertype stream, mainly because of the increased noise in the sensor stream and numerous drifts. However, HTL consistently outperforms, achieving the highest performance across all chunks, whereas CORAL and CDTL display nearly identical values in most chunks.

3.3.2.5 Results on One Heterogeneous Source Domain

This section presents the overall performances of the compared methods (CORAL, CDTL, and HTL) across five key performance metrics: F1 score, precision, recall, G-mean, and BAG. Table 2 showcases the performance of each method across different experiments for the five metrics. The table illustrates the average value of each metric, which represents the overall performance of each method for 100 chunks. The emphasis on the bold highlighting in the table underscores the superiority of the HTL algorithm in various source domain configurations. In the first experiment (without the source domain), CORAL showed the best performance in terms of precision, while HTL demonstrated the best performance in the other metrics. Furthermore, in most experiments, HTL outperformed the other methods across most metrics, except for the recall metric in the third experiment, in which CDTL achieved the best performance. Notably, each subsequent experiment

demonstrated improved performance compared with the previous experiment. This trend arose from the incremental nature of the source domain in each experiment. The incremental addition of the source domain enhances the positive knowledge of each experiment, leading to an increased performance of the compared methods in most experiments.

Table 3.2: Comprehensive Performance of CORAL, CDTL, and HTL Across All Previous Experiments.

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

3.3.3 Analysis Runtime between CORAL, CDTL, and HTL Techniques

Our experimental results highlight that the selection of the optimal algorithm for handling heterogeneous transfer learning is influenced by various factors. These factors include the dataset characteristics and runtime requirements of the algorithm. In the experiments, the focus was on analyzing the runtimes of different algorithms, and the results were insightful. Notably, the HTL algorithm was exceptionally efficient. For example, when considering the training of the first experience (homogeneous source domain), the HTL algorithm completed 100 iterations within 304 seconds. In contrast, the CORAL and CDTL algorithms require more time to achieve the same number of training iterations. Furthermore, the HTL algorithm consistently demonstrated remarkable efficiency, particularly when the source domain settings were absent, single heterogeneous, or multisource heterogeneous. These findings highlight the superior runtime performance of the HTL algorithm, positioning it as an attractive choice for scenarios with stringent time constraints. The bold highlighting in Table 3 further accentuates the efficiency of the HTL algorithm across various source domain settings, including homogeneous, absent, single heterogeneous, and multi-source heterogeneous settings, making it a compelling option for applications where time efficiency is of paramount importance.

Table 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**

3.3.4 Analysis accuracy and runtime of HTL for online learning and chunk-based concept drift

In this experiment, we investigated the performance durations of various learning flow techniques, specifically online learning and chunk-based concept drift, aim-

ing to provide a thorough comparison between them. As shown in Table 3.4, the EDDM drift detector demonstrated significant time savings by successfully detecting 40 drifted chunks out of 100 chunks, resulting in 40 learning times. Conversely, the ADWIN drift detector achieved time savings superior to the online technique, having detected 63 drifted chunks and requiring 63 learning times. On the other hand, online learning exhibited the least efficient time savings, as it necessitated 100 times for 100 chunks. Moreover, in Figures 3.8,3.9, each figure includes two diagrams (radar and line diagram). As seen in the radar and line diagrams of Fig. 3.8, used to compare online learning and the ADWIN detector, online learning emerges as the most effective technique for achieving optimal performance, albeit with an increase in time consumption. Similarly, Fig. 3.9 is employed to compare online learning with the EDDM detector, demonstrating that online learning is the most effective method for optimal performance. Consequently, online learning stands out as the most effective approach. Hence, by examining the radar diagrams of both figures, it becomes evident that ADWIN is the most similar alternative to online learning. As a result, ADWIN proves to be a favorable compromise, striking a balance between online learning and EDDM detectors in terms of accuracy. In contrast, EDDM exhibits the lowest accuracy. Therefore, in the context of chunk-based concept drift, particularly when utilizing the ADWIN detector, this approach is ideal for environments that demand strong performance while minimizing time expenditure.

Table 3.4: Runtimes (in seconds) for the Online Learning, ADWIN, and EDMM drift detectors.

Technique	**Runtime**	**Learning Times**
Online Learning	3102	100
ADWIN detector	2257	63
EDMM detector	1948	40

4

Conclusions and Future Work

In this chapter, we presented a comprehensive methodology for incremental learning in data streams, particularly addressing the complexities of imbalanced data, including minority and overlapping classes 4.1, Emerging new classes in the incremental drifted streams 4.2 and the heterogeneous transfer learning in the drifted streams 4.2.

4.1 Imalanced multiclass stream

This study integrates a novel oversampling technique, concept drift detection, and Dynamic Ensemble Selection (DES) to identify the most suitable ensemble classifier. Extensive experimentation across various datasets, including benchmarks, real-world streams, and synthetic data, demonstrated the effectiveness of our methodology. A key feature of our approach is the rapid detection of concept drifts, which allows the system to adapt quickly by training new base classifiers, ensuring continued relevance and accuracy in real-time scenarios. We addressed the minority

class problem through advanced oversampling, while the KNN algorithm was employed to prevent the creation of overlapping class instances. The DES technique played a crucial role in selecting the most effective classifiers, optimizing overall performance.

The results of our evaluation, based on multiple performance metrics, show that our methodology is particularly effective in handling multiclass imbalanced stream challenges. It excels in maintaining high accuracy as class distributions evolve. Beyond accuracy, the approach is marked by its adaptability, efficiency, and scalability. The capability for dynamic model updates driven by incoming data allows for continuous adaptation to changing data distributions, ensuring the model's reliability and relevance in real-time applications.

However, our methodology does have certain limitations. The time required to generate non-overlapping synthetic instances is significant, and the approach's success heavily depends on the accuracy of MLSMOTE and MLSOL techniques. Future research should focus on developing more advanced oversampling techniques that avoid overlapping synthetic instances. Additionally, exploring meta-learning methods to better assess imbalanced multiclass ratios and minority class identification could further improve the approach.

4.2 Emerging new classes

This section, a robust framework for incremental learning in data streams with emerging new classes. By integrating the ADWIN algorithm for concept drift detection, K-means clustering, and Gaussian Naive Bayes (GNB) as the base classifier within an ensemble stratified bagging framework, our method effectively tackles the challenges posed by such dynamic data streams. The ADWIN algorithm is particularly important, enabling the prompt detection of concept drifts and new classes, which is critical for training new classifiers and maintaining accuracy over time.

The use of ensemble stratified bagging combined with DES significantly enhances predictive performance and robustness. Our evaluations confirm that the framework is highly effective in classifying data streams with evolving distributions, especially when GNB serves as the base classifier. In addition to its high

accuracy, the framework's strengths lie in its adaptability, efficiency, and scalability. The dynamic updates to the model, based on incoming data, ensure continuous adaptation to shifting data patterns, supporting real-time reliability and relevance.

Looking ahead, future work could explore several avenues for improvement. These include the development of advanced concept drift detection algorithms, the exploration of alternative ensemble strategies, and the incorporation of deep learning techniques. Additionally, expanding the framework to address a broader range of real-world applications and refining the selection and prediction methods for classifiers will be crucial for further enhancing the methodology's effectiveness.

4.3 Heterogeneous transfer learning

This study introduces a comprehensive Approach for incremental learning in data streams, particularly focusing on heterogeneous transfer learning. The framework effectively addresses the challenges associated with this context by integrating concept drift detection through the ADWIN algorithm, employing eigenvectors, and utilizing ensemble classifiers. The efficacy of this approach was validated through extensive experiments on a variety of datasets, including benchmark datasets, real-world application streams, and synthetic data.

A key strength of this approach is the pivotal role played by the ADWIN algorithm, which enables the timely detection of concept drifts and changes in data patterns. This allows the framework, referred to as HTL, to adapt by training new base classifiers, ensuring their continued relevance and accuracy in real-time scenarios. The use of ensemble classifiers further enhances predictive performance and robustness by aggregating predictions from multiple base classifiers. Dynamic Ensemble Selection (DES) is utilized to select the most suitable base classifiers, optimizing overall performance.

Our thorough evaluation using various performance measures confirms the approach's effectiveness in addressing the challenges of heterogeneous multisource domains. The framework excels in accurately classifying data streams with evolving class distributions, particularly when leveraging ADWIN as the drift detector, eigenvectors for knowledge transfer across heterogeneous domains, and DES for classifier selection.

Beyond its strong accuracy and performance, the framework offers significant advantages in terms of adaptability, efficiency, and scalability. The ability to update the model dynamically, based on incoming data instances, ensures continuous adaptation to changing data distributions, maintaining the framework's reliability and relevance in real-time scenarios.

Looking ahead, future research could focus on further enhancing this approach. Potential improvements include developing advanced concept drift detection methods, refining classifier selection by focusing on positive target and multisource classifiers, and incorporating deep learning techniques to predict future drifts and optimize classifier performance. These advancements could further bolster the framework's capability to manage complex, evolving data streams in diverse real-world applications.