ARM-Stream: Active Recovery of Miscategorizations in Clustering-Based Data Stream Classifiers

Douglas Monteiro Cavalcanti [1], Ricardo Cerri [2], Elaine Ribeiro Faria [1]

[1]FACOM, Universidade Federal de Uberlândia, MG, Brazil.
[2]ICMC, Universidade de São Paulo, SP, Brazil.

Contributing authors: douglas.m.cavalcanti@gmail.com; cerri@icmc.usp.br; elaine@ufu.br;

Abstract

The non-stationary feature-class interaction problem arises in dynamic data stream environments, where online classifiers avoid frequent label requests by relying on feature-class interaction assumptions to obtain class information from unlabeled data, even though these very assumptions may become invalid as this interaction changes unexpectedly. Clustering-based data stream classifiers mitigate this by leveraging active learning strategies. However, because these strategies are integral to the update process, enhancing reliability within the classifier is typically limited to increasing labeling resources, diminishing the classifier's ability to learn from unlabeled data. In this paper we tackle this problem by decoupling the error-handling concerns from the classifier into a new framework called ARM-Stream, which works as a fail-safe layer that aims to intervene only when the classifier's update process would fail, otherwise preserving the classifier's original label resource usage patterns. ARM-Stream's architecture, defined through abstract modules, ensures easy integration across different classifiers and easy customization and extension of its modules. To the best of our knowledge, ARM-Stream is the first error recovery framework for clustering-based data stream classifiers. As a study case, we test the framework over three classifiers: MINAS [1], CDSC-AL [2], and ECHO [3], chosen for their differing update process. An off-the-shelf source code library for the framework is provided.

Keywords: Data Stream, Concept Drift, Concept Evolution, Active Learning

1 Introduction

Classifier design uses learning algorithms to estimate class probabilities from labeled samples. Traditional methods assume a stationary distribution, requiring only a single estimation [4–6]. However, with modern data streams often generated by non-stationary distributions, incremental learning models have become advantageous for their adaptability [7–11]. Clustering-based data stream classifiers stand out as they allow updates with unlabeled data [1–3, 12–19]. These classifiers update a model composed of cluster summaries by generating and categorizing new clusters as extensions of known classes or the emergence of new ones. However, these classifiers face the problem of having to infer class information from unlabeled data through assumptions about the relationship between the data distribution and classes posterior probabilities, even though this very relationship is subject to unexpected changes [7, 20]. We refer to this issue as the non-stationary feature-class interaction problem.

Classifiers often tackle the problem by embedding active learning strategies [21, 22] to acquire additional label information, improving update reliability [2, 3, 15, 16, 18, 19, 23]. However, these strategies are often defined as an inherent part of the classifier's update process, limiting their reuse across different classifiers. As a result, new classifiers frequently need to be developed from scratch to address similar issues, making it difficult to share or generalize improvements. Moreover, improving the update process reliability without redefining the classifier is often limited to increasing the label consultations, diminishing the potential to learn from unlabeled data.

To address these limitations, we introduce ARM-Stream, a classifier-independent framework that serves as an external error recovery layer for clustering-based data stream classifiers. ARM-Stream operates by detecting when a classifier's update process is likely to fail and intervenes selectively, applying error correction strategies. The framework detects unreliable categorizations using a query-by-committee strategy that avoids the overhead of training new models by forming a committee of decision rules over the preexisting model.

ARM-Stream's architecture is defined through abstract modules, allowing easy integration with different clustering-based data stream classifiers. This modular design provides a high degree of customizability and extensibility, thus, while reference implementations are provided for the framework modules, new implementations can be proposed and easily incorporated.

ARM-Stream contributes a novel solution to error recovery in clustering-based data stream classifiers by decoupling error handling from the update process itself, which, to our knowledge, has not been addressed in prior work. As a demonstration of ARM-Stream's applicability, we validate the framework with three different classifiers—MINAS [1], CDSC-AL [2], and ECHO [3]—chosen for their varied update mechanisms and distinct approaches to clustering. The experimental results provide robust evidence of ARM-Stream's efficacy and efficiency across multiple classifiers and datasets [1].

[1] The datasets and source code used in the experiments are available in these public repositories: https://github.com/douglas444/arm-stream-framework and https://github.com/douglas444/arm-cdsc-al. The ARM-Stream framework source code integrates natively with classifiers written in Java and provides an HTTP interface for those implemented in other languages, such as Python.

1.1 Paper Organization

The paper is organized as follows: Section 2 describes the main clustering-based data stream classifiers in the literature. Section 3 outlines definitions relevant to our approach, while Section 4 details the ARM-Stream framework, its modules, and the reference implementations. Section 5 describes the evaluation methodology, and Section 6 presents experimental results. Section 7 discusses the framework scope. Section 8 concludes with insights and future research directions.

2 Related Works

This section presents an overview of the cluster categorization methods used in clustering-based data stream classifiers, focusing on their theoretical limitations. We divide these classifiers into two groups: those that employ a semi-supervised learning algorithm in their update process [3, 12, 15, 18], and those that utilize unsupervised novelty detection algorithms [1, 2, 14, 16, 17].

In the semi-supervised scenario, the clustering is performed over a buffer of partially labeled data by minimizing the impurity among the labeled instances inside the same cluster and the dispersion of the cluster's instances. If the clustering algorithm fails at building a complete pure cluster in terms of labeled data, the approach may assign multiple labels to the cluster, weighting each label by frequency within the cluster [3, 12, 15].

In this scenario, the miscategorization of a cluster happens when the labeled portion of the data does not represent the actual cluster distribution. Suppose the clustering assumption [20] fails and instances from different classes get clustered together. In that case, the strategy can only address the problem if each class in the cluster has at least one of its instances labeled. Since the approach labels the instances before the clustering process, there's no guarantee that each cluster will contain enough labeled data to represent its inner distribution.

In the unsupervised scenario, a novelty detection procedure detects patterns within a buffer of unlabeled data. Some algorithms handle only one pattern per buffer, relying on indicators that identify a single cohesive cluster [14], while others detect multiple patterns via unsupervised clustering [1, 2, 16, 17]. Categorization is based on similarity to previously labeled clusters, marking sufficiently similar clusters as extensions of known classes and distinct ones as novel. However, these methods depend on clustering and low-density assumptions [20], which can fail, leading to impure clusters or clusters of different classes getting close to each other or clusters of the same class being apart by low-density regions of the feature space.

To evaluate ARM-Stream, we selected MINAS [1], CDSC-AL [2], and ECHO [3].

MINAS relies exclusively on unsupervised novelty detection algorithms for adjusting its learning model. If a cluster is categorized as a novelty, MINAS treats it as a novel pattern. Instances from the data stream explained by such clusters are classified with the respective pattern number, as MINAS does not obtain class information.

CDSC-AL also utilizes an unsupervised novelty detection algorithm to categorize the clusters. However, CDSC-AL uses the predicted category to guide a strategy that selects and labels a fixed percentage of instances from the clustered instances. These

labeled instances are then propagated to all clusters before they are added to the model.

Unlike MINAS and CDSC-AL, ECHO utilizes a semi-supervised clustering algorithm to adapt its learning model. Even though ECHO uses labeled data, it also keeps a buffer of unlabeled data only, composed of instances the learning model could not explain. When this buffer reaches its maximum size, an unsupervised novelty detection algorithm is applied to detect and declare a new class's emergence.

These three classifiers cover different integration scenarios, from unsupervised to semi-supervised models, allowing full assessment of ARM-Stream's capabilities.

3 Preliminary

This section presents the main definitions required to describe the proposed approach.

3.1 Concept drift and evolution over data streams

A data stream is an infinite sequence of instances $(x_1, x_2, \ldots, x_{now}, \ldots)$ in the D-dimensional feature space $\mathbb{R}^D$ [24, 25]. Each instance x_p has an associated arrival timestamp t_p, with $t_{p+1} = t_p + 1$ and $t_1 = 1$. Each instance is also linked to a data class c, which is unknown but can be queried at a specific labeling cost [13].

In dynamically changing environments, the data stream is non-stationary, meaning its underlying distribution can change over time. This phenomenon, known as concept drift, can occur unexpectedly and take various forms, altering the input data characteristics or the relationship between the data and their classes [7, 24].

The concept drift between two timestamps t_p and t_{p+q}, with $q > 0$, can be formally defined as [7, 8]

$$\exists X : p_{t_p}(X) \cdot p_{t_p}(c|X) \neq p_{t_{p+q}}(X) \cdot p_{t_{p+q}}(c|X), \tag{1}$$

where $p_t(X)$ is the distribution of features in the timestamp t, and $p_t(c|X)$ is the posterior probability of class c in the timestamp t.

Considering that the data is continuously generated, another phenomenon usually present in problems involving data streams is the concept evolution [25]. Concept evolution refers to the appearance of new classes in the data stream. Formally, let C_t denote the set of known classes at time t. A concept evolution occurs at time t_e if:

$$\exists c_{new} \notin C_{t<t_e} : \exists x_e \in \mathbb{R}^D \text{ with class } c_{new}, \tag{2}$$

i.e., there exists a class label c_{new} that has never appeared in the stream before time t_e, and for which instances begin to arrive in the stream from that point onward.

This phenomenon poses significant challenges, as the learning model must be able to detect and learn emerging classes without prior knowledge or labeled data, beyond distinguishing the occurrence of new classes from changes in the known classes [13, 25].

3.2 Active learning

Classification tasks require sufficient labeled data for effective training, but not all instances are equally valuable [21, 22]. For example, instances near decision edges may

provide more training value than those clearly belonging to a class. This suggests that the quality, as well as the quantity, of labeled data impacts training. To address high labeling costs, active learning strategies aim to reduce the required labels by actively selecting the most valuable instances for training.

An active learning system typically includes a query system for identifying these key instances and an oracle to label them, adapting the process to different needs and scenarios [21, 22].

3.3 Bayes classifier, Bayes error, and Grouped Error Estimate

The input data of a classifier is an instance of a random vector X [6]. Each random variable of X represents one of the D features of the objects of classification. Many instances of X represent a distribution of the vector in the D-dimensional feature space. The data classes are different distributions of X, each characterized by its density function. The probability density function that characterizes the distribution of class c_i in the feature space is called the conditional density of class i and is expressed as $p(X|c_i)$. The density function that considers all M classes in the feature space is called the unconditional density function of X, and is given by [6, 26]:

$$\sum_{j=1}^{M} p(X|c_j) \cdot p(c_j),$$

where $p(c_j)$ is the prior probability of class c_j.

If we consider that the prior probability and the conditional density of all classes are known, the real probability $p(c_i|x)$ of an instance x of X belonging to a class c_i can be calculated by [6]:

$$p(c_i|x) = \frac{p(x|c_i) \cdot p(c_i)}{\sum_{j=1}^{M} p(x|c_j) \cdot p(c_j)} \tag{3}$$

A discriminant function calculated from an inequation of the real probabilities of the different classes defines the Bayes classifier [6].

The classification risk of an instance x is calculated as $r(x) = 1 - p(c|x)$, where c is the class with the highest probability. The Bayes error, R, is the expectation of r over the random vector X, representing the minimum error achievable for any classifier. It remains above zero when there's a non-zero probability of an instance belonging to multiple classes. When the class density functions are unknown and labeling is costly R can be estimated through the Grouped Error Estimate [27, 28].

The Grouped Error Estimate, denoted by $\hat{R}$, is calculated over a finite set X' of unlabeled instances by taking the mean of the classification risk $r(x)$ of all instances $x \in X'$ [27]. In scenarios where $p(c|X')$ is unknown, a non-parametric approximation $\hat{r}(X')$ of $r(X')$ is used [27, 28], resulting in the following formula for estimating $\hat{R}$ [27]:

$$\hat{R} = \frac{\sum_{x \in X'} \hat{r}(x)}{n} \tag{4}$$

In this case, the randomness of $\hat{R}$ comes from two sources: one from the estimate $\hat{r}(X')$ of $r(X')$, and the other from X'[6].

4 Proposed Approach

Clustering-based data stream classifiers rely on a predefined hypothesis to categorize each cluster as either an extension of a known class or the emergence of a new one, before using it to update the decision model. To address the non-stationary feature-class interaction problem without being tied to any specific classifier, we abstract this decision rule as an abstract module named base-categorizer, focusing on improving its accuracy by intercepting clusters and correcting potential miscategorizations before the miscategorized clusters are used to update the learning model.

The error recovery framework proposed is called ARM-Stream (**A**ctive **R**ecovery of **M**iscategorization in Clustering-Based Data **Stream** Classifiers). The framework was built to meet the following criteria: (i) to be memory and computation-efficient, minimizing the impact on the classifier's performance; (ii) to detect errors with high precision, optimizing label usage; and (iii) to be customizable and extensible, allowing adaptation to various inductive biases.

ARM-Stream operates using a two-level active learning strategy. Instead of querying samples directly from the data stream, it evaluates clusters generated by the classifier's update process. It first identifies clusters likely to be miscategorized and then queries for the most informative instances in the context of the cluster's inner distribution. The two-level active learning strategy, shown in Fig. 1, efficiently targets labeling resources toward clusters, the core elements of the update process of clustering-based data stream classifiers.

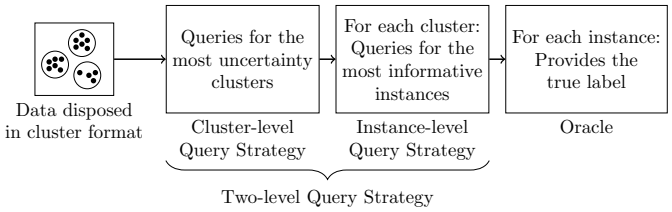


Fig. 1 The general flow of the proposed two-level active learning strategy.

As an implementation of the two-level active learning strategy, ARM-Stream needs to define two query strategies: one for querying the clusters and another for selecting informative data points within those clusters.

To query the clusters, ARM-Stream uses a query-by-committee strategy, forming a two-member committee by adding a secondary categorization rule sided with the base-categorizer. If the committee disagrees about the cluster category, ARM-Stream queries the cluster.

To query the instances of a cluster, ARM-Stream employs a query strategy that selects the K most informative instances. The labels of these instances are queried and used to re-categorize the cluster.

The categorization hypothesis used to compose the two-member committee with the base-categorizer is abstracted in a module called meta-categorizer. The meta-categorizer receives this name because, instead of keeping a learning model of its own,

it relies on the learning model kept by the base classifier to define its decision rule. In this way, it avoids the memory and computational complexity of a new learning model. However because the base-categorizer is likely to already consider the learning model in its own decision rule, the meta-categorizer needs to significantly diverge in the statistical approach, ensuring the contrasting perspectives in the committee.

The selection of the K most informative instances is managed by the active-categorizer, which guides ARM-Stream's instance-level queries within each queried cluster. Together, the two-member committee and the active-categorizer forms ARM-Stream's two-level query strategy.

While defined in terms of the meta and active categorizers, ARM-Stream is decoupled from the actual implementation of these modules, allowing custom implementations to accommodate various classifier designs and inductive biases.

A clustering-based data stream classifier is compatible with the framework if it presents the following characteristics:

- At some point in the flow of the classifier, a valid cluster of unlabeled or partially labeled data is made available.
- A base-categorizer can be defined, meaning that, at some point in the update process of the classifier, a cluster will be used to update the model representing either the extension of a known class or the emergence of a new one.
- The composition of the learning model of the classifier includes a set of cluster summaries representing the known classes. We refer to this set as the data class summary.

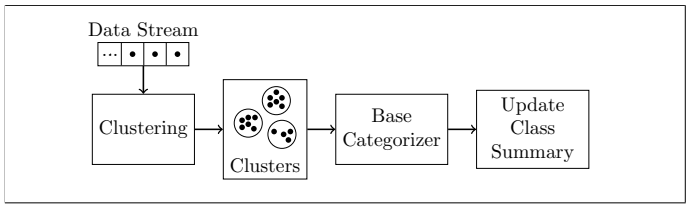
When the cluster is assigned to a known class, we consider that the base-categorizer categorized it as "known", otherwise as "novelty". Clusters categorized as something else, like noise, are not considered. We refer to the moment when the base-categorizer categorizes a cluster as the interception point. ARM-Stream works as an interceptor, called at the interception point. It receives as input the cluster, the category predicted by the base-categorizer, and the data class summary.

Fig. 2 shows the base classifier's cluster categorization flow and how ARM-Stream's modules integrate with the base classifier's modules. Subfigure (a) illustrates the original update process, where clusters are categorized by the base categorizer before updating the class summary. Subfigure (b) shows the updated flow after integrating ARM-Stream, which introduces a committee formed by the base and meta categorizers. If their predictions disagree, an active categorizer is invoked to resolve the conflict.

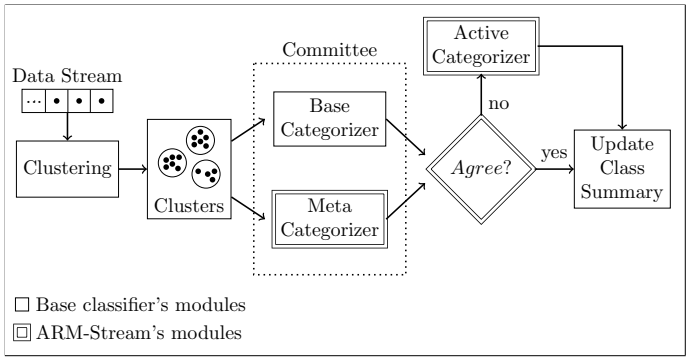
Following, we describe in detail the meta-categorizer and active-categorizer modules. We also present reference implementations for both modules.

4.1 The meta-categorizer module

When a new cluster is intercepted, the meta-categorizer receives the target cluster and data class summary as inputs and outputs the predicted category. It can use any decision rule to categorize a cluster as concept evolution or drift, provided it employs a different hypothesis than the base-categorizer to ensure diverse perspectives.



(a) General cluster categorization flow.



(b) General cluster categorization flow after integration of ARM-Stream.

Fig. 2 General cluster categorization flow in clustering-based data stream classifiers. The data stream classification task is omitted to emphasize the classification model update process. The base-categorizer abstraction is used to decouple the figure from concrete implementations of the underlying cluster categorization algorithm used in the classifier's update process.

Following, we present the reference implementations for the meta-categorizer module.

4.1.1 Meta-categorizer reference implementations

The meta-categorizers in this paper are based on a new assumption: if $\hat{r}(X')$, an estimate of $r(X')$ (see Section 3.3), indicates that the Grouped Error Estimate $\hat{R}$ exceeds a threshold, the cluster is more likely to belong to an unknown class than to a known one. The assumption comes from the fact that high Bayes error regions yield less classification information compared to low Bayes error regions [6, 26–28]. Consequently, if a cluster lacks sufficient classification information about known classes, we assume it is more likely to belong to an unseen class.

To implement a meta-categorizer based on the Grouped Error Estimate, a classification rule for estimating $r(X')$ must be defined. To avoid the computational and memory overhead of training a new model, the meta-categorizer needs to rely on data class summary to define the decision rule.

As explained earlier in this section, the data class summary is derived from the baseline classifier's learning model and is composed of multiple cluster summaries. Each cluster summary contains information about the cluster it is summarizing.

Through this information we can obtain the cluster's centroid and a label associating the cluster to a data class.

In this context, at any point during the classifier's lifetime, each known class may be associated with multiple cluster summaries within the model, where each cluster summary provides its own centroid. These centroids are used to define the classification rules employed by the meta-categorizer. Specifically, to implement meta-categorizers based on the Grouped Error Estimate, we adopt a first nearest neighbor (1-NN) approach.

In this setting, classifying an instance using the 1-NN classifier involves computing its distance to the nearest centroid among all cluster summaries. The class with highest classification probability will be the one associated with the cluster summary whose centroid is the closest to the instance.

While this rule may appear simplistic, it serves as a foundational implementation for using the Grouped Error Estimate, offering a straightforward way to test its applicability in novelty detection.

Following, we present the mathematical development of the formula to calculate the Grouped Error Estimate of a cluster using the 1-NN of cluster summaries.

In the context of the 1-NN classification rule, the estimate $\hat{p}(x|c_i)$ of the probability $p(x|c_i)$ of an instance x given a class c_i is given by [27, 28]:

$$\hat{p}(x|c_i) = \frac{1}{L_i(x)} \tag{5}$$

where $L_i(x)$ is the area around the instance x, with a radius equal to the distance between x and the closest centroid belonging to a cluster summary associated with the class c_i.

Accordingly with [27, 28], $L_i(x)$ can be given by:

$$L_i(x) = \frac{2 \cdot d(x, \overline{x_i})^D \cdot \pi^{D/2}}{D \cdot \Gamma(D/2)} \tag{6}$$

where d is the euclidean distance function, D is the dimensionality of the feature space, $\overline{x_i}$ is the centroid representing the class c_i, and Γ is the gamma function, defined as $\Gamma(n) = (n-1)!$, if $n \in \mathbb{Z}_+$, or as $\Gamma(n) = \int_0^\infty x^{n-1} \cdot e^{-x} dx$, if $n \in \mathbb{R}_+$, where n is the input value for the function.

The estimate $\hat{p}(c_i)$ of $p(c_i)$ is given by [27, 28]:

$$\hat{p}(c_i) = \frac{1}{M} \tag{7}$$

where M is the number of classes the data class summary knows.

Using equations 5, 6, and 7, the unconditional density function can be estimated as follows (See Section 3.3):

$$\sum_{j=1}^{M} \hat{p}(x \mid c_j) \cdot \hat{p}(c_j) = \sum_{j=1}^{M} \frac{1}{M \cdot L_j(x)} = \sum_{j=1}^{M} \frac{D \cdot \Gamma(D/2)}{M \cdot 2 \cdot d(x, \overline{x_j})^D \cdot \pi^{D/2}}$$

This simplifies to:

$$\sum_{j=1}^{M} \hat{p}(x \mid c_j) \cdot \hat{p}(c_j) = \frac{D \cdot \Gamma(D/2)}{M \cdot 2 \cdot \pi^{D/2}} \sum_{j=1}^{M} d^{-1}(x, \overline{x_j})^D \tag{8}$$

where d^{-1} represents the inverse Euclidean distance function.

Based on Equations 3 and 8, the probability $p(c_i \mid x)$ of an instance belonging to class c_i is estimated as:

$$\hat{p}(c_i \mid x) = \frac{\hat{p}(x \mid c_i) \cdot \hat{p}(c_i)}{\sum_{j=1}^{M} \hat{p}(x \mid c_j) \cdot \hat{p}(c_j)} = \frac{\frac{D \cdot \Gamma(D/2)}{M \cdot 2 \cdot \pi^{D/2}} \cdot d^{-1}(x, \overline{x_i})^D}{\frac{D \cdot \Gamma(D/2)}{M \cdot 2 \cdot \pi^{D/2}} \cdot \sum_{j=1}^{M} d^{-1}(x, \overline{x_j})^D}$$

This simplifies to:

$$\hat{p}(c_i \mid x) = \frac{d^{-1}(x, \overline{x_i})^D}{\sum_{j=1}^{M} d^{-1}(x, \overline{x_j})^D} \tag{9}$$

The classification risk $r(x)$ for an instance x being classified as c_i is estimated by:

$$\hat{r}(x) = 1 - \hat{p}(c_i \mid x) \tag{10}$$

From Equation 9, 10 and 4, we propose the first meta-categorizer, referred to as 1-NN Average Risk (NNAR).

The NNAR technique computes the Grouped Error Estimate of a cluster by measuring the distance from each instance to all centroids in the data class summary. It applies equations 9 and 10 to determine each instance's classification risk, averaging these with equation 4. If the average risk exceeds a threshold, the cluster is categorized as "novelty"; otherwise, it is "known".

NNAR computes distances $s \cdot n$ times, where s is the number of centroids and n is the number of instances. To reduce computations, we propose the 1-NN Centroid Risk (NNCR), which estimates the Grouped Error Estimate by calculating the classification risk of the cluster's centroid using only equations 9 and 10. This method assumes $r(X') = r(\overline{x'})$ (centroid of X') and reduces computations to s times. Algorithms for the NNAR and NNCR methods are are available at Appendix A.

We also propose variations of NNAR and NNCR that modify Equation 8 to include both centroids and the clusters' standard deviations. This adjustment scales the distance between the centroid and the target instance by the corresponding standard deviation. The classification probability is given by:

$$\hat{p}(c_i \mid x) = \frac{[d^{-1}(x, \overline{x_i}) \cdot \sigma_i^{-1}]^D}{\sum_{j=1}^{M} [d^{-1}(x, \overline{x_j}) \cdot \sigma_j^{-1}]^D} \tag{11}$$

where σ_i^{-1} is the inverse standard deviation of class c_i. The classifier represented by Equation 11 is called the First Nearest Denser Neighbor (1-NDN), while the variations are named 1-NDN Average Risk (NDNAR) and 1-NDN Centroid Risk (NDNCR). This approach is based on the low-density assumption [20], suggesting that denser clusters are less likely to be near the decision boundary, leading to better class characterization. A summary of the proposed meta-categorizers is presented in Table 1.

The classification risk of an instance ranges from 0 to $1 - \frac{1}{M}$. Since the Grouped Error Estimate of a cluster is the mean classification risk of its instances, it also lies

Table 1 Proposed meta-categorizers.

Meta-Categorizer	Grouped Error Estimate	Equations
NNAR	Average 1-NN classification risk of the cluster.	9, 10 and 4
NNCR	1-NN classification risk of the cluster's centroid.	9 and 10
NDNAR	Average 1-NDN classification risk of the cluster.	11, 10 and 4
NDNCR	1-NDN classification risk of the cluster's centroid.	11 and 10

within this interval. The categorization threshold t can be defined as $t = (1 - \frac{1}{M}) \cdot factor$, where $factor$ is a parameter between 0 and 1.

A key consideration is the behavior of the classification probability estimate (Eq. 9) in high-dimensional spaces. When a single class c_i reaches the classification probability, the inverse distance $d^{-1}(x, \overline{x_i})$ to the nearest centroid of c_i is the maximum among all classes. Thus, as dimensionality D increases, we have $\lim_{D\to\infty} \frac{d^{-1}(x,\overline{x_i})^D}{\sum_{j=1}^{M} d^{-1}(x,\overline{x_j})^D} = 1$, leading the classification probability estimate to approach 1 and the classification risk to approach 0. Consequently, for high-dimensional datasets, the Grouped Error Estimate compresses close to zero, making it difficult to effectively define the threshold separating low and high estimates.

To mitigate this issue, we replace D with 1 in Equations 9 and 11. This adjustment maintains consistent threshold values across datasets of varying dimensionality.

4.2 The active-categorizer

Whenever the base-categorizer and the meta-categorizer diverge about the category of a cluster, the active-categorizer is called. The active-categorizer is responsible for querying the data at the instance-level. The module also categorizes the cluster using the label information obtained.

Following, we present the reference implementations for the active-categorizer module.

4.2.1 Active-categorizer reference implementations

We propose two different implementations for the active-categorizer. Both implementations select K instances of the cluster to be labeled, where K is a parameterized value. After selecting instances, the cluster is categorized by considering the majority class among the labeled instances. If the majority belong to a known class, the cluster is categorized as "known", otherwise as "novelty". The implementations are called the Majority of the K Centroid Neighbors (MKCN) and the Majority of the K Random (MKR). The implementations differ from each other by the way each one selects the instances to be labeled. MKCN active-categorizer selects from the cluster the K instances more close to the centroid to be labeled. MKR active-categorizer selects K random instances in the cluster to be labeled.

The hypothesis for the MKCN active-categorizer is that instances close to the centroid of the cluster provide more information about the cluster distribution and, consequently, are adequate for indicating the category that results in less inference

error. The hypothesis for the MKR is that all instances inside the cluster have the same informational value.

4.3 Time complexity

To calculate the Grouped Error Estimate for a given cluster, the NNAR and NDNAR meta-categorizers compute the Euclidean distance of every cluster's instance to every centroid obtained from the data classes summary. So, the time complexity of NNAR and NDNAR is $O(n \times s \times D)$, where n is the number of instances inside the cluster, s is the number of centroids in the classes summaries, and D is the dimensionality of the data.

The NNCR and NDNCR meta-categorizers calculate the Grouped Error Estimate of a given cluster by first computing the cluster's centroid and then computing the distance of the obtained centroid to every centroid in the class summaries. The time complexity to compute the cluster's centroid is $O(n \times D)$, while the time complexity to compute the distance of the obtained centroid to every centroid in the class summaries is $O(s \times D)$. So, the final time complexity of NNCR and NDNCR is $O(D \times (s + n))$.

To select which cluster's instances to ask the oracle for the label, the MKCN active-categorizer starts by computing the cluster's centroid. This procedure presents a time complexity of $O(n \times D)$. After the centroid is computed, the method calculates the distance of the centroid to every cluster's instance while keeping track of the K closest instances. The distance computation and the selection of the K instances closer to the centroid have a time complexity of $O(n \times (K + D))$. After the K instances are selected, each one is labeled by the oracle. Considering that an instance labeling by the oracle has the constant cost of $O(1)$, the time complexity of labeling the K selected instance is $O(K)$. So, the final time complexity of the MKCN active-categorizer is $O(n \times (K + D))$.

The MKR active-categorizer chooses at random K cluster's instances to be labeled by the oracle. Since choosing an instance at random has a constant cost $O(1)$, the time complexity of the MKR active-categorizer is $O(K)$.

5 Evaluation Strategy

This section describes the evaluation strategy for ARM-Stream, defining two integration methods: tight integration, the standard approach, and loose integration, a non-intrusive method that preserves the base classifier's original behavior. We then present the evaluation measures used to assess the framework.

5.1 Integration with Clustering-Based Classifiers

ARM-Stream can be integrated with clustering-based data stream classifiers in two different ways: tight and loose integration.

5.1.1 Tight Integration

Tight integration is the standard way to integrate the framework into a base classifier, as described in Section 4. In this setting, the base classifier is modified to bypass the

base-categorizer's decision whenever the active-categorizer is called, instead relying on the active-categorizer's prediction.

Because this integration helps keep the learning model more consistent, the base-categorizer is expected to perform better by itself, as a more consistent learning model benefits the module directly, reducing miscategorizations and consequently the very need for ARM-Stream interventions, especially if new classes stop emerging. Because ARM-Stream also relies on the learning model through the meta-categorizer, it will also benefit from the interventions. Because of this self-impact, we propose the loose integration for isolated evaluation of the framework.

5.1.2 Loose Integration

In the loose integration, the baseline classifier's behavior remains unchanged. The active-categorizer's outputs are used for evaluation only. The true category is compared with those predicted by the base-categorizer, meta-categorizer, and active-categorizer. The goal is to assess ARM-Stream without altering the classifier's original behavior.

5.2 Evaluation Measures

A cluster may be categorized as "known" or "novelty." The true category is determined based on the learning model's state at the interception point. If most instances belong to known classes, the cluster is "known"; if they belong to unknown classes, the cluster is "novelty."

When a new cluster is generated, a categorization has two possible outcomes: hitting or missing the correct category. Let S be the set of the possible outcomes $S = \{Hit, Miss\}$. The outcome of a categorizer x for a cluster g can be represented as $s_{x_g} \in S$. The outcomes of the base-categorizer, meta-categorizer, and active-categorizer for a cluster g will be denoted by s_{B_g}, s_{M_g} and s_{A_g} respectively.

To count how many times a given combination of outcomes happened considering a set G of valid clusters generated in the execution of a classifier, let's define the function C_G as shown in equations 12, 13 and 14.

$$C_G(s) = \sum_{g \in G} \begin{cases} 1, & \text{if } s_{\mathrm{B}_g} = s \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

$$C_G(s_1, s_2) = \sum_{g \in G} \begin{cases} 1, & \text{if } (s_{B_g}, s_{M_g}) = (s_1, s_2) \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

$$C_G(s_1, s_2, s_3) = \sum_{g \in G} \begin{cases} 1, & \text{if } (s_{B_g}, s_{M_g}, s_{A_g}) = (s_1, s_2, s_3) \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

Let's consider N the set of all generated valid clusters whose true category is "novelty", and K the set of all generated valid clusters whose true category is "known". We can calculate the base-categorizer categorization sensitivity (Se) and categorization specificity (Sp) as shown in the equations 15 and 16, respectively.

$$Se = \frac{C_N(Hit)}{C_N(Hit) + C_N(Miss)} \tag{15}$$

$$Sp = \frac{C_K(Hit)}{C_K(Hit) + C_K(Miss)} \tag{16}$$

The final categorization performance measures, which account for miscategorizations corrected by ARM-Stream and any potential categorization errors, are called final sensitivity (Se_f) and final specificity (Sp_f), calculated through the Equations 17 and 18 respectively. These final measures can be directly compared to the Se and Sp to determine the exact improvement in categorization accuracy for each cluster category independently.

$$Se_f = \frac{C_N(Hit) - C_N(Hit, Miss, Miss) + C_N(Miss, Hit, Hit)}{C_N(Hit) + C_N(Miss)} \tag{17}$$

$$Sp_f = \frac{C_K(Hit) - C_N(Hit, Miss, Miss) + C_N(Miss, Hit, Hit)}{C_K(Hit) + C_K(Miss)} \tag{18}$$

To evaluate the overall error recovery considering ξ the set of all valid clusters, given by $\xi = N \cup K$, we calculate the error recovery rate ($ErrRec$) using Equation 19. This measure represents the proportion of miscategorizations corrected by ARM-Stream relative to the total number of miscategorizations.

$$ErrRec = \frac{C_\xi(Miss, Hit, Hit)}{C_\xi(Miss) + C_\xi(Hit, Miss, Miss)} \tag{19}$$

To evaluate ARM-Stream efficiency, we calculate the false positive ratio (FPR) through equation 20. This measure represents the percentage of clusters that, despite being correctly categorized by the base-categorizer, were unnecessarily queried by ARM-Stream, in relation to the total number of correctly categorized clusters by the base-categorizer.

$$FPR = \frac{C_\xi(Hit, Miss)}{C_\xi(Hit)} \tag{20}$$

We define the L and L_f measures to quantify ARM-Stream's impact on label queries. L represents the proportion of samples whose labels were queried by the base classifier's update process, while L_f includes label queries from both the base classifier' update process and ARM-Stream.

For experiments with tight integration, we define the Average Update Time (AUT) as the mean duration required to complete an update operation triggered by the base classifier during the processing of a dataset. Each update may involve generating multiple clusters, which are intercepted by ARM-Stream. To quantify the additional time introduced by this interception process, we define the *Latency* metric as the proportion of the update time attributed to ARM-Stream's operations. This provides a normalized view of the overhead introduced by ARM-Stream in relation to the total update time. AUT and *Latency* metrics are calculated through equations 21 and 22 defined bellow:

$$AUT = \frac{1}{U} \sum_{u=1}^{U} \Delta t_u \tag{21}$$

$$Latency = \frac{\sum_{u=1}^{U} \sum_{g \in G_u} \Delta t_g}{\sum_{u=1}^{U} \Delta t_u} \tag{22}$$

where $u \in \{1, \ldots, U\}$ is one of the U updates triggered by the base classifier during the processing of a given dataset, Δt_u is the total time taken by update u, and Δt_g is the portion of that time spent by ARM-Stream to intercept cluster g from the set G_u of all clusters generated during update u.

6 Experiments

In this section, we analyze ARM-Stream in the context of three different clustering-based data stream classifiers from the literature: MINAS [1], CDSC-AL [2], and ECHO [3]. We start by describing the benchmark datasets used in the experiments. Then, we describe the baseline results of three base classifiers. In sequence, we evaluate ARM-Stream by setting up a loose integration experiment for each base classifier. After that, we tightly integrate ARM-Stream to MINAS and ECHO.

All experiments involving time measurements were executed on a machine with an 11th Gen Intel Core i5-11300H CPU @ 3.10GHz, 16 GB RAM, running Ubuntu 24.04 LTS. The code was executed on a JVM (OpenJDK 11.0.10).

6.1 Datasets

Experiments were conducted using four benchmark datasets: MOA3 and Syn, synthetic datasets, and KDD99 and covtype, real-world datasets. Table 2 includes columns for the number of instances and classes for training and test phases. If no initial training is required, the classifier is applied to the entire dataset. Additional dataset details are provided in Appendix B.

Table 2 Benchmark datasets

		Training		Test	
Dataset	Dimensionality	*#Instances*	*#Classes*	*#Instances*	*#Classes*
MOA3	4	10000	2	90000	4
Syn	40	40000	7	360000	20
KDD99	34	48993	2	444619	5
covtype	54	47045	2	522911	7

6.2 Baseline results

Tables 3, 4, and 5 summarize the baseline results of MINAS, CDSC-AL, and ECHO respectively. The $\#Novelty$ column shows the number of true novelty clusters generated, while the $\#Known$ column indicates the true known clusters. The L column represents the percentage of label consultations made by the classifier's update process (excluding those from the initial training phase). The Se and Sp columns reflect the base categorizer's performance in classifying the generated clusters.

Details about the definition of the base-categorizers and the the parameter configuration for the base classifiers are presented in Appendix C and D, respectively.

Table 3 MINAS' baseline results

	$\#Novelty$	$\#Known$	L	Se	Sp
MOA3	108	367	0%	100%	0.54%
Syn	25	9	0%	56%	100%
KDD99	39	525	0%	82.05%	49.90%
covtype	201	1042	0%	88.55%	29.55%

Table 4 CDSC-AL's baseline results

	$\#Novelty$	$\#Known$	L	Se	Sp
MOA3	1	30	10%	100%	83.60%
Syn	20	398	10%	0%	100%
KDD99	3	6928	10%	0%	96.49
covtype	7	39019	10%	0%	93.64%

Table 5 ECHO's baseline results

	$\#Novelty$	$\#Known$	L	Se	Sp
MOA3	44	9411	11.59%	100%	100%
Syn	16	37404	1.51%	100%	99.99%
KDD99	29	4265	4.47%	100%	99.08%
covtype	221	15673	16.04%	100%	98.30%

6.3 Loose Integration Results

This section presents the results of applying ARM-Stream to MINAS, CDSC-AL and ECHO in the loose integration setting. For MINAS and ECHO, ARM-Stream was set with the 1-NDN Centroid Risk meta-categorizer (NDNCR). For CDSC-AL, ARM-Stream was set with the the 1-NDN Average Risk meta-categorizer (NDNAR). For all classifiers, the active-categorizer used was the Majority of the K Centroid Neighbors (MKCN).

The meta-categorizer and active-categorizer combinations were chosen based on a comparison between the performance of the different implementations, which is presented in Appendix F and G. The threshold factor used in the experiments was decided through an analysis presented in Appendix E.

Table 6 presents the results of ARM-Stream integrated with MINAS. On the KDD99 dataset, ARM-Stream achieved the highest error recovery, correcting 97.03% of the miscategorizations made by MINAS' base-categorizer, while querying only 0.067% of the samples and maintaining a false positive rate of 12.92%. The lowest error recovery was observed on the Syn dataset, where 45.45% of miscategorizations were corrected. However, this dataset also yielded the best false positive rate of 0% and required minimal label querying, increasing the labeled data by only 0.001%.

Table 6 Results of the Loose Integration with MINAS.

	Se	Sp	L	Se_f	Sp_f	L_f	$ErrRec$	FPR
MOA3	100%	0.54%	0%	100%	87.19%	0.37%	87.12%	14.54%
Syn	56%	100%	0%	76%	100%	0.001%	45.45%	0%
KDD99	82.05%	49.90%	0%	82.05%	99.80%	0.067%	97.03%	12.92%
covtype	88.55%	29.55%	0%	89.05%	77.83%	0.12%	66.79%	26.54%

Table 7 presents the results of ARM-Stream integrated with CDSC-AL. On the covtype dataset, ARM-Stream achieved the highest error recovery, correcting 99.11% of the miscategorizations made by CDSC-AL's base-categorizer, increasing labeled samples from 10% to 10.5% and the lowest false positive rate of only 0.002%. Conversely, the lowest error recovery was observed on the MOA3 dataset, where ARM-Stream recovered 60% of miscategorizations with a false positive rate of 1.92% and an increase in labeled samples from 10% to 10.007%.

Table 7 Results of the Loose Integration with CDSC-AL.

	Se	Sp	L	Se_f	Sp_f	L_f	$ErrRec$	FPR
MOA3	100%	83.60%	10%	100%	93.44%	10.007%	60%	1.92%
Syn	0%	100%	10%	95%	100%	10.071%	95%	66.83%
KDD99	0%	96.49%	10%	0%	99.94%	10.055%	97.15%	0.77%
covtype	0%	93.64%	10%	0%	99.96%	10.5%	99.11%	0.002%

Table 8 shows ARM-Stream's performance with ECHO. On the KDD99 dataset, ARM-Stream achieved the highest error recovery, correcting 46.15% of miscategorizations made by ECHO's base-categorizer. This result required a slight increase in labeled samples, from 1.57% to 1.517%, and the lowest false positive rate of 0.86%. In contrast, the lowest error recovery occurred on the Syn dataset, where no miscategorizations were corrected, yielding the minimal false positive rate of 0.07% and a slight labeling increase from 4.47% to 4.48%. For the MOA3 dataset, ECHO's base-categorizer made no errors and ARM-Stream appropriately minimized its queries, resulting in a low false positive rate of 0.09%.

Table 8 Results of the Loose Integration with ECHO.

	Se	Sp	L	Se_f	Sp_f	L_f	$ErrRec$	FPR
MOA3	100%	100%	11.59%	100%	100%	11.60%	–	0.09%
Syn	100%	99.99%	1.51%	100%	99.99%	1.517%	0%	0.07%
KDD99	100%	99.08%	4.47%	100%	99.50%	4.48%	46.15%	0.86%
covtype	100%	98.30%	16.04%	100%	98.93%	16.06%	37.21%	0.13%

The results from the loose integration indicate that ARM-Stream effectively recovers a substantial portion of miscategorizations while minimizing queries in instances where the base-categorizer performs well independently. For ECHO, despite its already high categorization accuracy, ARM-Stream demonstrated a strong error-recovery capability, retrieving a considerable portion of the base-categorizer's errors. This was achieved with high query precision, minimal labeling increase, and a false positive rate below 1% for all datasets, highlighting ARM-Stream's ability to avoid label consultations when the classifier's update process performs well by itself.

6.4 Results of Tight Integration with MINAS

This section presents the results of ARM-Stream's tight integration with MINAS. Table 9 compares the number of true novelty clusters ($\#Novelty$) and true known clusters ($\#Known$) generated by MINAS and the tightly integrated version, denoted as $\text{MINAS}_{\text{ARM-Stream}}$, and also the average update time (AUT) for both the approaches. In the case of MINAS, the AUT includes both the time required by the "Algorithm for Detection of Novelty Patterns or Extensions" described in the original paper [1], and the time taken by ARM-Stream to intercept the clusters generated during this process. Table 10 shows the categorization results for $\text{MINAS}_{\text{ARM-Stream}}$.

As seen in Table 9, $\text{MINAS}_{\text{ARM-Stream}}$ produced fewer novelty clusters, which is expected since by labeling certain instances, ARM-Stream provides MINAS with information on previously unknown classes.

Table 9 Comparison between the number of clusters generated by MINAS and $\text{MINAS}_{\text{ARM-Stream}}$

	MINAS			$\text{MINAS}_{\text{ARM-Stream}}$		
	$\#Novelty$	$\#Known$	$AUT(ms)$	$\#Novelty$	$\#Known$	$AUT(ms)$
MOA3	108	367	462.57	2	473	436.85
Syn	25	9	990.25	19	15	908.60
KDD99	39	525	691.85	5	559	844.71
covtype	201	1042	1074.8	4	1511	1115.89

Table 10 shows that $\text{MINAS}_{\text{ARM-Stream}}$ achieved its lowest error recovery rate of 93.59% with the covtype dataset. The highest false positive rate occurred with the Syn dataset at 6.89%, where ARM-Stream successfully recovered 100% of miscategorizations. Overall, tightly integrating ARM-Stream with MINAS resulted in more reliable cluster categorization, maintaining a very low false positive rate.

The latency values were all bellow 2%, evidencing ARM-Stream's low time overhead over MINAS' update process.

Table 10 Performance of $\text{MINAS}_{\text{ARM-Stream}}$

	Se	Sp	L	Se_f	Sp_f	L_f	$ErrRec$	FPR	$Latency$
MOA3	100%	67.23%	0%	100%	100%	0.17%	100%	0.62%	1.35%
Syn	73.68%	100%	0%	100%	100%	0.001%	100%	6.89%	0.04%
KDD99	40%	72.98%	0%	40%	99.82%	0.034%	97.40%	0.73%	0.27%
covtype	100%	57.67%	0%	100%	97.28%	0.11%	93.59%	1.48%	1.53%

6.5 Results of Tight Integration with ECHO

This section presents the results of ARM-Stream's tight integration with ECHO. Table 11 compares the number of true novelty clusters ($\#Novelty$) and true known clusters ($\#Known$) generated by ECHO and the tightly integrated version, denoted as $\text{ECHO}_{\text{ARM-Stream}}$, and also the average update time (AUT) for both the approaches. In the case of ECHO, the AUT includes both the time required by the "Update Classifier" algorithm, described in the original paper [3], and the time taken by ARM-Stream to intercept the clusters generated during this process. Table 12 shows the categorization results for $\text{ECHO}_{\text{ARM-Stream}}$.

As shown in Tables 11 and 12, for the MOA3 and Syn datasets, $\text{ECHO}_{\text{ARM-Stream}}$ performed the same as under the Loose Integration setting (See Table 8). For MOA3 dataset, ECHO's base-categorizer achived the maximum categorization performance, leaving no errors for ARM-Stream to recover. For the Syn dataset, the base-categorizer was able to correctly categorize 99.99% of the true known clusters, performance that ARM-Stream could not improve. For both datasets, however, ARM-Stream successfully minimized label waste by maintaining a false positive rate below 0.1%.

For the KDD99 and covtype datasets, Table 11 shows that $\text{ECHO}_{\text{ARM-Stream}}$ generated more true novelty and true known clusters compared to the original ECHO. Unlike $\text{MINAS}_{\text{ARM-Stream}}$ (Table 9), the number of true novelty clusters did not decrease. This is because ECHO achieved 100% sensitivity, correctly categorizing all true novelty clusters. Combined with ARM-Stream's low false positive rate, very few of these clusters were queried, so labels from unknown classes were rarely added to the model, preserving their unknown status.

One possible explanation for the increased number of clusters is that by improving the model's consistency, ARM-Stream leads to more accurate detection of concept changes and, consequently, the generation of more valid clusters for updating the model.

The latency values show that, on average, the time overhead introduced by ARM-Stream accounts for only a small portion of the total time taken by the classifier's update process. This behavior is consistent with the time complexity analysis in Section 4.3, as ARM-Stream's execution time scales linearly with the dataset dimensionality, the size of intercepted clusters, and the data class summary size—factors to which the update process is already, at minimum, linearly sensitive, since

Table 11 Comparison between the number of clusters generated by ECHO and $\text{ECHO}_{\text{ARM-Stream}}$

	ECHO			$\text{ECHO}_{\text{ARM-Stream}}$		
	$\#Novelty$	$\#Known$	$AUT(ms)$	$\#Novelty$	$\#Known$	$AUT(ms)$
MOA3	44	9411	22.55	44	9411	26.29
Syn	16	37404	96.83	16	37404	104.31
KDD99	29	4265	22.87	46	8931	62.52
covtype	221	15673	21.55	246	25866	40.60

it is responsible for both clustering and the categorization of clusters through the base-categorizer.

As for ARM-Stream's error recovery performance, Table 12 shows that, even with ECHO's update process reaching a Sp of more than 99%, ARM-Stream was able to recover 20% of the errors on the KDD99 dataset and 4.02% on the covtype dataset, while maintaining a false positive rate of just 0.01% and 0.05%, respectively.

Table 12 Performance of $\text{ECHO}_{\text{ARM-Stream}}$

	Se	Sp	L	Se_f	Sp_f	L_f	$ErrRec$	FPR	$Latency$
MOA3	100%	100%	11.59%	100%	100%	11.60%	–	0.09%	15.80%
Syn	100%	99.99%	1.51%	100%	99.99%	1.517%	0%	0.07%	2.82%
KDD99	100%	99.66%	5.753%	100%	99.73%	5.754%	20%	0.01%	2.72%
covtype	100%	99.23%	15.04%	100%	99.26%	15.05%	4.02%	0.05%	3.42%

7 Scope and Applicability

From our experiments we see that ARM-Stream is easy and lightweight integrated with different clustering-based data stream classifiers. However, it is important to make clear the scope in which it is most effective. Thus, the following considerations are take-away messages to further clarify ARM-Stream current scope:

- **ARM-Stream helps interpret the nature of changes, but not their timing:** ARM-Stream does not maintain a separate model or determine when the classifier's model should be updated. It operates on the model and clusters produced by the classifier's existing update process. For this reason, ARM-Stream does not participate on detecting when changes occur. Instead, it focuses on helping identifying what kind of change happened.
- **ARM-Stream does not analyze the stream directly:** ARM-Stream operates at the instance level only after data has been clustered by the underlying classifier's update process. This design avoids the cost of processing the data stream directly but assumes the underlying clustering process generates coherent patterns.
- **ARM-Stream supports the model update process, not the stream classification:** ARM-Stream is designed to assist the model adaptation process by helping

categorize new patterns as either concept drift or concept evolution. While classification remains the main goal of data stream classifiers, ARM-Stream focuses on recovering errors in the categorization of unlabeled clusters before they are used to update the model. It does not classify individual instances, nor does it intervene when the classifier struggles to distinguish between known classes.

- **ARM-Stream also relies on feature-class interaction assumptions:** The meta-categorizers proposed (See Section 4.1.1) rely on the Grouped Error Estimate (See Section 3.3) to identify when a cluster may belong to an unknown class. This approach is based on the hypothesis that high error estimates indicate a mismatch between the cluster and the current model. While validated experimentally, this hypothesis is also a feature-class interaction assumption and thus is subject to the non-stationary feature-class interaction problem (See Section 1) and may fail as the data distribution changes. High error estimates values may also arise due to noise, overlapping distributions, imprecise estimates of the classification function $r(X')$ (See Section 3.3), or high variance in the cluster data. Additionally, as clustering-based classifiers often use parametric approximations for class densities, further work is needed to understand how these approximations affect the variance and reliability of Grouped Error Estimate. In this context, the meta-categorizers are not intended to provide definitive answers individually but to contribute to a committee-based decision process that balances multiple, diverging perspectives.

8 Conclusion and Future Work

In this paper, we addressed the non-stationary feature-class interaction problem in the context of clustering-based data stream classifiers by proposing a customizable and extensible error recovery framework called ARM-Stream.

ARM-Stream relies on the base-categorizer and meta-categorizer abstractions to define a query-by-committee strategy that allows the framework to detect unreliable cluster categorizations while avoiding memory overhead and keeping a linear time complexity. The clusters selected are then sent to the active-categorizer module, which recategorizes the cluster by querying for label information.

The results obtained in the experiments validate ARM-Stream as a powerful error recovery tool, able to improve the reliability of clustering-based data stream classifiers while preserving their advantage of being able to learn changes in the distribution of the stream through unlabeled data.

Future research involves extending ARM-Stream to additional classifiers and tightly integrating it with them. Moreover, given the good results obtained with the proposed meta-categorizers, studying the variance of the Grouped Error Estimate (Section 3.3) through parametric density estimates using cluster summaries (Section 4.1.1) is suggested as, to the authors' knowledge, this topic remains unaddressed in prior research. Additionally, we plan to explore the use of a dynamic threshold factor to reduce the meta-categorizers' reliance on this parameter.

Beyond that, while this paper focused on proposing and validating the concept of a generic, modular error recovery framework over clustering-based data stream classifiers, the experiments were intentionally conducted on classical datasets and relied on

simple distance-based metrics. This setup allowed us to demonstrate ARM-Stream's applicability across more general classifiers with differing update strategies. In future work, we plan to target more specific domains, such as text and image datasets, and develop meta-categorizers designed specifically for these types of data. As long as the classification model retains cluster summaries with central tendency and dispersion information, ARM-Stream remains applicable. Existing studies on similarity functions and cluster summarization techniques for streaming text and image data as [29] and [19] provide evidence that meta-categorizers can be effectively developed to support these domains.

References

[1] Faria, E.R., Leon Ferreira, A.C.P., Gama, J., *et al.*: Minas: multiclass learning algorithm for novelty detection in data streams. Data mining and knowledge discovery **30**(3), 640–680 (2016) https://doi.org/10.1007/s10618-015-0433-y

[2] Yan, X., Homaifar, A., Sarkar, M., Girma, A., Tunstel, E.: A clustering-based framework for classifying data streams. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, pp. 3257–3263 (2021). https://doi.org/10.24963/ijcai.2021/448

[3] Haque, A., Khan, L., Baron, M., Thuraisingham, B., Aggarwal, C.: Efficient handling of concept drift and concept evolution over stream data. In: 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pp. 481–492 (2016). https://doi.org/10.1109/ICDE.2016.7498264

[4] Bishop, C.M., Nasrabadi, N.M.: Pattern Recognition and Machine Learning vol. 4. Springer, New York, USA (2006)

[5] Jain, A.K., Duin, R.P.W., Mao, J.: Statistical pattern recognition: A review. IEEE Transactions on pattern analysis and machine intelligence **22**(1), 4–37 (2000) https://doi.org/10.1109/34.824819

[6] Fukunaga, K.: Introduction to statistical pattern recognition. Elsevier, San Diego, CA, USA (2013)

[7] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM computing surveys (CSUR) **46**(4), 1–37 (2014) https://doi.org/10.1145/2523813

[8] Losing, V., Hammer, B., Wersing, H.: Knn classifier with self adjusting memory for heterogeneous concept drift. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 291–300 (2016). https://doi.org/10.1109/ICDM.2016.0040

[9] Lughofer, E., Pratama, M.: Online active learning in data stream regression using

uncertainty sampling based on evolving generalized fuzzy models. IEEE Transactions on fuzzy systems **26**(1), 292–309 (2017) https://doi.org/10.1109/TFUZZ.2017.2654504

[10] Ksieniewicz, P., Woźniak, M., Cyganek, B., Kasprzak, A., Walkowiak, K.: Data stream classification using active learned neural networks. Neurocomputing **353**, 74–82 (2019) https://doi.org/10.1016/j.neucom.2018.05.130

[11] Zyblewski, P., Ksieniewicz, P., Woźniak, M.: Combination of active and random labeling strategy in the non-stationary data stream classification. In: International Conference on Artificial Intelligence and Soft Computing, pp. 576–585 (2020). https://doi.org/10.1007/978-3-030-61401-0_54

[12] Masud, M.M., Gao, J., Khan, L., Han, J., Thuraisingham, B.: A practical approach to classify evolving data streams: Training with limited amount of labeled data. In: 2008 Eighth IEEE International Conference on Data Mining, pp. 929–934 (2008). https://doi.org/10.1109/ICDM.2008.152

[13] Masud, M., Gao, J., Khan, L., Han, J., Thuraisingham, B.M.: Classification and novel class detection in concept-drifting data streams under time constraints. IEEE Transactions on knowledge and data engineering **23**(6), 859–874 (2010) https://doi.org/10.1109/TKDE.2010.61

[14] Abdallah, Z.S., Gaber, M.M., Srinivasan, B., Krishnaswamy, S.: Anynovel: detection of novel concepts in evolving data streams. Evolving Systems **7**(2), 73–93 (2016) https://doi.org/10.1007/s12530-016-9147-7

[15] Haque, A., Khan, L., Baron, M.: Sand: Semi-supervised adaptive novel class detection and classification over data stream. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, pp. 1652–1658 (2016). https://doi.org/10.1609/aaai.v30i1.10283

[16] Mohamad, S., Sayed-Mouchaweh, M., Bouchachia, A.: Active learning for classifying data streams with unknown number of classes. Neural Networks **98**, 1–15 (2018) https://doi.org/10.1016/j.neunet.2017.10.004

[17] Garcia, K.D., Poel, M., Kok, J.N., Carvalho, A.C.: Online clustering for novelty detection and concept drift in data streams. In: EPIA Conference on Artificial Intelligence, pp. 448–459 (2019). https://doi.org/10.1007/978-3-030-30244-3_37

[18] Wang, Z., Kong, Z., Changra, S., Tao, H., Khan, L.: Robust high dimensional stream classification with novel class detection. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 1418–1429 (2019). https://doi.org/10.1109/ICDE.2019.00128

[19] Lima, M.C., Faria, E.R., Barioni, M.C.N.: A robust hubness-based algorithm for image data stream classification. International Journal of Data Science and

Analytics, 1–19 (2024) https://doi.org/10.1007/s41060-024-00605-x

[20] Van Engelen, J.E., Hoos, H.H.: A survey on semi-supervised learning. Machine Learning **109**(2), 373–440 (2020) https://doi.org/10.1007/s10994-019-05855-6

[21] Cacciarelli, D., Kulahci, M.: Active learning for data streams: a survey. Machine Learning **113**(1), 185–239 (2024) https://doi.org/10.1007/s10994-023-06454-2

[22] Aggarwal, C.C., Kong, X., Gu, Q., Han, J., Philip, S.Y.: Active learning: A survey. In: Data Classification: Algorithms and Applications, pp. 571–605. CRC Press, Boca Raton, FL (2014). https://doi.org/10.1201/b17320

[23] Lughofer, E., Weigl, E., Heidl, W., Eitzinger, C., Radauer, T.: Recognizing input space and target concept drifts in data streams with scarcely labeled and unlabelled instances. Information Sciences **355**, 127–151 (2016) https://doi.org/10.1016/j.ins.2016.03.034

[24] Gama, J.: Knowledge Discovery from Data Streams. CRC Press, Boca Raton, FL, EUA (2010). https://doi.org/10.1201/EBK1439826119

[25] Faria, E.R., Gonçalves, I.J., Carvalho, A.C., Gama, J.: Novelty detection in data streams. Artificial Intelligence Review **45**(2), 235–269 (2016) https://doi.org/10.1007/s10462-015-9444-8

[26] Fukunaga, K., Kessell, D.: Nonparametric bayes error estimation using unclassified samples. IEEE Transactions on Information Theory **19**(4), 434–440 (1973) https://doi.org/10.1109/TIT.1973.1055049

[27] Fukunaga, K., Hostetler, L.: K-nearest-neighbor bayes-risk estimation. IEEE Transactions on Information Theory **21**(3), 285–293 (1975) https://doi.org/10.1109/TIT.1975.1055373

[28] Fukunaga, K., Mantock, J.M.: A nonparametric two-dimensional display for classification. IEEE transactions on pattern analysis and machine intelligence (4), 427–436 (1982) https://doi.org/10.1109/TPAMI.1982.4767276

[29] Aggarwal, C.C.: A survey of stream clustering algorithms. In: Data Clustering, pp. 231–258. Chapman and Hall/CRC, ??? (2018)

[30] Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., Seidl, T.: Moa: Massive online analysis, a framework for stream classification and clustering. In: Proceedings of the First Workshop on Applications of Pattern Analysis, pp. 44–50 (2010)

[31] Al-Khateeb, T.M., Masud, M.M., Khan, L., Thuraisingham, B.: Cloud guided stream classification using class-based ensemble. In: 2012 IEEE Fifth International Conference on Cloud Computing, pp. 694–701 (2012). https://doi.org/10.

1109/CLOUD.2012.127

[32] Frank, A., Asuncion, A.: Uci machine learning repository. http://archive.ics.uci.edu/ml (2010). University of California. School of information and computer science

[33] Aggarwal, C.C., Philip, S.Y., Han, J., Wang, J.: A framework for clustering evolving data streams. In: Proceedings 2003 VLDB Conference, pp. 81–92 (2003). https://doi.org/10.1016/B978-012722442-8/50016-1

Appendix A NNAR and NNCR Meta-categorizer Algorithms

Algorithm 1 and 2 presents the step-by-step of the NNAR and NNCR meta-categorizers.

Algorithm 1 Meta-Categorizer NNAR

Require: $Cluster$ - target cluster; S - data classes summary; T - threshold;
Ensure: $Category$ - predicted category;

$RiskSum \leftarrow 0$
for Each instance I_i in $Cluster$ **do**
  $MaxInverseDistance \leftarrow 0$
  $InverseDistSum \leftarrow 0$
  for Each class C_j in S **do**
    $Distance \leftarrow DistanceToClosestCentroid(I_i, S_{class=C_j})$
    $InverseDistance \leftarrow \frac{1}{Distance}$
    if $InverseDistance > MaxInverseDistance$ **then**
      $MaxInverseDistance \leftarrow InverseDistance$
    end if
    $InverseDistSum \leftarrow InverseDistSum + InverseDistance$
  end for
  $Probability \leftarrow \frac{MaxInverseDistance}{InverseDistSum}$
  $Risk \leftarrow 1 - Probability$
  $RiskSum \leftarrow RiskSum + Risk$
end for
$GroupedErrorEstimate \leftarrow \frac{RiskSum}{Size(Cluster)}$
if $GroupedErrorEstimate > T$ **then**
  $Category \leftarrow Novelty$
else
  $Category \leftarrow Known$
end if
return $Category$

Algorithm 2 Meta-Categorizer NNCR

Require: $Cluster$ $--$ target cluster; S $--$ data classes summary; T $--$ threshold;
Ensure: $Category$ $--$ predicted category;

$Centroid \leftarrow ComputeCentroid(Cluster)$;
$MaxInverseDistance \leftarrow 0$;
$InverseDistSum \leftarrow 0$;
for Each class C_i in S **do**
    $Distance \leftarrow DistanceToClosestCentroid(Centroid, S_{class=C_i})$;
    $InverseDistance \leftarrow \frac{1}{Distance}$;
    if $InverseDistance > MaxInverseDistance$ **then**
        $MaxInverseDistance \leftarrow InverseDistance$;
    end if
    $InverseDistancesSum \leftarrow InverseDistSum + InverseDistance$;
end for
$Probability \leftarrow \frac{MaxInverseDistance}{InverseDistSum}$;
$Risk \leftarrow 1 - Probability$;
if $Risk > T$ **then**
    $Category \leftarrow Novelty$;
else
    $Category \leftarrow Known$;
end if
return $Category$;

Appendix B Datasets Description

B.1 MOA3

This synthetic dataset was created using the MOA (Massive Online Analysis) software [30], and used by [25]. Normal-like distributions generated the dataset's instances. Each distribution is represented by a 4-dimensional hypersphere that moves in the hyperspace, generating changes in the classes' distributions. The stream starts with two classes, and two more appear during the stream, totalizing four classes in the dataset. The dataset contains 100.000 instances. The attribute values are between 0 and 1.

B.2 Syn

This synthetic dataset was used by [13], [31], and [25]. It contains both concept drift and concept evolution. The instances are generated by Gaussian distributions in the 40-dimensional space, with different means (-5.0 to +5.0) and variances (0.5 to 6) per class. Beyond that, some classes appear and disappear during the stream. The attribute values are between 0 and 1.

B.3 KDD99

This dataset in its original format can be found in the UCI dataset repository [32]. It refers to the real problem of real-time automatic detection of cyberattacks. Each instance of the dataset represents the information about a network connection, classified as normal, or one of the 22 different types of attack that can be condensed into five classes. This paper's version of the dataset corresponds to the 10% version of the original dataset. Beyond that, we removed 409 instances from the 49402 first instances to start the stream with only two known classes ("dos" and "normal"). The resulting dataset contains a total of 493612 instances. Only the original dataset's numeric attributes were considered, totaling 34 attributes normalized to the range [0, 1].

B.4 covtype

This dataset in its original format can be found in the UCI dataset repository [32]. It refers to a real problem of predicting the forest cover type from cartographic information such as elevation, slope, soil type, etc. The dataset contains 581.000 instances, with seven different types of forest cover. Each sample has 54 numeric attributes. All attributes were used and normalized to the [0, 1] interval.

Appendix C Base-categorizer definition

The base-categorizer, data classes summary, and interception point of MINAS, CDSC-AL, and ECHO are described in more detail in sequence.

C.1 MINAS

The data classes summary of MINAS [25] is defined as the set of all micro-clusters assigned to a known class in its classification model at a given moment. Its base-categorizer is defined as follows: If a cluster is assigned to a known class by the novelty detection procedure, it is categorized as "known", otherwise, as "novelty". Finally, we define the interception point as the moment after the cluster is categorized.

C.2 CDSC-AL

The data classes summary of CDSC-AL [2] is defined as the set of all the micro-level cluster summaries kept by the model in the current timestamp. Its base-categorizer is defined as follows: A cluster is categorized as known if it is merged with historical clusters during the cluster merge procedure. A cluster is categorized as novelty if not merged with historical clusters and if its density falls inside the one-sigma distance from the average density of historical clusters at the timestamp.

C.3 ECHO

The data classes summary of ECHO [3] is defined as the set of all pseudopoints of all models kept in the ensemble in the current timestamp. Its base-categorizer is defined as follows: If a cluster comes from the outlier buffer, it is categorized as a "novelty". If a cluster comes from the partially labeled window, it is categorized as "known" if

the class with the highest frequency among the cluster's labeled instances is known by the data class summary in its current state. If a cluster comes from the partially labeled window, it is categorized as "novelty" if the class with the highest frequency among the cluster's labeled instances isn't known by the data class summary in its current state. A class is known by the data class summary if it contains at least one pseudopoint where the class appears with the highest frequency.

ECHO presents two interception points. The first interception is defined as the moment soon before a cohesive cluster is detected in the outlier buffer. The second interception point is defined as the moment soon before a cluster resultant from the partially labeled window is summarized in a pseudopoint and used to build a new model. For both interception points, the cluster and its instance, the data classes summary in its current state, and the category defined by the base-categorizer compose the information to be intercepted by the framework.

Since originally ECHO could only detect one new class at a time, we needed to adapt the ECHO novelty detection algorithm to utilize the same benchmark datasets for MINAS and CDSC-AL. When ECHO's outlier buffer reaches its maximum size, instead of finding one cohesive cluster of unlabeled instances, we modified the procedure to apply a K-Means++ clustering algorithm over the buffer. Each of the resulting clusters is then categorized as a different novelty pattern added to the separated decision model. If an instance cannot be explained by ECHO's ensemble, before sending it to the outlier buffer, we try to classify the instance using the novelty pattern decision mode, using the same nearest neighbor approach used by the pre-existing decision rule for classification using the ensemble.

Appendix D Base classifiers parameter configuration

D.1 MINAS parameter configuration

In the experiments, the parameters of MINAS [25] were configured as follows. For all datasets, the number of instances to execute the novelty detection procedure ($TemporaryMemoryMaxSize$) was defined as 2000, the number of clusters generated in the novelty detection procedure ($NoveltyDetectionNumberOfClusters$) was defined as 100, the minimum number of instances in a cluster ($MinimumClusterSize$) was defined as 20 and the window size to forget outdated instances ($SampleLifespan$) was defined as 4000. The window size to send micro-clusters from the decision model to the sleeping memory ($MicroClusterLifespan$) was defined as 4000. We choose not to update the micro-clusters at each new instance explained by the model. The clustering algorithm used in the offline phase was CluStream [1, 33], and the K-Means++ for the online phase. The seed used by the random procedures of the K-Means++ algorithm was fixed for all runs. The parameters are listed in Table D1.

The decision rule to verify if a micro-cluster explains an instance was defined as follows: if the distance between the instance x and the centroid $\overline{x_i}$ of the micro-cluster i is lower than two times the standard deviation σ_i of the micro-cluster i, the instance is classified. In order words, if the condition $d(x, \overline{x_i}) < 2 \cdot \sigma_i$ is true, the instance x

Table D1 MINAS' parameters for all datasets

Parameter	Value
$TemporaryMemoryMaxSize$	2000
$NoveltyDetectionNumberOfClusters$	100
$MinimumClusterSize$	20
$SampleLifespan$	4000
$MicroClusterLifespan$	4000
$IncrementallyUpdateDecisionModel$	False
$OfflineClustering$	CluStream
$OnlineClustering$	KMeans++

is classified with the class of the micro-cluster i, where d is the Euclidean distance function.

The decision rule to verify if a target micro-cluster i is the extension of another micro-cluster j was defined as follows: if the distance between the centroids $\overline{x_i}$ and $\overline{x_j}$ is lower than the sum of the standard deviations σ_i and σ_j, the target micro-cluster i is said to be an extension of the micro-cluster j. In order words, if the condition $d(\overline{x_i}, \overline{x_j}) < \sigma_i + \sigma_j$ is true, the micro-cluster i is considered an extension of the micro-cluster j.

D.2 CDSC-AL parameter configuration

In the experiments, the parameters of CDSC-AL [2] were configured as follows. For datasets MOA3, Syn, and KDD99, we used the version of CDSC-AL intended for gradual concept drift, while for dataset covtype we used the version intended for abrupt drifts. For datasets MOA3, KDD99, and covtype, the chunk size was set to 2000, while for dataset Syn it was set to 1000. For all datasets, the percentage of labeled data by chunk was set to 10%. The parameters are listed in Table D2.

Table D2 CDSC-AL's parameters by dataset

Paramaters	MOA3	Syn	KDD99	covtype
$ChunkSize$	2000	1000	2000	2000
$LabeledData$	10%	10%	10%	10%
$Drifts$	*Gradual*	*Gradual*	*Gradual*	*Abrupt*

D.3 ECHO parameter configuration

In the experiments, the parameters of ECHO [3] were configured as follows. The radius of a pseudopoint ($ClusterRadius$) was set as two times the standard deviation of the respective cluster. For all datasets, the maximum size of the outlier buffer ($FilteredOutlierBufferMaxSize$) was defined as 2000, the number of clusters generated over the buffer ($NumClusters$) was defined as 25, the value of the gamma parameter ($Gamma$) was defined as 0.5, the active learning threshold ($ActiveLearningThreshold$) was defined as 0.4, the value of the sensitivity parameter ($Sensitivity$) was defined as 0.001, the confidence threshold ($ConfidenceThreshold$) was set to 0.7, the size of the ensemble ($EnsembleSize$) was defined as 5. The chunk

($ChunkSize$) and window size ($WindowSize$) was defined as the number of training instances ($\#Training$) divided by the ensemble size for each dataset. The seed used by the random procedures of the K-Means++ and MCIKmeans algorithms was fixed for all executions. The parameters are listed in Table D3.

Table D3 ECHO's parameters for all datasets

Paramaters	Value
$ClusterRadius$	$2 \cdot \sigma$
$FilteredOutlierBufferMaxSize$	2000
$NumClusters$	25
$Gamma$	0.5
$ActiveLearningThreshold$	0.4
$ConfidenceThreshold$	0.7
$Sensitivity$	0.001
$EnsembleSize$	5
$ChunkSize$	$\frac{\#Training}{EnsembleSize}$
$WindowSize$	$\frac{\#Training}{EnsembleSize}$

Appendix E Meta-categorizers' threshold factor parameter analysis

In this section, we analyze the meta-categorizers' threshold factor parameters to select a suitable threshold factor configuration for use in all the experiments. To do so, we execute MINAS and ECHO in a loose integration scenario with two benchmark datasets: MOA3 and covtype. For each meta-categorizer, each classifier is executed once for a different threshold factor value. The threshold factor varies between 0.1 and 1 in steps of 0.1. For each execution, we register the meta-categorizer categorization specificity ($CSp = \frac{TrueKnow}{TrueKnown+FalseNovelty}$) and sensitivity ($CSe = \frac{TrueNovelty}{TrueNovelty+FalseKnown}$). The obtained results are plotted in Fig. E1 for MINAS and Fig. E2 for ECHO. For each figure, the subfigures (a), (b), (c), and (d) refer to the application of the base classifier to the MOA3 dataset. The subfigures (e), (f), (g), and (h) refer to the application of the base classifier to the covtype dataset.

The results obtained for MINAS over the MOA3 dataset (Fig. E1 (a), (b), (c) and (d)) show that for all the meta-categorizers, the CSe measure kept the performance of 1 from the threshold value of 0 to 0.8, and then started to decrease at the threshold value of 0.9 until reaching the performance of 0 at the threshold value of 1. This result implies that all the meta-categorizers can correctly categorize all the true novelty clusters generated by MINAS for the MOA3 dataset if the threshold factor is set to 0.8 or a lower value. The graphics also show that the CSp grows almost linearly, meaning that the Grouped Error Estimate of the true known clusters is almost equally distributed over the range of possible values. As a consequence of that, the greater the value of the threshold, the better the CSp performance.

The results obtained for MINAS over the covtype dataset (Fig. E1 (e), (f), (g) and (h)) show that for all the meta-categorizers, the CSe continuously decreased as the

threshold factor increased. The decrease of CSe got more significant after the threshold value of 0.8 for the NNCR and NNAR, and 0.9 for the NDNCR and NDNAR. On the other hand, the CSp increased in an accelerated way as the threshold value increased.

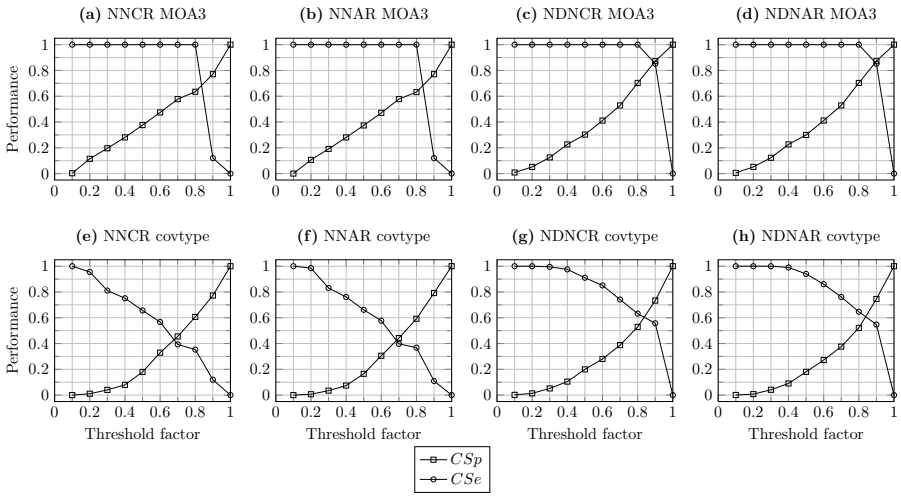


Fig. E1 Meta-categorizers' threshold factor parameter analysis for MINAS

The results obtained for ECHO over the MOA3 dataset (Fig. E2 (a), (b), (c) and (d)) shows that in the interval between 0.4 and 0.8, for all meta-categorizers, both CSe and CSp reached perfect performance. For the NNCR and NNAR applied to ECHO over the covtype dataset (Fig. E2 (e) and (f)), any threshold factor value in the interval between 0.4 and 0.8 is also suitable. However, only CSe reached maximum performance, with CSp variating between the performance values of 0.8 and 0.9. For the NDNCR and NDNAR applied to ECHO over the covtype dataset (Fig. E2 (g) and (h)), the best threshold factor would be 0.3 since, after this point, the CSe falls drastically.

From the analysis of the threshold factor parameter in the context of base classifiers, we may conclude that the threshold factor value of 0.8 is suitable for the NNCR and NNAR. As for the NDNCR and NDNAR, a better value for the threshold factor is 0.9, even though the experiments with ECHO and the covtype dataset point out an exception. The main conclusions used to base this configuration were two: (i) the higher the value for the threshold factor, the better the CSp performance; (ii) in general, the biggest decrease in the CSe performance occurs after the threshold factor of 0.8 for the NNCR and NNAR meta-categorizer, and after the threshold factor of 0.9 for the NDNCR and NDNAR meta-categorizers.

In the experiments (See Section 6), we configure the threshold factor of NNCR and NNAR to 0.8 and the threshold factor of NDNCR and NDNAR to 0.9.

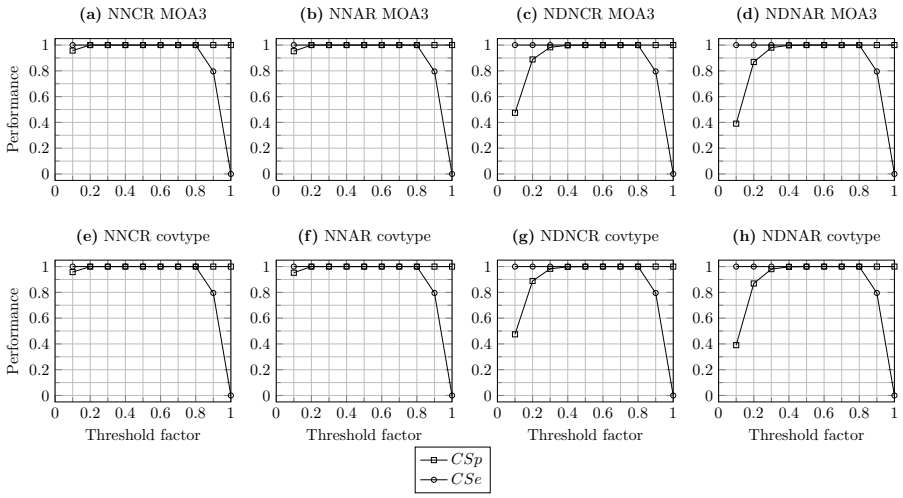


Fig. E2 Meta-categorizers' threshold factor parameter analysis for ECHO

Appendix F Meta-Categorizers Comparison

F.1 Meta-categorizers' query performance comparison

To compare the different implementations of the meta-categorizer we use the query sensitivity (F1) and query precision (F2) measures. These measures allow the evaluation of a meta-categorizer in the context of the two-member committee defined together with the underlying base-categorizer, thus, the performance of the meta-categorizer is assessed in the context of the base classifier ARM-Stream is being applied.

The QPr measure indicates the proportion of queried clusters that were miscategorized by the base-categorizer. The QSe measure quantifies the number of clusters that were incorrectly categorized by the base-categorizer and subsequently queried by the framework.

$$QSe = \frac{C_\xi(Miss, Hit)}{C_\xi(Miss)} \tag{F1}$$

$$QPr = \frac{C_\xi(Miss, Hit)}{C_\xi(Miss, Hit) + C_\xi(Hit, Miss)} \tag{F2}$$

Table F4 shows the querying performance of the meta-categorizers for the ECHO classifier.

The table shows that the NDNAR presented the highest values for the QPr and QSe measures except for the QPr in the MOA3 and KDD99 datasets. From this result, we may conclude that the NDNAR is the best meta-categorizer to compose the committee together with the base-categorizer.

Even though the NDNAR presented the best results, it is important to consider that the meta-categorizer presents higher computational complexity than the NDNCR and NNCR meta-categorizers. In this sense, the better results obtained by the NDNAR

may not justify its use as the NNCR and NDNCR reached a similar performance presenting a lower computational complexity. Compared to the NDNCR, the NNCR performed better for the QPr in MOA3 and the QSe in the Syn dataset. Other than that, NDNCR performed better than NNCR. Because of that, we chose NDNCR as the default meta-categorizer for the experiments involving MINAS.

Table F4 Cluster Query Performance in the Loose Integration with MINAS

	NNCR		NNAR		NDNCR		NDNAR	
	QSe	QPr	QSe	QPr	QSe	QPr	QSe	QPr
MOA3	63.28%	100%	63.01%	100%	87.12%	95.20%	87.12%	95.20%
Syn	90.90%	100%	100%	55%	45.45%	100%	90.90%	100%
KDD99	33.33%	75%	34.07%	75.40%	97.03%	87.33%	97.03%	89.11%
covtype	54.16%	67.99%	53.63%	66.88%	68.56%	80.09%	69.74%	80.61%

Table F5 shows that for the CDSC-AL classifier, the NDNAR meta-categorizer achieves the highest values across most QPr and QSe measures, suggesting it is the most effective choice to pair with the base-categorizer in the committee. Despite its higher computational complexity compared to NDNCR and NNCR, NDNAR's performance gains justify its use, leading us to select it as the default meta-categorizer for CDSC-AL in our experiments.

Table F5 Meta-Categorizer's Cluster Query Performance for CDSC-AL

	NNCR		NNAR		NDNCR		NDNAR	
	QSe	QPr	QSe	QPr	QSe	QPr	QSe	QPr
MOA3	60%	100%	60%	100%	60%	85.71%	60%	85.71%
Syn	95%	6.69%	95%	5.42%	95%	33.33%	95%	6.66%
KDD99	62.19%	7.59	94.30%	43.60%	96.34%	57.10%	97.15%	87.86%
covtype	18.77%	2.90%	99.15%	75.97%	64.59%	21.52%	99.11%	86.52%

Table F6 shows the querying performance of the meta-categorizers for the ECHO classifier.

In the table, some cells are marked with "−−". In the case where QPr is marked with "−−", it means that the committee agreed on all cluster categorizations. While in the case where QSe is marked with "−−", it means that the base-categorizer correctly categorized all clusters. In the cases where both the QPr and QSe are marked with "−−", it means that the base-categorizer and meta-categorizer both agreed on all categorizations and correctly categorized all clusters, so no clusters were queried by the committee.

Based on the results presented in the table, we can infer that the NDNAR meta-categorizer outperformed other options in terms of the QPr and QSe measures, except for the QPr on the Syn dataset. Nonetheless, the NDNCR meta-categorizer achieved similar performance while having lower computational complexity. Therefore, we have

decided to choose NDNCR as the default meta-categorizer for the experiments with ECHO.

Table F6 Meta-Categorizer's Cluster Query Performance for ECHO

	NNCR		NNAR		NDNCR		NDNAR	
	QSe	QPr	QSe	QPr	QSe	QPr	QSe	QPr
MOA3	−−	−−	−−	−−	−−	0%	−−	0%
Syn	50%	1.33%	0%	0%	50%	3.57%	100%	1.29%
KDD99	12.82%	14.70	12.82%	17.85%	46.15%	32.72%	46.15%	32.72%
covtype	0%	0%	0%	0%	37.59%	82.64%	37.59%	84.03%

Appendix G Active-Categorizers Comparison

To compare the different implementations of the active-categorizer we use the recovery rate (F1) and corruption rate (F2) measures. These measures allow the evaluation of a active-categorizer in the context of the cluster generated by the base classifier and queried by thee two-member committee, thus, the performance of the meta-categorizer is assessed in the context of the base classifier ARM-Stream is being applied.

$$RecR = \frac{C_\xi(Miss, Hit, Hit)}{C_\xi(Miss)} \tag{G3}$$

$$CorrR = \frac{C_\xi(Hit, Miss, Miss)}{C_\xi(Hit)} \tag{G4}$$

The tables show that the MKCN and MKR active-categorizer presented the same results for all datasets except covtype, where MKCN performed slightly better. Both active-categorizers presented a $CorrR$ higher than 0 for the covtype dataset. Both active-categorizer obtained some improvement by going from K = 1 to K = 3. The performance gain, however, may not justify triplicating the number of labeled data instances required.

Table G7 Active-Categorizer's Recovery for MINAS and NDNCR

	MKCN				MKR			
	$K = 1$		$K = 3$		$K = 1$		$K = 3$	
	$RecR$	$CorrR$	$RecR$	$CorrR$	$RecR$	$CorrR$	$RecR$	$CorrR$
MOA3	87.12%	0%	87.12%	0%	87.12%	0%	87.12%	0%
Syn	45.45%	0%	45.45%	0%	45.45%	0%	45.45%	0%
KDD99	97.03%	0%	97.03%	0%	97.03%	0%	97.03%	0%
covtype	67.23%	1.02%	67.37%	0.61%	67.10%	1.23%	67.76%	0.82%

The tables show that the MKCN and MKR active-categorizer presented the same results for all datasets. For both active-categorizers, no improvement was obtained from K = 1 to K = 3.

Table G8 Active-Categorizer's Recovery for CDSC-AL and NDNAR

	MKCN				MKR			
	$K = 1$		$K = 3$		$K = 1$		$K = 3$	
	$RecR$	$CorrR$	$RecR$	$CorrR$	$RecR$	$CorrR$	$RecR$	$CorrR$
MOA3	60%	0%	60%	0%	60%	0%	60%	0%
Syn	95%	0%	95%	0%	95%	0%	95%	0%
KDD99	97.15%	0%	97.15%	0%	97.15%	0%	97.15%	0%
covtype	99.11%	0%	99.11%	0%	99.11%	0%	99.11%	0%

As shown in the tables, the MKCN and MKR active-categorizer presented the same results for all datasets expect covtype, where MKCN performed slightly better. Both active-categorizers presented a $CorrR$ higher than 0 for the covtype dataset. For the same dataset, the performance of MKCN decreased from K = 1 to K = 3, while the performance of MKR increased.

Table G9 Active-Categorizer's Recovery for ECHO and NDNCR

	MKCN				MKR			
	$K = 1$		$K = 3$		$K = 1$		$K = 3$	
	$RecR$	$CorrR$	$RecR$	$CorrR$	$RecR$	$CorrR$	$RecR$	$CorrR$
MOA3	$--$	0%	$--$	0%	$--$	0%	$--$	0%
Syn	0%	0%	0%	0%	0%	0%	0%	0%
KDD99	46.15%	0%	46.15%	0%	46.15%	0%	46.15%	0%
covtype	37.21%	0%	36.84%	0.006%	36.84%	1.27%	36.84%	0%