Karlsruhe Institute of Technology

Change-Adaptive Active Learning on Data Streams

Master's Thesis of

Julien Aziz

at the Department of Informatics
KASTEL – Institute of Information Security and Dependability

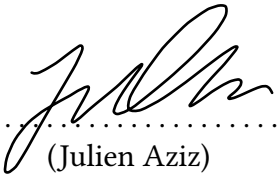
Reviewer:	Prof. Dr. Klemens Böhm
Second reviewer:	T.T.-Prof. Dr. Pascal Friederich
Advisor:	M.Sc. Marco Heyden

01. March 2023 – 01. September 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have submitted neither parts of nor the complete thesis as an examination elsewhere. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. This also applies to figures, sketches, images and similar depictions, as well as sources from the internet.

Karlsruhe, 1. September 2023



(Julien Aziz)

Abstract

Modern applications are increasingly encountering data streams. Real-time financial analytics, healthcare system monitoring, or social media constantly process continuously flowing data with rapid arrival rates. Supervised learning offers valuable insights and predictive capabilities. Yet, labeling vast amounts of data often induces substantial costs due to high computational demands or human interaction. Active learning presents a solution by selecting the most informative training data. However, modern data streams pose unique challenges to learning algorithms, such as large data volumes, rapid arrival rates, and unpredictable data behavior that can drastically impact model performance. Behavioral data changes, called concept drifts, occur at any time or severity. Furthermore, they can be local, manifesting only in a specific region of the data space. Adapting to these changes is crucial to maintain reliable predictions.

Previous solutions gradually adjust their model parameters or implement profound forgetting mechanisms to address potential changes, both inhering drawbacks. The former approach retains outdated concepts that might impair the prediction quality. The latter risks computational overhead and loss of vital data, which can delay a full recovery from changes. In this thesis, we aim to address the limitations of existing literature with a balanced adaption approach that selectively discards deprecated concepts while maintaining unaffected instances.

We propose a Clustering-Based Online Re-Active Learning framework (CORA) designed for online classification with sparse training data. Our approach leverages CluStream, a clustering algorithm for data streams. We enrich the clustering with label statistics to support the predictions, identify changes, and locate their affected areas. During stable phases, our model learns incrementally, guided by active learning to ensure a robust and resource-efficient training process. When faced with changes, the model adapts to affected areas while preserving relevant data. Thus, we achieve a faster recovery from regional changes compared to similar frameworks. Our adaption framework can be seamlessly integrated with any active learning strategy and incremental classifier, making it highly versatile and applicable to a wide range of scenarios.

We explore four design options, thereby comparing different model structures and alternative change detection approaches. Our evaluation includes a diverse set of real-world and artificial data streams. The artificial datasets cover a wide range of challenging data change behaviors while the real-world datasets simulate performances in real-world applications. Both provide valuable insights into the strengths and limitations of the proposed methods under different scenarios. Experiments showed that our partial adaption mechanism achieves faster recovery from local concept drifts than previous adaption approaches. In general, our approach surpasses related techniques on most evaluated data streams.

Zusammenfassung

Moderne Anwendungen stoßen immer häufiger auf Datenstromumgebungen, darunter Echtzeit-Finanzanalysen, die Überwachung von Gesundheitssystemen und soziale Medienplattformen. Obwohl überwachtes Lernen wertvolle Einblicke und Vorhersagefähigkeiten bietet, bringt das Labeln großer Datensätze erhebliche Kosten mit sich. Aktives Lernen bietet hier eine Lösung, indem es gezielt die informativsten Trainingsdaten auswählt. Bei der Anwendung auf moderne Datenströme stößt es jedoch auf besondere Herausforderungen, wie enorme Datenvolumina, schnelle Ankunftszeiten und unvorhersehbare Verhaltensänderungen in Daten, die die Modellleistung erheblich beeinflussen können. Diese Verhaltensänderungen können jederzeit mit unterschiedlichen stärken autreten. Außerdem, können sie lokal auftreten, dabei sich nur in bestimmten bereichen des Datenraums manifestierenl.

Frühere Lösungen passten entweder ihre Modellparameter stufenweise an oder führten intensive Vergessensmechanismen ein. Beide Ansätze haben ihre Nachteile. Das stufenweise Anpassen birgt das Risiko, veraltete Konzepte zu behalten, die die Vorhersagequalität beeinträchtigen könnten. Das Einführen von Vergessensmechanismen kann zu erhöten Rechenaufwand führen und wichtige Daten verlieren, wodurch sich die Reaktion auf Veränderungen verzögert. In dieser Arbeit beabsichtigen wir, die Einschränkungen der bestehenden Literatur mit einem ausgewogenen Anpassungsansatz zu überwinden, der veraltete Konzepte gezielt verwirft und dabei unbeeinträchtigte Daten beibehält.

Wir präsentieren ein Framework für Clustering-basiertes Online Re-Aktives Lernen (CORA) vor, das für die Klassifizierung in dynamischen Datenströmen entwickelt wurde. Unser Ansatz nutzt CluStream, einen unüberwachten Clustering-Algorithmus für Datenströme. Wir erweitern das Clustering mit Label-Statistiken, um Vorhersagen zu unterstützen, Änderungen zu identifizieren und ihre betroffenen Bereiche zu lokalisieren. In stabilen Phasen lernt unser Modell schrittweise, gesteuert durch aktives Lernen, um einen robusten und ressourceneffizienten Trainingsprozess sicherzustellen. Bei Änderungen passt das Modell sich den betroffenen Bereichen an und bewahrt dabei relevante Daten. So erreichen wir eine schnellere Anpassung an regionale Veränderungen im Vergleich zu ähnlichen Frameworks. Besonders hervorzuheben ist, dass sich unser Anpassungs-Framework nahtlos mit jeder aktiven Lernstrategie und jeden inkrementellen Klassifikator integriert lässt, was es äußerst vielseitig und für eine breite Palette von Szenarien anwendbar macht.

Wir untersuchen vier verschiedene Designoptionen und vergleichen unterschiedliche Modellstrukturen sowie alternative Ansätze zur Erkennung von Änderungen. Unsere Bewertung umfasst eine vielfältige Menge von realen und künstlichen Datenströmen, die ein breites Spektrum an anspruchsvollen Datenänderungsverhaltensweisen erfassen. Durch diese umfassende Analyse möchten wir wertvolle Erkenntnisse über die Stärken und Grenzen der vorgeschlagenen Methoden in verschiedenen Szenarien bieten.

Contents

List of Figures

List of Tables

1. Introduction

Data streams are becoming ubiquitous in modern applications due to rising real-time data generation such as online transactions, sensors, and social media [7]. With modern technologies and further connectivity, vast amounts of data are constantly flowing. Recognizing patterns or predicting future states within these time-sequenced, potentially infinite data streams has already become a mandatory part of modern society [20]. Whether it is detecting fraudulent activity in the financial sector, predicting machinery failures, or email spam filtering, predictive models operating on data streams are omnipresent. Traditionally, these supervised learning algorithms are applied in a stationary setting with a finite training set and non-evolving data behavior. Generating predictive models on data streams faces various new challenges, induced by the infinite nature and changing data [38]. Challenges are present because these streams are characterized by their immense volume, rapid arrival rates, and unpredictable changes in data patterns, commonly referred to as concept drifts [62].

With modern technologies and further connectivity, these challenges surpass our capabilities to store and process every incoming instance [7]. Active learning reduces the amount of required training data by strategically identifying the most informative instances. Traditional pool-based active learning selects these instances from a finite set of unlabeled data. However, to employ such models in dynamic data streams, algorithms need to address the imposed challenges. A prominent approach is to buffer the data stream into batches and apply stationary techniques. While this leverages well-studied techniques, it elevates resource demands and limits the ability to effectively address concept drifts [74]. Online active learning approaches process each instance instantaneously, making them resource-efficient and well-suited for modern data streams. However, they must explicitly consider concept drifts to reliably predict future outcomes.

Concept drifts are changes in the distribution that generates the data stream. They can occur at any time with different severity and characteristics [75]. The relationship between the input and the desired prediction target can deviate over time, making previously learned concepts obsolete. Furthermore, they can be local, manifesting only in a specific region of the data space. In online active learning, two dominant approaches have emerged: (1) Continuously adjusting the model parameters or (2) employing forgetting mechanisms. Models trained with the former technique benefit from long-term contexts and inherent adaption. However, when facing abrupt drifts with a certain severity, their reaction is delayed as they rely on old concepts and the adaption mechanism is not strong enough for a swift reaction [22]. In contrast, the latter group employs forgetting mechanisms to predict current states based only on the most recent instances. This is commonly done by maintaining a sliding window of recent instances and periodically retraining the model with it. While this results in a faster reaction to abrupt changes, it diminishes the model's stability as all past information is frequently discarded. This trade-off is commonly referred to as *stability-plasticity-dilemma* [3]. An algorithm decides to either preserve valuable knowledge for more robust systems (stability) or to maintain only the most recent data to swiftly learn new patterns (plasticity). The label sparsity in active learning further enhances this dilemma.

Explicit change detection appears to be a potent instrument to strike a balance here. Continuously preserving information in stable situations, while forgetting outdated concepts only if a drift is detected provides both stability and plasticity. However, false alarms or missed changes can severely limit the model quality over time. Furthermore, concept drifts can be regional, manifesting only in certain areas of the feature space [33]. Thus, only a subset of learned instances is affected, while the others still represent the current concept. Identifying changing areas and forgetting affected instances, preserves labeled training data. Hence, the model can learn the new concept on top of the still-relevant knowledge, resulting in more robustness and faster recoveries. Additionally, local change detection mitigates the negative impact of false alarms as only a subset is inadvertently discarded. Yet, the majority of previous work in online active learning focused on the development of effective instance identification techniques. Explicit adaption frameworks for active learning are rare. Existing approaches use global change detection with profound forgetting mechanisms, forfeiting stability and suffering from delayed recovery, as they discard all previous information upon a detected drift. If drifts only affect specific areas, they lose valuable information from unaffected instances that still retains the current concept. Thus, the model must learn the new concept again from scratch.

Bifet et al. (2010) [8] defined five fundamental requirements for data stream mining algorithms, summarized as: (1) Process an instance at a time, and inspect it once, (2) use limited time for processing, (3) use limited memory and (4) give predictions at any time. The fifth requirement states that models should "(5) adapt to temporal change". We identify that to effectively address all characteristics of changes in data streams, a model must additionally:

R1 Continously learn in stable stream states.

R2 Detect changes swiftly.

R3 Forget outdated training data.

R4 Preserve knowledge yielding current concepts.

1.1. Contributions

In this thesis, we investigate whether one can meet the above requirements using local change detection combined with a partial forgetting mechanisms. Our goal is to strike a balance in the *stability-plasticity-dilemma* in data stream environments where obtaining labels is costly.

We propose **C**lustering-based **O**nline **R**e-**A**ctive Learning (CORA), a framework designed to operate in dynamic data streams with high labeling costs and concept drifts. The novelty emerges from our local change adaption approach that combines a modified CluStream algorithm with an ensemble of change detectors. CluStream is an unsupervised online clustering technique designed to operate with minimal resources and continuous adaption [1]. It summarizes coherent subsets as aggregated statistics in Cluster-Feature (CF) and constantly refines its structure to reflect current data trends. We enrich the CFs with labeling statistics and recent training data, provided by the Online Probabilistic Active Learning strategy (OPAL) [35]. In parallel, we incrementally train a predictive base classifier, responsible for real-time class predictions. By constructing a change detector ensemble that monitors the enriched clustering, we are able to detect and localize potential concept drifts. Identified changes activate our adaption routine which involves updating the clustering structure as well as retraining the base classifier on preserved training data. Our adaption framework is highly flexible, allowing

for arbitrary incremental base classifiers and online active learning strategies to be seamlessly integrated. Furthermore, we offer different design options, all providing advantages in certain situations. We propose a CORA configuration that further leverages our clustering to support the base classier in forming class predictions. It is a paired ensemble of the incremental base classifier and majority voting from the clustering. This cooperation requires no additional resources and reflects its potential in our experimental results.

The thesis is structured as follows: Section 2 provides the foundation to comprehend our framework and to assess its potential. It involves key principles and challenges of data stream classification, active learning, and online clustering. In Section 3 we further investigate the current landscape of data stream active learning. In this comprehensive exploration, we identify further limitations of previous works and refine our requirements for online active learning approaches. Our framework CORA is presented in Section 4. Besides a detailed description of our detection, adaption, and classification mechanisms, we present four different configurations of CORA and highlight their potential advantages. In Section 5, we extensively evaluate the capabilities and limitations of all four CORA configurations. We investigate its sensibility to hyperparameters and provide guidelines to strategically select them based on the desired application. Our evaluation includes a comparison to renowned baseline approaches. We compare their prediction performance, their ability to detect changes, and the efficacy of the activated adaption in different active learning conditions. This is conducted on 8 widely used benchmarks, containing both real-world and artificial data streams.

2. Theoretical Foundation

This chapter lays the foundation for understanding key principles and methodologies applied in the domain of data stream mining and active learning. We provide a brief introduction to classification and data streams, thereby presenting the different facets of concept drifts and categorizing adaption instruments. Subsequently, we give a detailed introduction to active learning, covering the main components, and discussing the unique challenges it faces in data stream environments. In conclusion, we present the characteristics of online clustering and the widely recognized algorithm CluStream.

2.1. Classification in Data Streams

Classification on data streams is essential for analyzing and categorizing continuous flows of real-time data, allowing timely decisions and valuable insights. In this section, we define both the classification task and the characteristics of data streams. Subsequently, we provide a definition for concept drifts and distinct different types.

2.1.1. Classification

Classification involves categorizing instances into one of several predefined classes based on their features. It serves as an instrument to make informed decisions by associating patterns of data with known labels. For instance, predicting whether an email is spam or to classify customers based on their credibility for risk management [2]. The classifier learns such patterns and predictions by processing training data. We denote such a dataset as

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\},$$

where each instance is a tuple, containing the input $x \in \mathcal{X}$ and the target variable $y \in Y \subset \mathbb{N}$ from a finite set of unique classes $Y = \{y_1, y_2, \ldots, y_c\}$. In contrast, the target variable Y of a regression problem is continuous. In this thesis, we focus on classification, where the output is discrete. Formally, the goal in classification is to find a function

$$f : \mathcal{X} \to Y,$$

so that $f(x)$ returns the correct class label y for the input x. Thus, the classifier learns a boundary or rule to separate these classes based on the features of the data. Once trained, the classifier can be used to predict the class labels of new, unseen instances [10].

Popular algorithms for classification include Decision Trees, Naive Bayes, Support Vector Machines, Neural Networks, and many more. For more details on that we refer to the book "Pattern Recognition and Machine Learning" by Bishop (2006) [9] which provides a detailed overview of these topics.

2.1.2. Datastreams

Data streams present their own unique challenges and opportunities in the realm of classification. Unlike traditional static datasets, which are fixed in size and can be processed at once, data streams are unbounded and arrive in real-time [45]. We denote a data stream in context of a classification (and Regression) problem as:

$$S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t), \ldots\},$$

where each instance pair (x_t, y_t) arrives at a timepoint t and origins from a joint probability distribution $p_t(x, y)$ [16].

Due to the properties of modern data streams and special characteristics, stream classification poses additional challenges compared to stationary settings:

- **Volume and Arrival Rates**: One of the primary challenges with data streams is the sheer volume and speed of data. Handling and processing such vast amounts of information in real-time demands efficient algorithms and data structures.
- **Memory Constraints**: Given the unbounded nature of data streams, it is impossible to store all the data. Algorithms must thus make decisions based on limited memory, often requiring summaries or approximations of the data.
- **Real-time Processing**: For many applications, insights from data streams are valuable only if processed quickly. This necessitates real-time or near-real-time processing capabilities, balancing speed and accuracy [24].
- **Concept Drifts**: As data streams are dynamic, the underlying distribution of the data can change over time. This phenomenon, known as concept drift, poses challenges for predictive models, which may become outdated unless they adapt to the changing data [22].

Considering these challenges, Bifet *et al.* (2010) [7] defined fundamental requirements for stream mining algorithms:

B1 **Process an instance at a time, and inspect it (at most) once**: Given the infinite nature of streams, it's often not feasible or efficient to revisit data points. Thus, algorithms should be designed to process each data example only once.

B2 **Use a limited amount of time to process each instance**: To manage the velocity of incoming data, it's crucial that the algorithm processes each example in a fixed amount of time, ensuring real-time or near-real-time operations.

B3 **Use a limited amount of memory**: Given the voluminous nature of data streams, algorithms should operate within limited memory requirements.

B4 **Be ready to give a prediction at any time**: The algorithm should always be in a state where it can make a prediction based on the data it has seen up to that point.

B5 **Adapt to temporal changes**: Regarding the possibility of concept drift in data streams, algorithms should be adaptive and able to quickly adjust to changing data distributions.

Algorithms adhering to these guidelines are well-suited to address modern data streams in their complexity and dynamic nature.

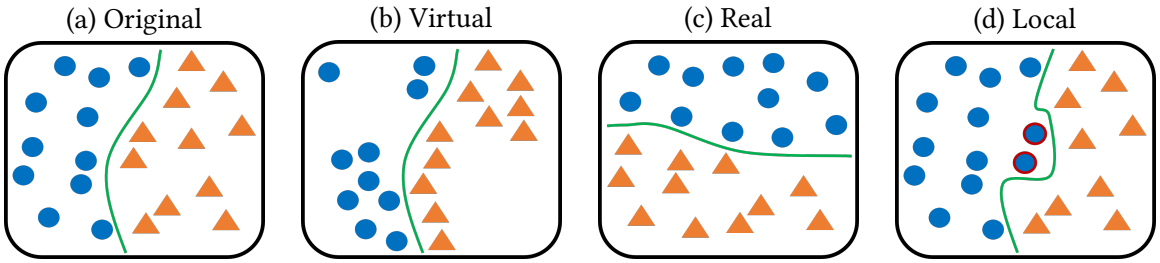


Figure 2.1.: Illustration of different types of concept drifts

2.1.3. Concept Drifts

Changes in the underlying data generating distribution $p(x, y)$, known as concept drift, challenge machine learning algorithms. Already established relationships between the input data x and the target variable y might become outdated which crucially affects the model's prediction quality. Concept drifts occur in various forms and severity. However, all involve a change in the joint probability distribution $p(x, y)$ [61]. Figure 2.1 visualizes the following types of concept drifts:

- **Virtual Drift** affects only the probabilities $p(x)$ or $p(y)$ without altering the conditional probability $p(y|x)$
- **Real Concept Drift** refers to a change in the conditional probability $p(y|x)$, leading to shift of the decision boundary [22]. More precisely, if the probability of observing a label y given an input x at time t is represented as $p_t(y|x)$, a real drift can be indicated when:

$$p_t(y|x) \neq p_{t+1}(y|x)$$

  for some timesteps t and $t + 1$.

- **Local Drift** is confined to a subset of the data. For instance, a deviation of the decision boundary in a specific region of the instance space [42].

Virtual drifts do not necessarily pose problems for a classification task, as the input-output relationship remains. In contrast, Real Drifts can significantly compromise the model's accuracy as the decision boundary alters and the majority of the learned concepts become outdated. However, these deviations might only affect a certain region. Efficiently addressing these local concept drifts requires a model to identify the region affected by the drift and to adapt only in the identified area.

Drifts can be further categorized by their rate and pattern [22]. We illustrate these types in Figure 2.3:

- **Sudden Drift**: This form of drift signifies an abrupt change in the data distribution with little to no transition time. Detection mechanisms must be particularly sensitive to cope with such rapid changes as no transition period exists [22].
- **Gradual Drift**: Refers to situations where the old concept gradually transitions into a new one over an extended period. It might involve periodic oscillations between the old and new concepts before the new one takes over completely. Gradual drifts require algorithms to have the flexibility to learn over extended periods without making premature adjustments [73].

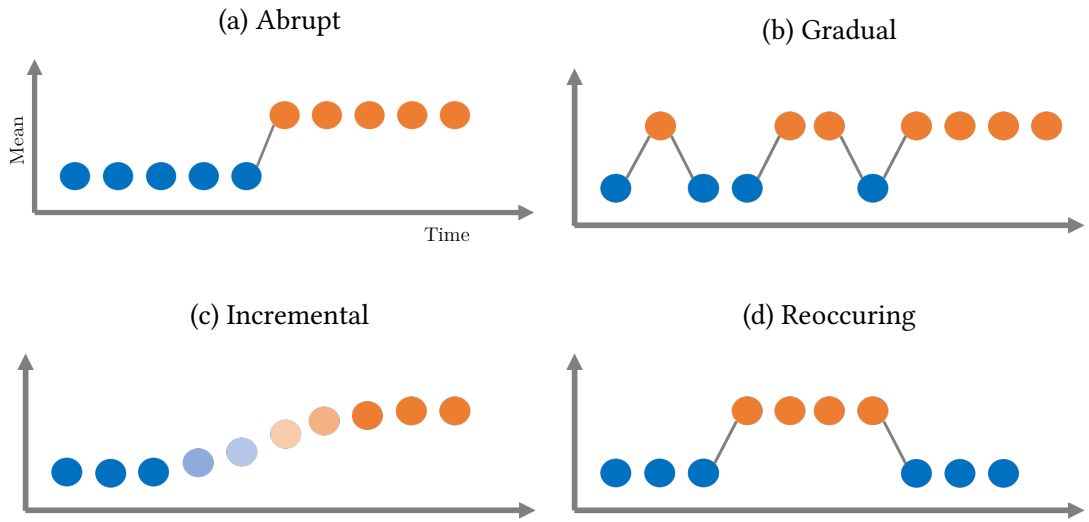


Figure 2.2.: Illustration of different types of concept drifts, adopted from [73]

- **Incremental Drift**: Incremental drift is characterized by the gradual but steady change in the underlying concept. Unlike the oscillations in gradual drift, incremental drift consistently changes toward the new concepts.
- **Reoccurring Drift**: These occur when previously encountered concepts or data distributions reappear after a period of absence. For models, it means not just adapting to new information but also remembering past states to adapt when they reoccur quickly [22].

The ability to accurately detect and adapt to these concept drifts is crucial in data stream mining. Failing to identify and adjust to these changes can severely compromise the performance and relevance of a predictive model. Adaptive techniques mostly focus on specific types of drifts. They assume concept drift types occur without the interference of others. Yet, in practice combinations of different categories often occur simultaneously [22].

2.2. Change Adaption

While identifying these drifts is paramount, the subsequent action of adapting to them is equally important to ensure the continued effectiveness of predictive models. The main challenge is to balance the consideration of historical and new information, summarized by the so-called *stability-plasticity-dilemma* [26]:

1. **Stability**: The ability of the system to retain previously learned information without it being too easily overwritten by new data. This ensures that the system doesn't forget the fundamental patterns and knowledge it has acquired.
2. **Plasticity**: The ability of the system to learn from new data and adapt to changes. This ensures that the system remains relevant and accurate in changing environments or when exposed to new information.

Change adaption techniques empower models to recalibrate, relearn, and adjust in line with the incoming data. Adaption strategies can be divided into two main approaches: Blind Methods

and Informed Methods [22]. While blind methods continually adjust to the flow of data without specific drift detection, informed methods only adapt after an identified drift.

In the following, we provides a comprehensive overview of these adaptation methods, discussing their mechanisms, strengths, and limitations.

2.2.1. Blind Methods

Blind adaptation methods, often termed passive methods or implicit adaption models, inherently assume that the nature of data streams is perpetually dynamic. Rather than actively seeking out drifts or sudden changes, these methods continuously and systematically adapt to the recent data.

Using a fixed-size window containing w labeled instances to periodically update the classifier is a widely adopted method [52]. In its most straightforward form, a fixed-size sliding window retains the latest instances for classifier training. With the flow of new data, the window updates by discarding the oldest instances and integrating the new arrivals [63]. The window's size w is a crucial hyperparameter: smaller windows better reflect the most recent data for fast adaptations, while larger ones enable richer training at the expense of slower response to concept drifts [22]. These windows grant the model a form of short-term memory, ensuring it remains attuned to the most recent data and any occurring shift, focusing on the model's *plasticity*. The underlying principle of using windowing is the assumption that recent data is the most relevant. However, this premise does not hold in all contexts. Situations with noisy data or recurring concepts challenge this assumption [22]. If reoccurring concepts persist beyond the duration of the window, the windowing approach might prove insufficient, as the long-term context is forgotten. The model learns the previous forgotten context from scratch as it reoccurs. Furthermore, to effectively address changes, the model must be retrained frequently on the window, causing a substantial overhead [74].

A special case of blind adaptation is incremental and online learning where the model evolves with data. These methods emphasize a model that is naturally responsive to changes, consistently tweaking its parameters or structure based on incoming instances. For instance, a decision tree designed for data streams as proposed by Domingos *et al.* (2000) [17], which is continuously updated with new data. As the tree expands, its new leaves capture the most up-to-date trends, reflecting the evolving nature of the data. However, one drawback of these techniques is their slow response to abrupt shifts in data patterns, mainly because they build upon previous concepts without explicit mechanisms for rapid forgetting [22].

Ensemble-based models are very popular in blind adaption methods. Their adaptability for dynamic streams is rooted in two primary attributes:

- The capacity to adapt new data, achieved by adding new classifiers to the ensemble and constantly adjusting their weights.
- The ability to selectively remove outdated information by removing old or imprecise classifiers.

For instance, a typical approach in batch-incremental setting is to construct the ensemble by training a new classifier on each incoming batch. Once a predefined maximum number of classifiers is reached, they discard one of the classifiers based on the age or performance [55]. Such ensemble setups effectively address the *stability-plasticity-dilemma* as classifiers in the ensemble represent different time horizons and combine historical data with recent concepts [16].

While blind methods eliminate the need for drift detection, their inherent adaptability comes with its own set of challenges. However, in scenarios where drifts are frequent or unpredictable, blind adaption methods can be particularly beneficial. In general, they are biased toward recent data as they assume recency equals relevance [22]. Hence, they often involve strong forgetting mechanisms by discarding old information and frequently retraining the model on recent data, focusing on the model's *plasticity*.

2.2.2. Informed Methods

In contrast to blind adaption methods, informed methods actively detect when changes occur and subsequently adapt in a more focused manner. Changes are identified by monitoring e.g. the classifier performance or change indicating properties of the data. Once a drift is detected, they adjust the model structure or retrain the predictor entirely on recent data. For instance, while blind adaption techniques often involve a fixed-size sliding window to frequently retrain the model, an informed adaption strategy would adjust the window based on the current state of the stream [73]. The generalization error might be assessed to fine-tune the window size while processing the stream [31]. In stable situations, the window size grows to enable a more comprehensively trained classifier while it shrinks for swift adaptions once drifts are detected.

The core of informed methods lies in accurate drift detection. Techniques like the Drift Detection Method (DDM) [23] or Early Drift Detection Method (EDDM) [4] have been specifically designed for data streams, focusing on monitoring error rates or other performance metrics over windows of data. Let $(x_i, y_i) \in S$ be an instance from a data stream S at timepoint i as defined in Section 2.1.2 and $\hat{y}_i = C_\theta(x_i)$ be the class prediction of a classifier C parameterized by θ. For each incoming labeled instance, the Drift Detection Method is updated with the error e_i of the i-th sample:

$$e_i = \begin{cases} 0, & \text{if } \hat{y}_i = y_i \\ 1, & \text{if } \hat{y}_i \neq y_i. \end{cases}$$

Thus, the detector derives two statistics that are constantly monitored:

1. Error Rate $p_i = \frac{\sum_{j=1}^{i} e_j}{i}$, where i is the number of instances seen so far and e_j the error at instance j.
2. Standard Deviation $s_i = \sqrt{p_i(1-p_i)/i}$ of the error rate p_i at instance i.

DDM keeps track of the minimum error rate p_{min} and minimum standard deviation s_{min} observed so far and derives a dynamic threshold. The current error rate and standard deviation are compared against this threshold. If

$$p_i + s_i >= p_{min} + \beta * s_{min},$$

a drift warning is notified and for

$$p_i + s_i >= p_{min} + \alpha * s_{min},$$

DDM detects a change, where α, β are confidence parameters, predefined by the user with $\beta < \alpha$.

EDDM, or Early Drift Detection Method, is a refinement of the Drift Detection Method (DDM) specifically designed to detect changes in data streams more effectively [4]. While DDM bases its drift detection primarily on the error rate and its standard deviation, EDDM advances this idea by focusing on the distances between classification errors. It is based on

the assumption that when a concept begins to change, not only does the error rate typically increase, but errors also tend to become more frequent. For every incorrect prediction at time point i, EDDM calculates the distance d between the current error and the previous error as $d = i - i_{previous}$, where $i_{previous}$ is the time point of the previous error. Thus, it monitors the two statistics:

1. Average Distance p' between errors.
2. Standard Deviation s' of p'.

A drift is detected when

$$\frac{p'_i + 2 * s'_i}{p'_{max} + 2 * s'_{max}} < \alpha,$$

where α is set by the user based on the desired confidence.

Once a drift is detected, one direct approach is to discard the old model entirely and retrain a new one using the most recent data [23]. This method ensures that the model is completely aligned with recent data. However, if not all data learned before the drift is detected are outdated, the model might lose valuable information from the past. Thus, some approaches strategically select instances or adjust the window sizes, depending on the current state of the stream. They aim to balance the window to be recent as well as representative, providing a reflective snapshot of the past [31] [22]. Nevertheless, complete retraining might lead to high computational complexity in frequently changing data streams.

Alternatively, the model can be partially adjusted on detected drifts. The model continues to function normally until a drift is detected. Upon detection, the model undergoes an update phase, where it may adjust its parameters, re-weigh its features, or undergo other forms of partial updating to better align with the new data distribution [30]. This strategy is more computationally efficient than full retraining but might suffer from delayed adaption to sudden drifts0 [22].

Ensemble techniques involve maintaining multiple models simultaneously. When a drift is detected, new models may be introduced, and old ones may be retired based on their performance on the new data. For instance, Adaptive Random Forest (ARF) [25] combine Hoeffding Trees (a type of decision tree suitable for data streams) and change detectors. The detectors monitor the performance of the trees in the ensemble. Once they detect drifts, trees are removed and replaced to adapt to the changes. In general, informed ensemble methods typically combine base classifiers with drift detection mechanisms to guide their adaptation processes. When drift is detected, base classifiers can be weighted, replaced, or pruned to keep the ensemble current with the evolving concept [32]. Since only parts of the ensemble are updated or discarded, some classifiers retain older data, representing concepts of different time-horizons. This is beneficial in situations where past concepts reoccur or when drifts are cyclic.

Regarding the *stability-plasticity-dilemma*, informed adaption methods, by their nature, lean towards both stability and plasticity. They retain their model structure until a drift is detected. Upon detection, they make informed decisions on adaptation, balancing between the learned historical knowledge and the newly observed concept.

2.3. Active Learning

In traditional supervised learning, models are trained using a labeled dataset, where every instance is associated with a known output. It heavily relies on substantial amounts of labeled

data, which can be time-consuming and expensive to obtain. In contrast, semi-supervised learning attempts to leverage both labeled and unlabeled data to improve learning accuracy, making it cost-effective but at the potential risk of incorporating noise or biases from the unlabeled data. Active learning, strikes a balance between these approaches. It strategically selects specific instances from available data to be labeled, aiming to maximize the learning efficiency [58]. Typically, active learning proceeds as follows:

1. The active learner identifies specific unlabeled instances.
2. The learner queries the candidates to an *oracle*, which provides the according label.
3. The learner trains on the annotated instances.

The upper limit of allowed oracle queries is defined as the Labeling Budget b. Its determination depends on the specific costs per query, data characteristics, and the desired accuracy of the model [49]. Generally, b is predefined by the user as a proportion relative to the total volume of available data e.g. $b \in [0, 1]$. Active learning techniques can vary significantly depending on the specific input scenario and the implementation of its core components. These components can be categorized into three main pillars, namely:

1. Data Management,
2. Labeling Strategy,
3. Learning Approach [57].

In this section, we provide an overview of possible input scenarios and explore the characteristics of AL components.

2.3.1. Input Scenario

A popular categorization of AL scenarios is to distinguish between *membership query synthesis*, *selective sampling*, and *pool-based active learning* [49].

1. Membership Query Synthesis: This scenario involves the model generating instances itself and then requesting labeling from the oracle. The model actively selects and presents these synthesized instances to the oracle for annotation. This approach empowers the model to explore areas of uncertainty and complexity, guiding its learning process effectively.
2. Pool-Based Active Learning: In this scenario, instances are drawn either from a known data distribution or provided as a large set of data. The entire set or significant partitions is processed at once, and a selected subset is queried for labeling.
3. Selective Sampling: The model processes each instance one by one, drawing individual points from the distribution or incrementally scanning the dataset.

While this categorization is not consistent in the literature, they mostly assume prior knowledge of the data distribution or instance space, assuming a stationary relationship between instances and their corresponding labels. Thus, these traditional AL scenarios may not be suitable for nonstationary data stream environments, where the data distribution and label relationships may change over time. Therefore, to better address the challenges posed by non-stationary data streams, we categorize AL input scenarios into two main types: Pool-Based Active Learning and Data Stream Active Learning:

1. Pool-Based Active Learning: The whole instance space or a big part of it is available a priori. This includes available data sets or known data distributions that can be drawn arbitrarily.

2. Data stream Active Learning: Each instance arrives at a time. The underlying data distribution is unknown and the feature-target relationship as well as the underlying data distribution might change over time.

In a pool-based setting, the assessment of instances can rely on comparisons and relationships within the dataset. Moreover, there is no need to consider unexpected changes since the data is already available. On the other hand, in the datastream scenario, labeling decisions are constrained to the properties of arriving instances and information stored from the past. This poses additional challenges due to the infinite nature of streams and limited memory resources.

In this thesis, we focus on the Data Stream Active Learning scenario, where we tackle the challenges of learning from continuously evolving data streams with limited resources for labeling. However, many modern approaches are based on ideas from pool-based active learning techniques, transferred to dynamic drifting stream environments.

2.3.2. Data Management

Implementing labeling strategies and active learning frameworks in data streams requires making decisions regarding the retention of historical information. The choice of data management approach significantly influences the active learning procedure, as the amount and nature of available data lead to distinct instance selection methods. We categorize data management approaches into two types: Batch-Incremental and Instance-Incremental techniques.

2.3.2.1. Batch-Incremental

While active learning and semi-supervised learning in stationary pool-based scenarios have been extensively explored with established efficient techniques, directly applying them in non-stationary data stream environments is challenging. Cohn *et al.* (1994) [14] established a Batch-Incremental approach where the data stream is divided into batches, and each batch is processed individually. This allows the use of established stationary pool-based active learning strategies. For instance, comparison based labeling strategies can be leveraged. This provides a more sophisticated instance selection method compared to solely relying on properties of a single incoming labeling candidate [57]. The common approach in batch-incremental active learning is to process each batch as a whole, applying a utility metric to quantify the usefulness of each candidate instance. This utility assessment might also consider relationships between the instances within the batch. Instances in the batch are ranked based on their utility and the top n candidates are selected for labeling. The labels per batch n is typically $b * n_B$ where n_B is the batch size and b the labeling budget.

However, the batch-incremental approach violates Bife's data stream requirement **B1** we introduced in Section 2.1.2 since multiple candidates are buffered and inspected multiple times during the utility calculation. Depending on the specific implementation and batch size, maintaining the instance buffer may lead to additional memory requirements. Yet, while processing and comparing multiple instances at once may introduce increased time complexity compared to a simple threshold comparison, querying multiple instances simultaneously can be advantageous when multiple parallel oracles are available [57].

2.3.2.2. Instance-Incremental

Another data management approach is to process each incoming instance individually and make immediate decisions on whether to label the candidate instance. Zliobaite *et al.* (2014) [74] introduced the first Instance-Incremental learning approach, where they specifically focused on not storing historical data. In contrast to the ranking procedure used in Batch-Incremental techniques, the label decision in Instance-Incremental methods relies on comparing a utility measurement of candidates to some threshold. This approach is often combined with incremental learning classifier, ensuring high levels of time and memory efficiency as no training data has to be stored for training.

However, instance-incremental techniques have major drawbacks - the lack of context for query decisions and adaptation to changes in the data behavior. Unlike batch-incremental approaches, which can consider multiple recent instances and use candidate ranking procedures based on the current stream state, instance-incremental methods have to constantly adjust their thresholds. Moreover, since instance-incremental methods mainly rely on incrementally trained classifiers, they need to actively adapt their models as they might otherwise rely on outdated concepts.

In summary, selecting the data management approach significantly impacts the quality of obtained labels and overall training efficiency. Batch-Incremental models offer a more sophisticated instance selection process by allowing the application of pool-based comparison techniques [57]. However, regarding time- and memory-efficiency, Instance-Incremental data management appears to be better suited to meet the requirements for data streams. However, to successfully apply such techniques in non-stationary environments, additional change adaptation techniques must be employed.

2.3.3. Labeling Strategy

The labeling strategy, also known as the query strategy, is the core of active learning. It drives the model's decision-making process to select instances for labeling. The instance identification approach is the major distinction between AL implementations. It directly impacts the model's learning trajectory, accuracy and capability to react to concept drifts. A variety of techniques have been explored in the past. However, all these approaches share a common characteristic: they employ metrics to assess the utility of data points for training. This section provides an overview of different utility metrics, exploiting varying properties of features in order to assess their usefulness for training. They can be roughly grouped into information-based and representation-based labeling strategies [57].

2.3.3.1. Information Based

These techniques aim to select the most informative instances for the given classification problem. More precisely, they seek to identify instances around the decision boundary since they provide the most value for learning the class prediction task [70]. In practice, this property can be exploited by assessing the confidence of the model's prediction regarding a specific instance and its target class [19].

Uncertainty sampling is the simplest and widely used approach. It derives a utility metric from the class posterior distribution $P_\theta(y|X_i)$ given as output of the used model, parameterized by θ, on the corresponding instance X_i. A popular implementation is the margin posterior probability which can be defined as

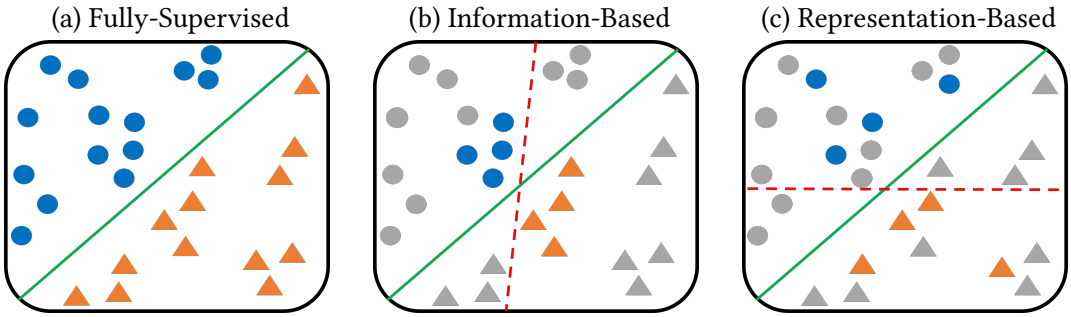


Figure 2.3.: Learned decision boundaries labeled by different strategies

$$utility(X_i) = margin_\theta(X_i) = p_\theta(y_1|X_i) - p_\theta(y_2|X_i),$$

where y_1 is the most likely class for X_i, predicted by the classifier and y_2 the second most likely [48]. This provides an assessment of the informativeness of the instance as this utility will be high if the model is uncertain about the class of the processed instance.

There are further approaches for assessing the model confidence on an instance prediction. Exemplary techniques are:

- Query by Committee, introduced by Seung *et. al* (1992) [51], trains multiple classifiers on subsets of the data and the utility is derived on the level of disagreement of the ensemble on a given instance. Thus, areas of uncertainty are identified and instances from those regions are assigned a high utility.

- Expected Model Change, by Settles *et. al* (2008)[50], identifies the instances that have the greatest impact on the model if trained with it. The utility is derived by considering e.g. the expected gradient length [69], or the expected weight change [59].

Information-based strategies all seek to measure how confident the model is on the instance class prediction. It approach provides an efficient way to rapidly generate good-performing models with small budgets as crucial areas where presumably the decision boundary traverses are labeled. Furthermore, the utility of an instance can be derived on the spot, based on the single instance itself, thus fulfilling the first requirement from Bifet. Using a utility threshold comparison, this strategy can be easily employed in data stream environments.

However, solely focusing on the model uncertainty has respective drawbacks, particularly in non-stationary stream scenarios and when using a fixed threshold. An incorrect set threshold can lead to rapidly exceeding the labeling budget or the model learns enough to rarely be uncertain enough to satisfy the condition [74]. Furthermore, the information-based strategy has a bias towards instances close to the decision boundary, neglecting substantial portions of the input space and the general data spread [19]. This may result in redundantly labeling similar points, wasting budget, and leading to incomplete modeling of the classification problem. We illustrate this in Figure 2.3b. The classifier's decision boundary (red dotted line), learned after processing the colorized instances and their labels, deviates significantly from the true boundary (green line) due to labeling points sorely on their informativeness. This can result in misclassifications of both classes.

Besides the redundant labeling, overdapation towards areas of uncertainty might lead to delayed reactions to sudden drifts that occur apart from the decision boundary [52]. The model's confidence in predictions away from the decision boundary is high, as the true class of

these instances is mostly unambiguous. However, the drifts might affect these areas, potentially changing the dominant class there. In order to learn the new concept, labels in this area have to be provided by the labeling strategy. As Information-Based strategies focus on providing training data where the model is uncertain, learning these changes can be delayed or missed entirely. Until the strategy obtains labels in the changed area, class predictions are based on the old concepts, and the confidence will not change as the model receives no feedback.

2.3.3.2. Representation Based

Representative labeling strategies aim to capture the general structure of the input space by identifying the most representative data points. Instances with strong representative properties are assigned high utility which facilitates an exploration of the input space. This becomes particularly advantageous at the beginning of the learning process and during concept drifts, as labels in regions of change are more likely to be obtained [57][28]. There exist different notions of representativeness:

- Density Based Utility: A common way to measure representativeness is to consider the similarity to other instances in the input space. This can be accomplished using a similarity metric, such as a Mercer kernel on the features. Wu *et. al* (2006) [64] presented sampling techniques for active learning, where representativeness is the average similarity of an instance to other points in the input space. Given a set of unlabeled data points $X = \{x_1, \ldots, x_n\}$, the utility of an instance x_i can be defined as:

$$utility(x_i) = representativeness(x_i) = \frac{\sum_{i \neq j} K(x_i, x_j)}{n - 1}.$$

  .

- Diversity Based Utility: This technique addresses the issue of redundant labeling that may arise when using informative strategies. It aims to provide a versatile exploration of the input space in order to generate a robust model. In most implementations, a set of instances S is gathered incrementally and presented to the oracle for the next labeling iteration. The utility of an instance x_i is calculated based on the already collected set of queried data. Using a similarity Kernel operation it can be derived as:

$$utility(x_i) = diversity(x_i) = 1 - max_{s_j \in S} \frac{K(x_i, s_j)}{\sqrt{K(x_i, x_i) K(s_j, s_j)}}.$$

  This approach minimizes the redundancy of the learning process [64].

- Clustering Based Utility: This approach leverages clustering to guide the labeling decisions. Initially, a clustering algorithm is applied to a subset of the data space, and subsequently, a selection technique is employed. Clustering enables exploration of different parts of the input space while utilizing cluster properties to support instance selection. Ienco *et. al* (2013) [28] introduced an approach that combines diversity and density by iterating over clusters and forcing a selection of instances from each cluster. The instance selection within each cluster considers the distance of the points to their centroids, effectively considering both diversity and density, resulting in a strategic and versatile learning process.

In contrast to information-based strategies, representative utility measures necessitate comparisons with other elements of the input space. As a consequence, to apply such techniques in data stream scenarios, instances have to be temporarily stored, which may violate Bifet's requirements **B1** and **B3**. Additionally, since the utility calculation is solely based on the structural properties of the data, no consideration regarding the classification problem is taken. While this tends to generate more robust models, capturing uncertain areas may require more queries, leading to a slower learning speed [57]. Furthermore, with small labeling budgets, the true decision boundary might be missed since the crucial areas are not prioritized. An illustration can be seen in Figure 2.3c, where the modeled class decision boundary deviates from the true boundary significantly due to unlearned instances at the boundary.

2.3.3.3. Combined Strategies

In practice, active learning approaches rarely rely solely on representative labeling strategies due to their slow convergence in learning curves. On the other hand, some of the early published methods use information-based query strategies exclusively. However, as discussed earlier, these techniques suffer from certain drawbacks. Modern approaches now tend to leverage a combination of information- and representation-based labeling techniques to strike a balance between exploitation and exploration. The main selection criteria usually remain information-based, while supportive representation assessment techniques are employed to counteract overadaptation. For instance, Krempl *et al.* (2014) [37] introduced a utility measure called "probabilistic gain", which quantifies the expected improvement in model performance upon acquiring the label for a candidate instance. This measure incorporates both the model's uncertainty and the distribution of already acquired labels in the neighborhood of the candidate instance. Additionally, the probabilistic gain is weighted by the density of labeled and unlabeled instances in the candidate's location. Thus, this approach can be broadly categorized as a combination of the Expected Model Change and Density-Based utility functions, leveraging the advantages of both informative and representation-based techniques.

2.3.4. Online Label Budgeting

Data stream poses significant challenges regarding the management of the labeling budget due to the potentially infinite and initially unknown volume of instances. If a batch-incremental training approach is implemented, pool-based budget management can be effectively applied to a data stream scenario by choosing an exact proportion b from each buffered batch [60][39][28]. However, when an instance-incremental approach is employed, achieving a balance within the labeling budget becomes significantly more complex. Here, labeling decisions are made instantaneously on each incoming instance, and there is limited access to the labeling history. A simplistic approach would involve maintaining a count, u, of assigned labels from the start of the process. The label spending at any time step t could then be computed as $b = u/t$. However, over an infinite time, this method is impractical as each label's contribution to the label spending would progressively diminish [74]. An alternative strategy involves maintaining a window of the last w instances. Here, the spending would be calculated as $b = u/w$, where u is the number of oracle queries within the last w instances. Despite its benefits, this strategy requires the storage of not just a counter but also each label decision within the timespan. Zliobaite *et al.* (2014) [74] proposed a more efficient strategy for balancing the label budget in instance-incremental settings, in which they approximate the current label spending by estimating the probability of labeling and derive an approximation of how many labels might be assigned to the past w instances.

Algorithm 1: Utility threshold adjustment, adopted from [74]

Input: Instance x_i, utility function $uncertainty_\theta()$, threshold adjustment λ
Output: $labeling \in \{True, False\}$

$u_i \leftarrow uncertainty_\theta(x_i)$
if $u_i < \delta_i$ **then**
    $\delta_{i+1} \leftarrow \delta_i * (1 - \lambda)$ `/* Reduce threshold */`
    return $labeling \leftarrow True$
else
    $\delta_{i+1} \leftarrow \delta_i * (1 + \lambda)$ `/* Increase threshold */`
    return $labeling \leftarrow False$

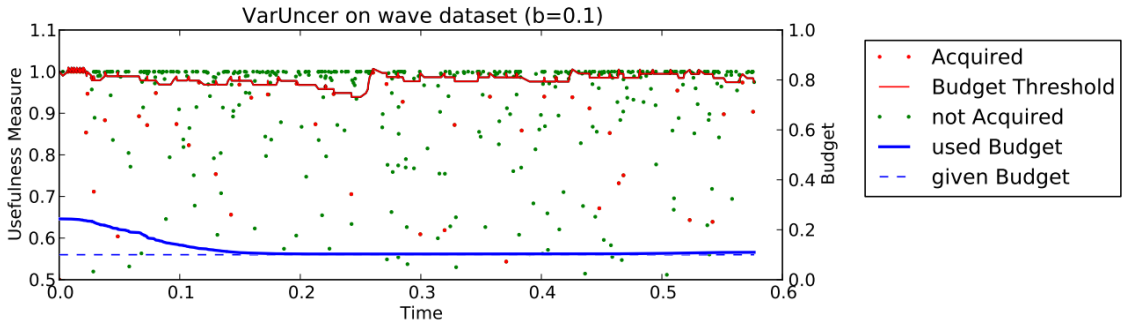


Figure 2.4.: Usage of labeling budget $b = 0.1$ with estimated budget technique from Zliobaite *et al.* 2014 [74]. Figure from [34]

2.3.5. Change Adaption in Active Learning

In Section 2.1.3 we introduced the challenges of changing concepts in data stream scenarios. Adapting to these concept drifts in an active learning context introduces unique challenges that are not necessarily present in fully supervised scenarios. The active learning mechanism must be aware of changes to ensure that the queried instances remain representative of the current concept.

For instance, information-based labeling strategies primarily use the classification model's predictions to assess the value of labeling a candidate instance by measuring the model's confidence and comparing it to some utility threshold. As the data stream evolves, and concept drifts occur, the model's confidence continually changes, potentially leading to a lack of acquired labels during periods of critical change. Consequently, utility thresholds need continual adjustments.

Zliobaite *et al.* 2014 [74] proposed an adaptable labeling strategy wherein the utility threshold is continuously adjusted based on previous labeling decisions. A high-level illustration is shown in Algorithm 1. If the model's uncertainty towards the sample x_i falls below the current threshold δ, the true label is queried and the threshold is decreased using a set step size λ. Conversely, if the uncertainty is above the threshold, no label is fetched, and the threshold increases. In essence, when the classifier accurately approximates the decision boundary within the labeling area, it displays high confidence and expands the labeling region. On the other hand, if it struggles to model the decision boundary, resulting in low confidence, the labeling area is narrowed [74]. It is a flexible and straightforward adaptive approach to the labeling process, theoretically allowing for any arbitrary utility metric to be employed.

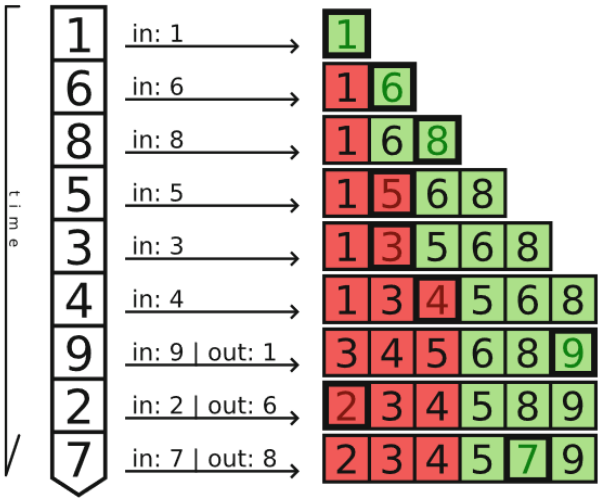


Figure 2.5.: Balanced Incremental Quantile Filter with $b = 0.5$ and $w = 6$, from [37]

Even though the authors claim that only instances with the highest utility get labeled, Kottke et al. (2014) [34] demonstrated that this is not necessarily the case. Using Algorithm 1, they analyzed the development of the utility threshold over time with a 10% budget ($b = 0.1$), as can be seen in Figure 2.4. Here, the model's uncertainty serves as a measure of utility. Consequently, instances with the lowest utility hold the most value and should be prioritized for labeling. The variable threshold δ_i, is represented as a red line, and any candidates that fall below this line meet the condition for labeling. As can be observed, instead of focusing on candidates with the highest uncertainty, the strategy only excludes some of the most certain instances, assigning labels once the budget allows it rather than basing the decision on utility. Clearly, the threshold isn't stringent enough, causing the strategy to miss out on labeling the most valuable candidates because the budget is expended on instances with certain labels.

An alternate approach was suggested to maintain balance in the budget and adjust the utility threshold, called the Balanced Incremental Quantile Filter (BIQF) [37]. Given the predefined budget $b \in [0, 1]$, last w utility values $\{u_i, \ldots, u_{i+w}\}$, from an arbitrary utility function $utility_\theta(x_i) = u_i$, are stored in a queue Q, sorted by the utility values. The labeling decision for a candidate x_i depends on the position $rank_{u_i}$ of its utility value u_i in Q and the given budget b. More precisely, the true label for an incoming instance x_i is queried, if

$$rank_{u_i} <= \lceil Q * b \rceil.$$

This technique is illustrated in Figure 2.5 for a budget $b = 0.5$ and a window size $w = 6$. New utility values that are added to the green side of the queue obtain labels. This method serves as an alternative threshold comparison approach as the threshold δ_i can be derived as the utility value of the smallest labeled candidate in the queue (most left green value). However, in order to ensure that the budget is not underutilized or exceeded due to changing utilities, they introduced a balancing technique based on the value range in the queue Δ and a predefined tolerance window w_{tol}. By tracking the number of instances processed n_{inc} and the number of labels acquired n_{acq} so far, the number of label acquisitions left can be derived as

$$n_{left} = n_{inc} * b - n_{acq}.$$

Let Δ be the difference between the highest and lowest ranked utilities in the queue as

$$\Delta = Q[w-1] - Q[0],$$

then the adjusted utility threshold can be derived as

$$\delta_{bal} = \delta - \Delta * \frac{n_{left}}{w_{tol}}.$$

As Δ and n_{acq} continuously change while the stream progresses, the threshold is constantly adjusted based on the current utility value range and past acquisition counts. As the current state of utilities in the stream is considered, the threshold ensures labeling of the currently most important instances. The amount of adaptation depends on the predefined tolerance window w_{tol}, defined by the user. The greater the balancing window size is, the less restrictive the balancing will force the threshold value to its requested budget [35].

2.3.6. Evaluation in Stream Active Learning

Stationary performance evaluation techniques to assess model quality are often not applicable or relevant in data streams. The traditional hold-out test set paradigm, commonly used in batch learning scenarios, is not expedient in continuous real-time streaming classification. In the realm of data stream classification, *prequential evaluation*, also called *test-then-train paradigm*, has emerged as a prevalent method [22]. In this approach, each data instance is first used to test the model's prediction accuracy and is subsequently used for training, thereby updating the model. This ensures that every data instance in the stream is used both for testing and training, but importantly, testing always precedes the training for any given instance, ensuring the model's predictions are always made on unseen data.

For a more robust evaluation, it's also beneficial to utilize random partitions of the stream. This allows to evaluate the model's performance on different segments of the stream, offering insights into its consistency and adaptability across varied data distributions. Randomly partitioning the stream ensures that the model encounters different data characteristics, patterns, and potential drifts in each segment, simulating real-world unpredictability[8].

In the context of active learning, the evaluation is further nuanced. Active learning inherently involves making strategic choices about which instances to label, based on a given labeling budget. Consequently, it's vital to evaluate active learning performance across different budgets. By considering multiple budget scenarios, we can understand the model's efficiency at various resource levels. This is especially pertinent given that labeling can be expensive or time-consuming in many real-world scenarios. Thus, understanding the trade-offs and benefits of different labeling budgets provides insights into how approaches might perform in resource-constrained situations. Furthermore, it helps in tuning the active learning strategy for optimal performance under different resource allocations [49].

In essence, evaluating active learning in streaming contexts involves understanding not only the classifier's accuracy but also the value derived from the chosen instances to label. This combined assessment helps in quantifying a comprehensive overview of the model's capabilities and the effectiveness of the active learning strategy employed.

2.4. Online Clustering: CluStream

Clustering is a fundamental task in data analysis and machine learning, aiming to partition a dataset into subsets, such that data points in the same cluster are more similar to each

other than to those in different clusters. Traditional clustering algorithms were designed for static datasets where the entire data is available for processing. In data streams, it's impractical to store all incoming data, making traditional clustering techniques unsuitable. The possibility of concept drifts further complicates the clustering process [12]. Thus, online clustering algorithms tailored for data streams were designed to efficiently apply clustering in dynamic streaming scenarios. One of the most recognized approaches for online clustering in data streams is CluStream [1]. Unlike traditional clustering methods, CluStream focuses on dynamically maintaining cluster features in real-time, while deferring the actual cluster formation process to an offline phase. This two-step approach enables CluStream to handle high-speed data streams efficiently and adapt to changes in the data distribution over time.

2.4.1. Cluster Representation

CluStream stores clusters as *Micro-Clusters* (MC), a summary of statistics called CF (Cluster Feature), which is maintained for each cluster over time. The CF-Vector in CluStream is adopted from BIRCH (1996) [68], extended with temporal information. The CF in CluStream is defined as:

$$CF = (LS_x, SS_x, LS_t, SS_t, N),$$

where:

- $LS_x = \sum x_i$ represents the linear sum of the data points
- $SS_x = \sum x_i^2$ represents the squared sum of the data points
- $LS_t = \sum t_i$ represents the linear sum of the time stamps of the data points
- $SS_t = \sum t_i^2$ represents the squared sum of the time stamps of the data points
- N represents the number of data points added to the cluster

This ensures that CluStream doesn't need to retain individual data points, leading to significant storage savings. The CF's are additive, meaning that the cluster's characteristics can be derived by simply adding (or, in some cases, subtracting) features of individual data points or other clusters, ensuring an efficient process of adding and merging as the stream progresses. The time information is critical for the clustering process, as it allows the algorithm to assign higher importance to recent data. This time-decay function enables the algorithm to adapt to changes in the data stream over time.

At the core of CluStream's storage methodology is the concept of the *Pyramidal Time Frame*. Instead of storing information about every micro-cluster at every time instance (which would be highly inefficient), CluStream maintains summaries at varying time granularities. This allows CluStream to take *snapshots* of the clustering state at different intervals. For instance, the highest level of the pyramid might store the state every hour, the next level every minute, and so forth. This multi-resolution model enables a compact representation of cluster states over time. This is the offline phase of CluStream, where the micro-clusters formed in the online phase are further clustered to form macro-clusters, offering an overview of data patterns. When a user desires to obtain the clustering state for a specific past time, CluStream can quickly derive this state as macro cluster from the stored snapshots without needing to reprocess past data.

Algorithm 2: CluStream online clustering

Input: Instance x_i, Clustering C, radius factor r, max cluster M

1 $MC_{cls} \leftarrow closestCluster(x_i)$
2 $d_{cls} \leftarrow distance(x_i, MC_{cls})$
3 $r_{cls} \leftarrow radius(MC_{cls})$
4 **if** $d < r_{cls} * r$ **then**
5 $\quad MC_{cls}.add(x_i)$ /* Add to closest cluster */
6 **else**
7 $\quad N_{cluster} \leftarrow length(C)$
8 $\quad$ **if** $N_{cluster} = M$ **then**
9 $\quad\quad C.mergeOrDeleteCluster()$ /* Maximum clusters reached */
10 $\quad C.create(x_i)$ /* Create new cluster with x_i */
11 **Function** `mergeOrDeleteCluster()`
12 $\quad c \leftarrow leastRelevantCluster(C)$ /* Find least relevant cluster */
13 $\quad \delta \leftarrow C.relevanceThreshold$
14 $\quad$ **if** $c.relevance < \delta$ **then**
15 $\quad\quad C.delete(c)$
16 $\quad$ **else**
17 $\quad\quad c_1, c_2 \leftarrow findClosestClusters(C)$ /* Cluster with minimal distance */
18 $\quad\quad C.mergeCluster(c_1, c_2)$ /* Merge c_2 in c_1 */
19 $\quad\quad C.delete(c_2)$

2.4.2. Online Clustering Process

The online micro-clustering phase of CluStream is the algorithm component responsible for maintaining the summary statistics of data points from a data stream. From the CF-vector, valuable cluster characteristics can be derived:

- **Center**: It is the centroid of the micro-cluster and is given by $center = LS_x/N$
- **Radius**: Is a measure of the spread of the data points around the center, defined as $radius = \sqrt{(SS_x/N) - (LS_x/N)^2}$.

These are used to assign data points to the closest clusters and potentially merge nearby clusters. We illustrate the online clustering procedure in Algorithm 2. Let C be a set of existing micro-clusters, and x_i the incoming data point. For each incoming data point x_i, CluStream finds the closest micro-cluster based on the euclidean distance between the instance and the centroid $center$ of the micro-cluster as $MC_{cls} = argmin_j(distance(c_j, x_i))$ (line 1). If

$$distance(x_i, MC_{cls}) < r_{cls} * r,$$

where r_{cls} is the radius of the closest cluster and r is a predefined tolerance parameter, the point is added to the cluster MC_{cls}, and the CF is updated accordingly (line 5). However, if this condition is not true, a new cluster is created that contains only point (x_i) (line 10). If the maximum number of allowed clusters is reached, CluStream proceeds as follows (lines 11-19):

1. Deletion: The relevance of a cluster can be quantified by assessing the temporal information (LS_t) and (SS_t) in the CF's. If the relevance of a cluster is below a user defined threshold, the cluster is deleted.

2. Merging: If no cluster is irrelevant enough to be deleted, two clusters are merged. The clusters with the smallest distance of their centroids to each other are selected.

3. Creation: After either deletion or merging, a new cluster is created, containing the current data point x_i

The micro-clusters are continuously updated with incoming data, which allows them to capture the evolving trends and patterns in the data stream. As the data stream progresses and its underlying distribution changes, the micro-clusters naturally adapt to reflect these changes. In this way, the online clustering phase of CluStream inherently adapts virtual changes in the data stream.

3. Related Work

The domain of active learning in data streams has witnessed a variety of methodologies, aimed to overcome the challenges and dynamics of streaming data. While clear categorizations aren't universally agreed upon, we roughly group previous efforts into window-based approaches, ensemble strategies, clustering-based techniques, and incremental frameworks. Window-based techniques base their approaches on maintaining fixed-sized windows of selected instances for model training or labeling decisions. In contrast, ensemble-based approaches use a multitude of classifiers, each potentially capturing unique characteristics of the data which offers a promising solution to the stability-plasticity dilemma. Clustering can provide valuable insight into the spatial characteristics of streams. This can be leveraged to balance labels strategically over the feature space. Incremental training offers promising stability towards noise. However, in order to effectively address potential drifts, they can be combined with informed adaption mechanisms. In this chapter, we provide an overview of previous approaches for active learning in datastreams. Table 3.1 gives a summary of relevant approaches we discuss in this chapter. It includes their model structure, labeling strategies, and adaption approach. After discussing these techniques, we will identify existing limitations in their change adaptation capabilities. In response, we refine our requirements for adaptive active learning algorithms in data streams from Chapter 1.

3.1. Window Based Approaches

Historically, many early active learning methods for data streams employed fixed-size windows of selected instances. The primary objective was to periodically retrain models or support the labeling decisions using these windows. These approaches didn't directly address concept drift adaptation, instead, they focused on developing novel labeling strategies that balance exploration and exploitation within the constraints of the training window.

Lindstrom *et al.* (2010) [39] proposed a window-based active learning technique, based on uncertainty sampling. They employed a Support Vector Machine (SVM) as classification model to conduct real-time class predictions and to determine the utility of an instance on its proximity to the SVM margin. In their setup, the data stream is grouped into batches. With each new batch, a predefined number of instances with the highest utility (highest proximity) are labeled and integrated into a fixed-sized sliding window. Concurrently, the model undergoes retraining with every fresh batch. This method exemplifies how pool-based active learning strategies that are traditionally applied in static environments, can be adapted to a stream setting through batch processing. However, it suffers from high computational demand, redundant labeling, and overadaption [19][74]. The computational drawback is induced by the frequent retraining and the latter two are commonly present when using purely information-based labeling strategies as we discussed in Section 2.3.3.

Krempl *et al.* (2014) [37] aimed to balance exploration and exploitation for the labeling process to avoid the drawbacks of pure uncertainty sampling. They developed a Probabilistic Active Learning (PAL) technique and later adapted it for data stream contexts [35]. Their technique leverages both model uncertainty and the surrounding label frequency to assess

Table 3.1.: Comparison of related approaches

Approaches	Data Management	Model	Model Updates	Labeling strategy	Budget Management	Change Detection	Change Adaption
Lindstrom [39]	Batch	Singular	Training on Window	Uncertainty sampling		-	Fraction of Batch
Krempl [35]	Incremental	Singular	Training on Window	Probabilistic gain	BIQF [35]	-	Blind retraining
Liu [40]	Incremental	Singular	Incremental updates	Local Density + uncertainty sampling	Spending estimate [74]	-	-
Zhu [71]	Batch	Ensemble	Training on batches	Uncertainty sampling	Fraction of Batch	-	Blind model forgetting
Scholz [47]	Batch	Ensemble	Training on batches	Fully supervised	-	-	Informed model forgetting
Bifet [6]	Incremental	Ensemble	Incremental updates	Fully supervised	-	ADWIN	Informed model adjustments
Ienco [28]	Batch	Singular	Incremental updates	Clustering based: local density + uncertainty	Fraction of Batch	-	-
Yin [67]	Incremental	Cluster ensemble	Incremental updates	Uncertainty sampling + randomization	-	-	Informed model adjustments
Zliobaite [74]	Incremental	Dual	Incremental updates	Uncertainty sampling + randomization	Spending estimate [74]	DDM	Informed model adjustments
Xu [65]	Incremental	Dual	Incremental updates	Uncertainty sampling + randomization	Spending estimate [74]	Ensemble dissent	Informed model replacements
Wang [60]	Batch	Singular + Ensemble	Incremental updates	Uncertainty sampling + randomization	Spending estimate [74]	-	-

the utility of labeling candidates. A Parzen Window Classifier [13] is used to calculate the label frequency in the neighborhood for each incoming candidate. Thus, an assessment of the candidate's representation value is incorporated among the model's uncertainty to ensure a stratified distribution of labels. This utility is assigned to every incoming instance and subsequently fed into the BIQF, which we introduced in Section 2.3.5. The instance, independent of the labeling decision, is injected in a fixed-size window, and the model is retrained on the window in every time step.

An alternative approach to balance exploration and exploitation was proposed by Liu *et al.* (2023) [40]. The window guides the labeling decisions rather than serving as a basis for model retraining. Instead of the typical First-In-First-Out window maintenance method, they introduced a unique forgetting mechanism inspired by the forgetting curve observed in human memory cognition [18]. They assign instances within the window a measurement called 'memory strength' based on two factors: their time of arrival and their distance from other samples in the window. Using this window structure, a local density assessment of an incoming instance can be calulated. Thus, the derived labeling strategy is a dual-query procedure, involving the local density and the uncertainty of a candidate. Initially, they compute the local density based on the cognition window, and add the incoming instance accordingly. Should this density exceed a predetermined threshold, the candidate's uncertainty is assessed and compared to a separate threshold. If both conditions are met, the strategy queries the instance and the model is incrementally trained. Their experiments showed that this combination of incremental training and representative labeling strategy enables the model to resist noise and adapt to gradual or incremental drifts effectively. This robustness is attributed to the model's retention of older information, while the labeling strategy encourages exploration across different feature space regions. However, it stumbles in the face of abrupt drifts due to the absence of a proactive forgetting mechanism, potentially resulting in lagged responses to rapid concept shifts [40].

In sum, recalling the *stability-plasticity-dilemma* discussed in Section 2.2, these window-based techniques often prioritize one facet over the other. While some exhibit strong plasticity by frequently updating models on windows [39] [35], others lean towards stability, evading any forgetting mechanisms entirely [40].

3.2. Ensemble Approaches

Classifier ensembles can naturally balance stability and plasticity as they provide a flexible structure of multiple models. The classifier in the ensemble might capture specific areas in the feature space or different time horizons of historical data. Thus, concept drifts might only affect a part of the ensemble, and forgetting mechanisms can be applied to the concerning classifiers while the rest stays intact and retains valuable information for stability.

Zhu *et al.* (2007) [71] proposed an ensemble active learning approach, training multiple classifiers on different stream batches. For each incoming batch, their strategy labels instances based on the ensemble prediction variance. A new classifier is trained on the selected instances and added to the ensemble. If the ensemble reaches a maximum size, the oldest is removed. We illustrate this procedure in Figure 3.1. To make class predictions, the ensemble employs a weighted decision-making approach. The weights assigned to each classifier in the ensemble are continuously updated through gradient descent, aiming to minimize the overall variance in the ensemble's predictions. This setup ensures that classifiers trained on data from varying time horizons contribute to current predictions. The maximum allowable size of the ensemble

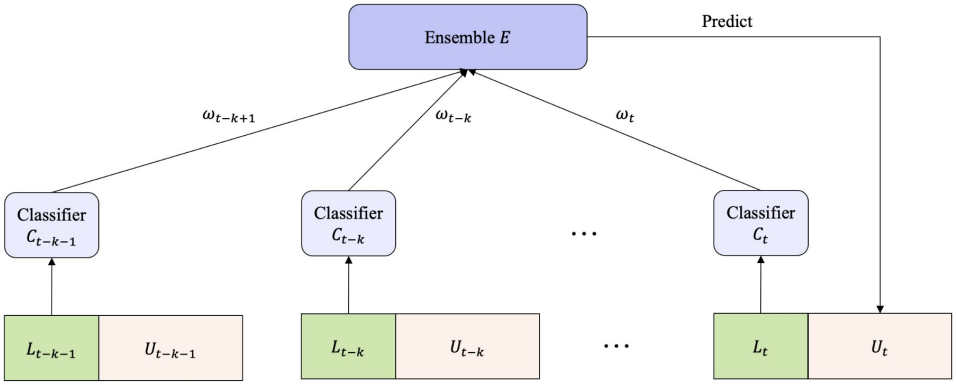


Figure 3.1.: Structure of the classifier ensemble approach from [11]

is a critical parameter; it determines the extent to which past data should influence current stream state predictions and thus needs to be selected with care. Such block-based ensemble methods can efficiently adapt to gradual concept drift but lack a response to the concept drift that occurs within the block. If there's a sudden shift, classifiers grounded on older data blocks before the shift, retain their significance. This can cause the ensemble's reaction to be delayed. The size of the data block plays a crucial role in this. While smaller blocks can swiftly address abrupt changes, they compromise the ensemble's performance during stable periods and elevate computational demand [52].

Instead of blindly training a new classifier on each batch, Scholz *et al.* (2005) [47] implemented an online ensemble technique where the update of the ensemble structure depends on the current state of the stream. Though originally developed for fully supervised situations, the concept of modifying the model structure according to the current state can be adapted for active learning. When faced with a new batch, they decide to either train a fresh classifier or update the latest from the ensemble. They train both classifiers (latest and fresh) with the incoming batch and compare their accuracies. The more efficient classifier of the two is retained. This approach avoids the issue of generating poor classifiers due to minimal batch sizes, as pre-existing classifiers can learn data from multiple batches. Individual classifiers are assigned weights based on their recent training performance. These weights serve a dual purpose: accelerating adjustments to concept shifts and discarding redundant classifiers. However, the approach struggles when faced with swift reoccurring concept drifts, primarily because the method's forgetting mechanism discards valuable past data [36].

An alternative approach to dynamically change the model structure based on the current stream state is to combine an ensemble classifier with explicit drift detection techniques. Bifet *et al.* (2012) [6] proposed an ensemble consisting of multiple Hoeffding Tree classifiers, incrementally trained with each incoming instance. They use the probability estimates of the trees to train the weights of perceptron classifiers. Each tree is assigned a drift detector which monitors the prediction accuracy. If the detectors identify a significant change in accuracy for a tree, they reset its learning rate, allowing the perceptrons to adapt faster. Such instance-incremental ensemble approaches are able to learn the data stream in one pass, potentially being faster and requiring less memory than batch-incremental approaches. Modifying the ensemble architecture based on a detected drift ensures that in stationary states, the model can proceed with learning the current concept without unnecessarily forgetting intact information. However, failing to detect drifts results in no adaption while false alarms can lead to unnecessary adjustments [36].

By maintaining multiple classifiers, ensembles can effectively capture a diverse range of feature space characteristics and assimilate data from varied temporal points. While certain parts of the ensemble may be influenced by concept drifts, selective adaptation, and partial forgetting can be applied to maintain valuable information while adapting to changing regions.

3.3. Clustering Based Approaches

Clustering-based active learning approaches aim to dynamically group data from streams into coherent subsets. This can naturally reflect the dynamic nature of the data and provide valuable insight into spatial characteristics of the current data distribution. The maintained clustering structure can be used as a classification model or to enhance representative labeling decisions.

Ienco *et al.* (2013) [28] leveraged stationary K-Means clustering [43] for a balanced label selection process. They divide the data stream into batches and apply the standard K-Means algorithm on each incoming set. Based on the clustering, a two-step label identification process is applied, combining geometrical information of the data point and the model uncertainty. Firstly, they rank the clusters on their homogeneity, derived from the predicted class distribution within the cluster. High entropy indicates interesting areas for training. a "micro-step" quantifies the utilities of candidates inside the clusters, by considering the distance to the centroid as well as the models uncertainty. The highest-ranked instance from the best cluster will be labeled first. Iteratively, an instance from every other cluster is labeled to ensure all areas of the input space are covered. This proceeds until the fixed labeling budget for the batch is exceeded. The classifier is incrementally updated with the labeled instances while processing each batch. By iterating over each cluster in the labeling process, the whole data space is considered in a stratified manner. Thus, the approach avoids overfitting and changes anywhere in the input space can be learned. However, this two-step procedure might elevate the required computation time as the batch has to be processed multiple times for clustering and labeling, thereby violating Bifet's requirement **B1**. Furthermore, the clustering is only leveraged to balance the labeling over the feature space, even though it contains additional valuable information that can be used to strengthen the predictions.

In a recent publication by Yin *et al.* (2023) [67], a data stream active learning framework was proposed, that incrementally updates a clustering structure and uses its information for class predictions. The clustering algorithm is a modified version of the online CluStream algorithm [1]. In contrast to the previous technique, clustering forms the classification model instead of supporting the labeling strategy. A simple hybrid label query strategy is used, combining an uncertainty threshold comparison and a random sampling. Class predictions are made as follows: Upon the arrival of a new, unlabeled instance, the model searches for the closest k microclusters to this instance. Each microcluster's weight is computed, based on the label homogeneity and distance to the instance. The final prediction for the instance is assigned based on the class label found most frequently within the highest-weighted microcluster. Updating the clustering depends on the corretness of the predictions. If the prediction was correct, the weights of all microcluster participating in the prediction are incremented. Furthermore, if the instance is close enough, it is added to the microclusters. Otherwise, with every incorrect prediction, the weights are decremented and a new cluster containing that instance is created. Upon reaching the maximum size, clusters are merged or deleted. We provide an overview of this procedure in Figure 3.2. Deletion happens once the weight of a cluster decreases below a predefined threshold, due to bad prediction performance or over time by an implemented

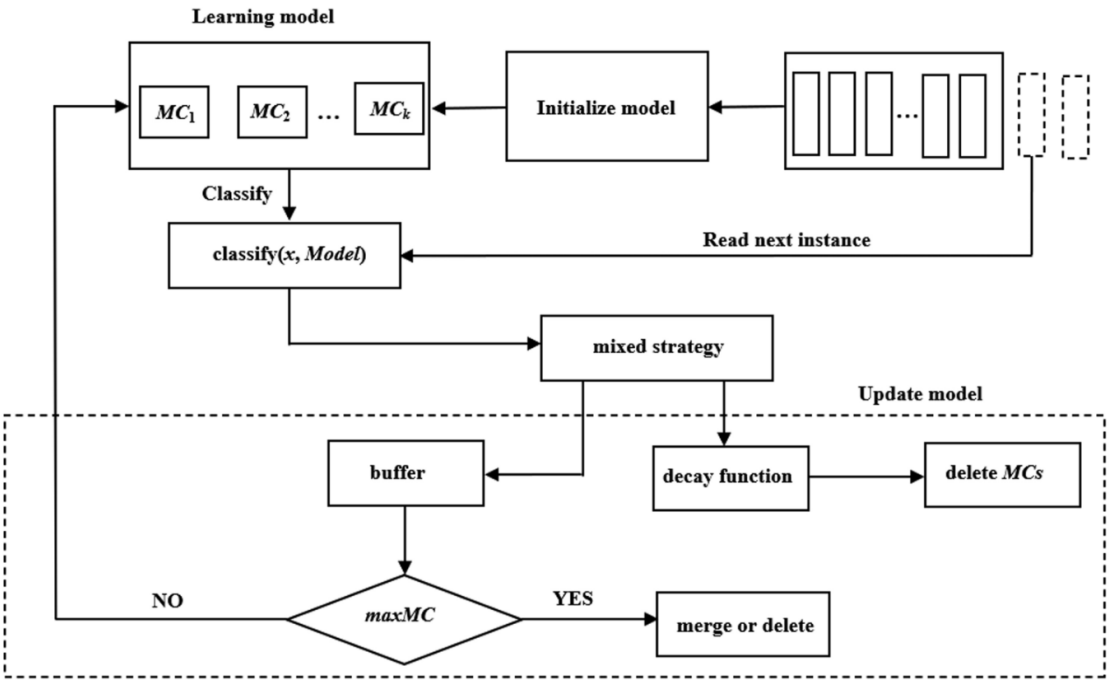


Figure 3.2.: Clustering-based active learning approach from [67]

weight decay mechanism. This provides selective forgetting of outdated or bad-performing clusters while maintaining the majority of the model. However, even though they use active learning to identify their training data, no labeling budget restrictions were considered. In theory, if the data constantly remains under the uncertainty threshold, the approach transitions to a fully supervised learning technique. Furthermore, as the clustering is only updated with labeled instances, restricting the number of labels significantly, might limit the clustering structure and performance of this technique. As we focus on restricted labeling budget active learning, this technique is not necessarily in the scope of this thesis.

Nonetheless, the idea of dividing the model into different spatial areas of the input space with clustering can be leveraged to strike a balance in the *stablility-plasticity-dilemma*, as it naturally assesses the locality of changes. Thus, models trained on the change-affected areas can be updated while unaffected remain.

3.4. Adaptive Incremental Frameworks

Several approaches in the literature use incremental training, including references [40],[28], and [6]. Experimental results demonstrated that incremental training, when combined with a balanced labeling strategy, exhibits stability towards noise and gradual concept drifts. Furthermore, they typically process each instance only once, fulfilling requirement **B1**. Previous approaches of that category rarely implement direct adaption mechanisms but rather focus on balanced labeling. Yet, some frameworks incorporate explicit change detection modules to actively identify changes and proceed with adjustment mechanisms. For instance, in the fully supervised approach by Bifet *et al.* (2012)[6], the learning rates of ensemble members are reset once a drift is indicated. However, abrupt drifts with a certain significance require a more radical adaption mechanism to be effectively addressed [22].

Algorithm 3: Generic adaptive active learning framework from [74]

Input: stream S, labeling budget B, Labeling Strategy $strategy()$
Output: at every time step classifier output $\hat{y}_i$
Initialize: label spending $\hat{b} \leftarrow 0$

```
1  while S not ended do
2      x_i ← nextSample(S)
3      ŷ_i ← predictClass_L(x_i)
4      if b̂ < B then
5          if strategy(x_i) then
6              y_i ← requestLabel(x_i)
7              updateBudget(b̂)
8              updateDDM(ŷ_i == y_i)
9              train_L(x_i, y_i)                    /* train stable classifier L */
10             if R exists then
11                 train_R(x_i, y_i)                /* train reactive classifier R */
12             else if drift warning then
13                 R ← createClassifier(x_i, y_i)   /* create R */
14             if drift detected then
15                 L ← R                            /* swap classifier */
16                 delete(R)                        /* delete reactive classifier */
```

Zliobaite *et al.* (2014) [74] introduced a widely-recognized adaption framework for incremental active learners working on data streams. It combines incremental active learning with the explicit drift detection module DDM. A singular classifier, termed the stable classifier L, undergoes continuous updates based on new labels and handles class predictions. Concurrently, the DDM monitors the prediction error rate. This module can distinguish between early drift warnings and definitive drift detections, a distinction based on the magnitude of the observed change. Depending on the drift's nature, the framework alters its learning and predictive models. This process is outlined in Algorithm 3. The labeling budget is controlled by an estimation of recent budget spending $\hat{b}$, as presented in Section 2.3.4. For each new instance, the stable classifier L offers a class prediction (line 3). The algorithm checks whether the budget is not exceeded and if the candidate should be labeled according to the labeling strategy (lines 4-5). If the true class y_i is acquired, the budget spending $\hat{b}$, the stable classifier L and the drift detector are updated (lines 6-9). If a second reactive classifier R exists, it is trained with the current instance as well (line 11). This reactive classifier takes over as the stable classifier once a drift is confirmed (lines 15-16). By distinguishing between drift warning and detection, adaption is only applied in uncertain states of the stream and the new model is not built from scratch, but already retains recent information. They proposed three effective labeling strategies that use uncertainty sampling and different combinations with randomization. The efficiency of these strategies oscillates based on the drift types encountered. In theory, any threshold-based labeling strategy can be implemented within the framework. However, a possible limitation is the singular incremental learning classifier. This could mean reduced model diversity, potentially leading to overadaption of older instances. with the incremental learning of a single classifier, the lack of diversity of classification models may lead to overadaptation of the older instances. Thus, the model's response to sudden drifts is delayed, potentially affecting learning precision [52].

Algorithm 4: Simplified Paired-Ensemble Framework from [65]

Input: stream S, labeling budget B, detection threshold θ, random sampling threshold δ
Output: at every time step stable classifier output $\hat{y}_L i$
Initialize: label spending $\hat{b} \leftarrow 0$

```
1  while S not ended do
2      x_i ← nextSample(S)
3      η ← uniform random variable η ∈ [0, 1]
4      if b̂ < B then
5          if η < δ then
6              labeling = randomStrategy(x_i)                    /* random sampling */
7          else
8              labeling = uncertaintyStrategy(x_i)               /* uncertainty sampling */
9          if labeling then
10             ŷ_Li ← predictClass_L(x_i)                        /* Prediction by L */
11             ŷ_Ri ← predictClass_R(x_i)                        /* Prediction by R */
12             if (ŷ_Li ≠ y_i)&(ŷ_Ri = y_i) then
13                 driftCount ← driftCount + 1
14             else
15                 driftCount ← driftCount − 1
16             if driftCount > θ then
17                 L ← R
18                 R ← createClassifier()
19                 driftCount = 0
20             train_L(x_i, y_i)                                 /* train stable classifier L */
21             if η < δ then
22                 train_R(x_i, y_i)                             /* train reactive classifier R */
23
```

Xu *et al.*(2016) [65] adopted this pairwise classifier framework and the idea of combining uncertainty sampling with randomization. In contrast to Zliobaite *et al.*(2014) [74], they do not implement an explicit drift detection module but rather compare the predictions of both classifiers as indicators for drifts. When the reactive classifier consistently outperforms the stable one over a certain duration, the latter gets substituted by the former. Furthermore, the two classifiers aren't trained by the same labeling strategy. The stable classifier employs a hybrid of uncertainty and randomization in its labeling, while the reactive model's training samples are determined purely by random selection, ensuring a uniformly distributed learning trajectory. We provide a simplified version of this procedure in Algorithm 4. For each incoming instance, a random variable dictates whether to apply uncertainty sampling or random sampling (lines 5-8). If a label is given, both classifiers perform their prediction on the instance (lines 10-11). If the reactive classifier's judgment aligns with the actual label but the stable's doesn't, a drift counter ticks up. This counter monitors such discrepancies within a set time frame. Upon hitting a designated limit, the reactive classifier becomes the stable, and the counter is reset to its initial state (lines 16-20). In essence, the current stable classifier learns every labeled instance while the reactive model is exclusively trained with labels selected by the random strategy. Thus, the reactive classifier learns scattered across the entire data space, while the stable classifier's knowledge primarily clusters around decision boundaries. However,

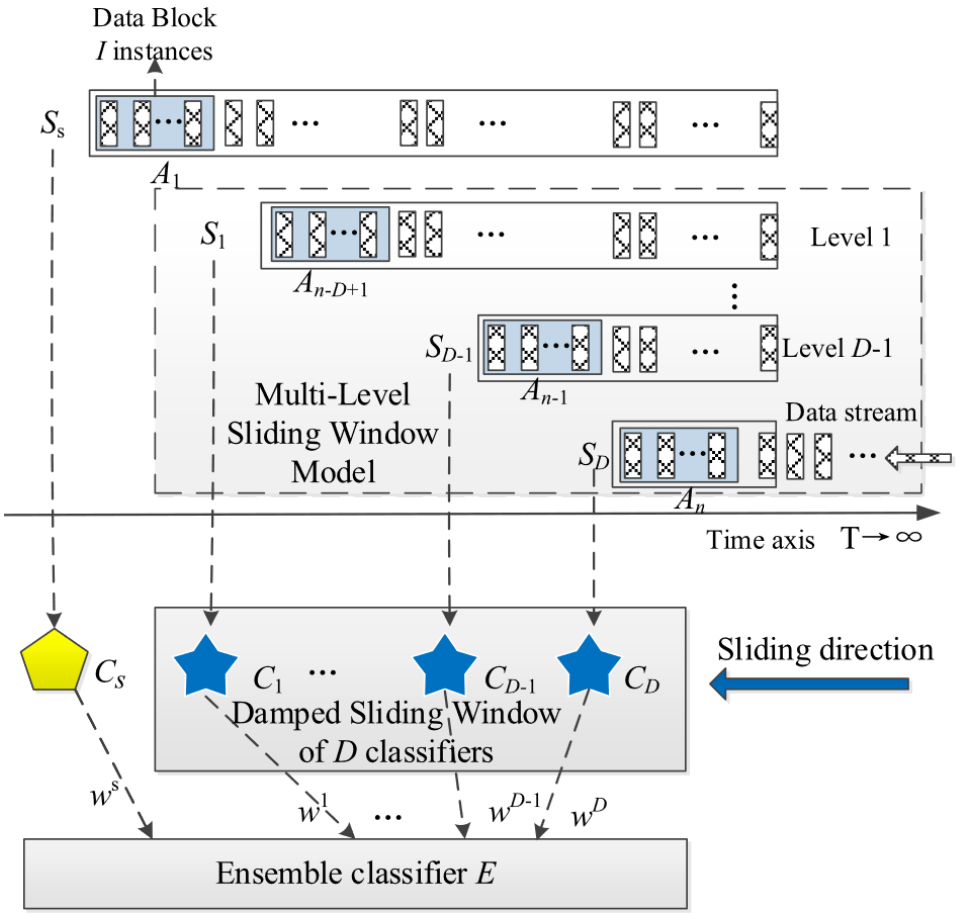


Figure 3.3.: Multilevel-Sliding-Window approach from [52]

on swapping the models, the paired ensemble loses the memory of instance distributions outside of the reactive model's scope. This diminishes the new stable classifier's capability to recognize slow, incremental changes or gradual drifts, as the history of its information is limited. Consequently, the system has limitations in handling gradual drifts efficiently and might overcompensate when faced with abrupt changes [52].

In 2019, Shan *et al.* [52] further extended the adaption framework from Zliobaite *et al.* [74]. In contrast to previous versions of the framework, their stable classifier remains and is constantly updated as the stream progresses to capture long-term contexts and gradual changes. The single reactive classifier is transformed into multiple dynamic classifiers, trained on varying time horizons. The ensemble is shaped by a multilevel sliding window model to monitor data changes at varying near-future time intervals. An illustration of this framework can be observed in Figure 3.3. The framework introduces a multilevel window to handle different types of concept drifts, overcoming the limitations of a single window size. It creates a new dynamic classifier for each new data block and updates this classifier with the next $D-1$ data blocks. Once the ensemble reaches its maximum capacity (D dynamic classifiers), any new dynamic classifier created will cause the oldest one to be discarded. Thus, the newest classifier inside the ensemble only learned the most recent data block while the oldest was trained on the last $D-1$ blocks, capturing recent data trends or potential sudden drifts in different time intervals. The stable classifier is maintained throughout and learns from all the data blocks that have arrived. Its purpose is to keep track of long-term trends and gradual drifts. In contrast to the previous two incremental frameworks, this approach does not incorporate drift detection.

3.5. Adapting to Changes

We presented various change adaptation approaches, all offering both benefits and drawbacks, especially regarding the *stability-plasticity-dilemma*. Incremental training, for instance, provides commendable stability against slowly developing drifts and noise, ensuring that models maintain their core structures to effectively learn gradual changes. When this method is paired with balanced label strategies, even drifts that occur at the far edges of data distributions are accounted for [52][28][35]. However, they might fail to swiftly react to sudden changes as the model retains outdated information.

For data streams characterized by frequent and sudden changes, radical forgetting mechanisms emerge as a potent tool [11]. These mechanisms, by design, allow models to swiftly "forget" older trends in favor of new ones. Among these, blind forgetting mechanisms, which periodically retrain their models, seem particularly suited for environments where rapid adaptation is crucial. Yet, these mechanisms require additional computational depend to effectively address drifts as the periodic retraining needs to be frequent [74]. Furthermore, their effectiveness can be limited in active learning due to the challenges of operating within a sparse label environment.

Explicit change detection is an effective technique to induce minimal forgetting, activating it only when a substantial change is perceived. However, many current active learning techniques, upon detecting change, are prone to almost entirely discarding their models [74][65]. Such an approach, while ensuring adaptivity, can discard valuable insights learned from previous data streams, impairing their stability towards noise and gradual drifts.

Ensemble methods offer valuable features for partially applying forgetting mechanisms while retaining historic and still-relevant information. Paired with informed adjustments, they show promise, as these combinations have already been explored in supervised scenarios [6][47]. The granularity of data stream ensembles is mostly based on training each classifier on different time horizons.

Perhaps the most present limitation across all these approaches is the missing assessment of spatial bounds of changes. Changes, as we know, can be localized, manifesting strongly in specific areas while leaving others unaffected. While the affected regions may necessitate a reevaluation of older data, the rest remains invaluable, especially data from unaffected regions of the decision boundary. Current methods, often neglect this spatial differentiation.

Based on the insights and limitations we identified in previous works, we further refine our in Chapter 1 introduced requirements for active learning algorithms to operate in non-stationary data streams:

R1 Employ balanced labeling strategies.

R2 Continously learn in stable stream states.

R3 Detect abrupt changes swiftly.

R4 Adjust the model by forgetting outdated training data.

R5 Preserve knowledge unaffected by concept drifts.

While **R3** and **R5** hold equal significance in all data stream contexts, **R1**, **R2** and **R4** are particularly vital in active learning due to limited availability of labeled training data. Supervised techniques obtain labels distributed over the entire feature space, thus any real concept drift will be noticed by rising error rates. Regarding **R1**, uncertainty-based labeling

Requirement	Lindstrom [39]	Krempl [35]	Liu [40]	Zhu [71]	Ienco [28]	Zliobaite [74]	Xu [65]	Shan [52]
R1 (Labeling Strategy)	-	✓	✓	-	✓	✓	✓	✓
R2 (Learn continuously)	-	-	✓	-	✓	✓	✓	✓
R3 (Detecting Changes)	-	-	-	-	-	✓	✓	-
R4 (Forgetting)	✓	✓	-	✓	-	✓	✓	-
R5 (Preserving)	-	-	✓	-	✓	-	-	✓

Table 3.2.: Requirements in related work

strategies, which are inherently biased towards the decision boundary, might overlook these outlying shifts without balanced strategies. For stability towards noise and slow-moving drifts, uncertainty labeling strategies should incorporate randomization or representation-based sampling to counteract this inherent bias. The robustness to noise also depends on the amount of learned data. Thus, **R2** ensures that learned concepts aren't unnecessarily discarded and the model can further train towards stability if no adjustment is required. In changing states of the stream, especially with significant abrupt drifts, the inherent label constraints in active learning can lead to lagging responses. Thus, forgetting mechanisms need to be more sensitive and ideally retain intact information (**R4**). A comparison of these criteria against existing methods can be seen in Table 3.2. Note that we left out the approaches by Scholz *et al.* (2005) [47], Bifet *et al.* (2012) [6] and Yin *et al.* (2023) [67] as they do not operate in the scope of budget restricted active learning.

Early publications extensively studied the limitations of pure uncertainty-based sampling when facing non-stationary data streams. Thus, besides early approaches as Lindstrom *et al.* (2010) [39] and Zhu *et al.*(2007)[71], all presented techniques implement some balancing mechanism, meeting requirement **R1**. Similar can be observed for **R2**, as incrementally trained classifiers have emerged as well-suited instruments to cope with the challenges of modern data streams. Thus, most modern approaches continuously learn during stable states of the stream. Regarding **R3**, Zliobaite *et al.* (2014) [74] and its extension by Xu *et al.* (2016) [65], representing the group of informed adaption approaches, exclusively consider to monitor the current state of the stream and apply trigger-based adaption mechanisms. Most intriguingly, no current approach satisfies both requirements **R4** and **R5**. They either omit any forgetting mechanisms [40][28][52] or profoundly discard their old models with blind periodic retraining [39][35] or informed replacement mechanisms [74][65].

Given this limitation, our contributions aim to bridge this gap. We propose a flexible, change-adaptive framework for active learning classification in non-stationary data streams. Leveraging an online clustering algorithm, we constantly maintain a spatially partitioned representation of recent streaming data. Clusters are enriched with label- and performance statistics in order to contribute to class predictions as well as indicate potential drifts. We distribute multiple change detectors across the clusters to closely monitor the precision in their respective region. Thus, we can assess the locality of potentially occurring drifts, and adjust the model cautiously. Combining this structure with active learning and an incrementally trained base classifier, we designed a framework that is able to:

- Continuously learn during stable stream states.
- Detect abrupt changes anywhere in the feature space.
- Forget outdated concepts, while,
- Preserving unaffected regions.

The framework can be implemented with any online labeling strategy and an arbitrary incremental base classifier, making it highly flexible. With its robust design and adaptability, we believe it fills a gap present in current active learning techniques for data streams.

4. CORA Framework

In the previous chapter, we explored the existing landscape of active learning in data streams and identified a key set of challenges. With these challenges as background, in this chapter we present the details of our proposed solution, the **C**lustering-based **O**nline **R**e-**A**ctive Learning Framework (CORA). We detail its architecture, algorithms, and functionalities. After providing a general framework overview, we present our online classification approach, involving an introduction to the modified clustering technique and design choices for the classification model. We then transition to present our partial adaptation strategy, introducing two distinct drift detection approaches and the following adaption mechanism.

4.1. Framework Overview

Our Framework is structured around four key components and operates within two main procedures. We illustrate the architecture of CORA in Figure 4.1. Key components:

- **Labeling Strategy**: This component strategically evaluates each incoming data instance to determine whether it should be presented to the oracle for labeling.
- **Clustering**: The clustering groups instances based on their spatial proximity. It stores statistics of unlabeled data and label distributions, as well as the most recent labeled instances. It contributes to the classification task and presents the heart of our change adaptation procedure.
- **Classification Model**: The classifier interacts with the clustering module to continuously predict the classes of each incoming instance. Based on design considerations, these predictions can be used to influence both the labeling strategy and change detection processes.
- **Change Detector**: We deploy one change detector per cluster. Each monitors clustering statistics, such as label entropy or prediction error. This allows the detection of changes in particular data regions.

The components above operate within two interleaved procedures:

1. **Online classification**: Continuously performed to predict incoming instances while simultaneously updating the classifier and the clustering. We update the aggregated cluster statistics with every data point. If the labeling strategy acquires a label for that instance, we enrich the statistics with labeling information, and the classifier learns the labeled training point.
2. **Adaption Mechanism**: Triggered by the change detector ensemble. The mechanism discards the cluster affected by the detected change and retrains the base classifier on the remaining cluster. Thus, both the classifier and the clustering forget presumably outdated information while preserving unaffected knowledge.

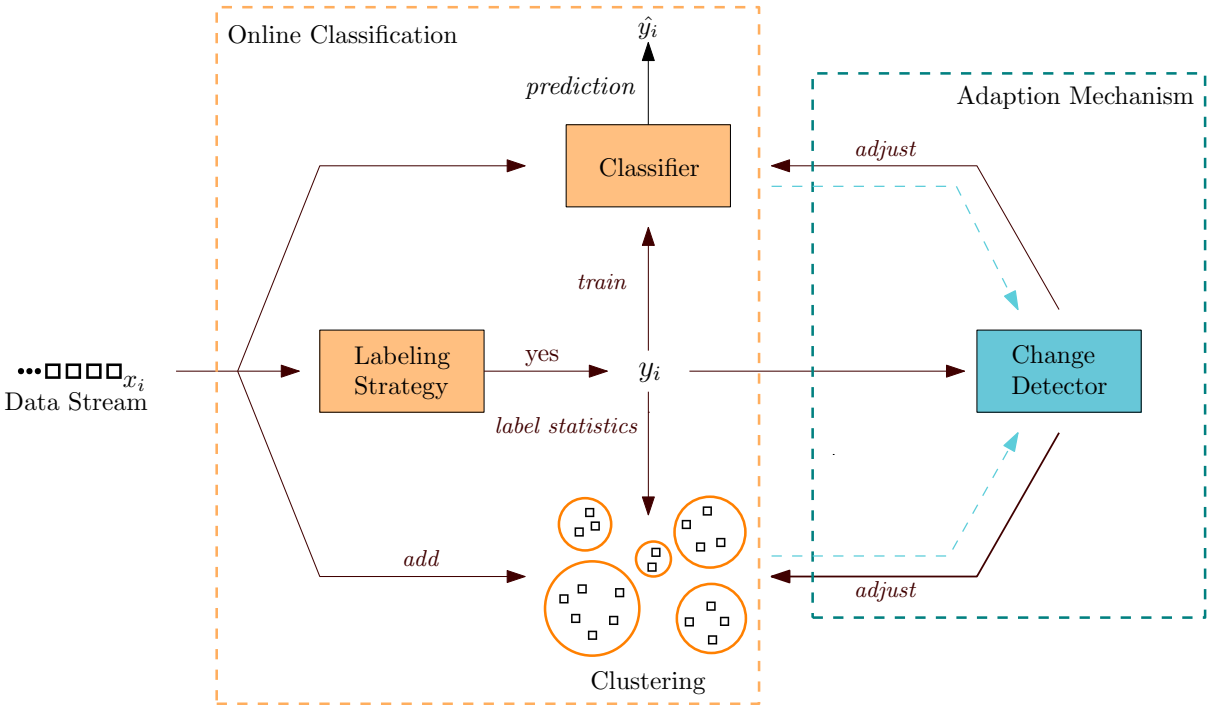


Figure 4.1.: CORA architecture

We illustrate the high-level interaction of components and procedures in Figure 4.1. We integrate each incoming instance into the clustering module to update the aggregated statistics. Concurrently, we feed the instance into the classification model and labeling strategy to obtain the class prediction and labeling decision. Depending on the instance utility, the labeling strategy presents it to the oracle to acquire its true class label. If it obtains a label, we use the annotated instance to incrementally train the classification model, and assign it to the relevant cluster. Simultaneously, each change detector in the ensemble monitors either the prediction error or the class entropy within its observation zone. The specific quantities monitored are based on design decisions, which we explore further in Section 4.3. We denote design decisions as blue dotted arrows in the illustration.

Upon detection of a change, the corresponding detector triggers the adaptation mechanism. It removes the cluster associated with the change detector and retrains the base classification model using the labeled training data retained in the remaining clusters. This selective approach ensures that the model only discards knowledge specific to a particular region of the feature space, as defined by the boundaries of the affected cluster. When a new instance arrives that does not align with existing clusters, we initialize a new cluster containing the instance, and we add a corresponding change detector to the ensemble.

In the following sections, we present the details of each of the core components and a detailed description of the adaptation routine.

4.2. Online Classification

This procedure comprises a continuous cycle of predictions, labeling decisions, model training, and the dynamic maintenance of a clustering structure. Below, we break down the components, algorithms, and potential design decisions.

4.2.1. Clustering

To effectively adapt to changing data trends, our clustering mechanism needs to be incrementally trainable and memory-efficient. We therefore adopt the CluStream algorithm as the foundation of our clustering approach, leveraging its compact cluster representation and inherent capacity to adapt to virtual drifts [1]. We introduce a modification to this algorithm by integrating labeled instances into cluster features and triggering cluster deletions based on detected concept drifts.

We define a cluster feature (CF_i), which represents a single micro-cluster (MC) at time step i, as

$$CF_i = (LS_i^x, SS_i^x, LS_i^t, SS_i^t, N_i, \boldsymbol{LI_i}, \boldsymbol{LD_i}), \tag{4.1}$$

where $(LS_i^x, SS_i^x, LS_i^t, SS_i^t, N_i)$ are aggregated statistics of the features and arrival times of assigned instances as in the original version of the algorithm. We enrich these quantities with two features:

1. $\boldsymbol{LI_i}$ is a sliding window of size w, containing tuples of the last w labeled instances that were assigned to the MC. We define LI_i as

   $$LI_i = \{(x_0, y_0, i_o), \ldots, (x_{w-1}, y_{w-1}, i_{w-1})\},$$

   where x_j represents the features of the j-th instance in the window, y_j its true class label, and i_j its arrival timestamp. We employ a First-In-First-Out policy with a fixed window size to minimize memory usage and ensure the data reflects the most recent trends in the cluster region.

2. $\boldsymbol{LD_i}$ is the class distribution of labeled instances as

   $$LD_i = [D_i^0, \ldots, D_i^{c-1}],$$

   where D_i^j indicates the sum of occurrences of the j-th class in cluster MC_i. In contrast to LI_i which is bounded to the last w instances, this is an aggregated statistic that contains all labeling decisions inside that cluster since creation.

Algorithm 5: CORA online clustering

Input: Instance x_i, Labeling Decision $label \in \{True, False\}$, Clustering C, Relevance Threshold δ, Window Size w

```
1  if label then
2      y_i ← trueLabel
3  else
4      y_i ← None
5  MC_cls ← closestMC(x_i)
6  if distance(x_i, MC_cls) < MC_cls.threshold then
7      C.add(MC_cls, x_i, y_i)                          /* Add to closest MC */
8  else
9      N_cluster ← length(C)                        /* Check for deleted cluster */
10     if N_cluster = M then
11         C.mergeOrDeleteCluster(C, δ, w)
12     C.create(x_i, y_i)                          /* Create new cluster with x_i */
   ---------------------------------------------------------------------------
13 Function add(MC, x_i, y_i):
14     MC.addFeatures(x_i)                                /* Add features to CF */
15     if y_i ≠ None then
16         LI ← MC.LI
17         LD ← MC.LD
18         LD[y_i] ← LD[y_i] + 1                    /* Update class distribution */
19         LI ← LI ∪ (x_i, y_i, i)                /* Add labeled instance to window */
20 Function mergeOrDelete(C, δ, w):
21     MC_o ← getLeastRelevantCluster(C)
22     r_o ← getRelevanceStamp(MC_o)          /* Relevance time stamp of Cluster */
23     if r_o <= δ then
24         C.delete(MC_o)                                          /* Delete */
25     else
26         MC_a, MC_b ← getClosestCluster                          /* Merge */
27         MC_a.addFeatures(MC.b)                /* Add unlabeled statistics */
28         MC_a.LD ← MC_a.LD + MC_b.LD             /* Add label distribution */
29         LI_merg ← MC_a.LI ∪ MC_b.LI              /* Add labeled instances */
30         MC_a.LI ← takeMostRecent(LI_merg, w) C.delete(MC_b)
31
```

We present the modified online clustering procedure in Algorithm 5. It specifically highlights the components that deviate from the original algorithm. For an in-depth description of the original algorithm, please refer to Section 2.4.2. The presented procedure is post-initiation, following the initial creation of all MCs. The steps are as follows:

1. Evaluation of the label decision (lines 1-4): We assess the decision made by the labeling strategy and assign the true class label y_i if positive.

2. Proximity Assessment (lines 5-6): We identify the closest cluster MC_{cls}, based on some distance measurement of x_i to the cluster centroids. In our experiments, we use euclidean distance.

3. Assignment (line 7): If the instance lies within the distance threshold of the closest cluster, we add it to MC_{cls} among its true class label y_i if available.

4. Structural Update (lines 9-12): If the condition is not fulfilled, we create a new cluster containing the new instance. Upon reaching the maximum number of clusters, defined by the user, we consolidate clusters either through a merge or by deletion based on their average time stamps. Thus, we enable the instantiation of a new cluster.

In lines 13-19 and 20-30 we highlight the two possible assignment mechanisms that are invoked depending on the proximity to the clusters. If the instance can be directly integrated into an existing micro-cluster MC, we add the feature and time stamps to the aggregated statistics (line 14). Let $CF'_{i_1} = (LS^x_{i-1}, SS^x_{i-1}, LS^t_{i-1}, SS^t_{i-1}, N_{i-1})$ be the unlabeled part of the feature vector CF at time point i. As CF' is additive, the instance can be efficiently integrated with

$$Cf'_i = CF'_{i-1} + (x_i, x_i^2, i, i^2, 1).$$

If a label for this instance was provided, we add y_i to the aggregated class distribution LD in line 18 by incrementing the corresponding class count D^{y_i} by one. Furthermore, we append a tuple, containing the instance x_i, its class label y_i and the current timestamp i to the window of labeled instances LI (line 19). Since we implement LI as FIFO sliding window, the oldest instance is automatically expelled to make room for the newer instances if it reaches the maximum size w. Notably, the instance and its label are not consolidated in an aggregated form. Instead, they are preserved in their original state as they constitute a training set, primed for potential model updates.

In a scenario where no existing cluster is in proximity to incorporate the instance, we create a new cluster, containing the current instance (lines 9-12). Before adjusting the structure of the current clustering to make room for a new cluster, we verify whether the maximum allowed number of clusters has already been reached. This consideration deviates from the original algorithm where the predefined number of clusters permanently remains in place once it is initially filled. We inspect for any available cluster slots because our approach allows for cluster deletions that occur externally to the regular clustering process, triggered by the adaptation algorithm. If no such external deletion was conducted, we free space by either merging or pruning. We highlight this procedure in lines 20-30. Firstly, we find the least relevant cluster in the current structure by assessing their aggregated time stamps (LS^t, SS^t) and deriving a relevance quantity as in the original algorithm. If the least relevant cluster is below the user defined relevance threshold δ, the according cluster is removed from the structure (lines 23-24). Else, we identify the two closest cluster based on the euclidean distance of their centroids. We merge them by adding their aggregated statistic (line 27), their label distributions LD (line 28),

and unifying the labeled instance windows (lines 29-30). The last step involves an assessment of the arrival times of included instances. As the size of LI is limited by w, we want the most recent instances in the resulting merged window to reflect the current data trends. Thus, the merged circular array LI_i results of unifying both sets, and drawing w tuples (x_j, y_j, j), with the highest time stamp j (line 29-30). Either by deletion or merging, one cluster is removed from the structure. As such, in line 12, a fresh cluster is crafted and incorporated into the existing cluster ensemble. We initialize the new cluster using the instance x_i and, where available, its class label y_i.

In summary, our modified online clustering procedure, adeptly combines adaptability and precision. Its foundational mechanism, based on the CluStream algorithm, enables the clustering process to inherently capture and adapt to virtual concept drifts in the data stream. Our addition of labeled instances and aggregated label statistics is a key enhancement. It provides the framework with the ability to detect significant concept drifts at specific locations proactively. In the following section, we present an additional possibility to leverage the enhanced clustering by incorporating the clustered labeling information into the class prediction process.

4.2.2. Classification Model

In this section, we present the integration of a classification component within our streaming framework. Our design is modular, allowing any incremental probabilistic classifier such as Naive Bayes Classifier [66] or Hoeffding Trees [17], to be used as a base model. We introduce two distinctive classification approaches: a Single Classifier Approach, and an Ensemble Approach that further leverages the clustering structure.

4.2.2.1. Single Classifier Approach

Here, the only involved classification model is an incremental base classifier M. This classifier continuously learns instances, labeled by the labeling strategy. Given an instance x_i, it outputs the associated probabilities for each class prediction $p_M(y_i, x_i)$. This constitutes the model's confidence in its prediction and is input for uncertainty-based labeling strategies. The predicted class $\hat{y}_i$ for a given instance x_i is given by:

$$\hat{y}_i = arg\,max_y\, p_M(y|x_i). \tag{4.2}$$

In this design choice, a single model produces the predictions that influence the labeling process, the change detection, and subsequently the overall framework structure. Thus, in order to provide stability towards noise and small drifts that might not be detected by our drift adaption mechanism, we suggest using a rather complex classification model that inherently adapts to current data trends such as Hoeffding Trees [17].

4.2.2.2. Ensemble Classifier Approach

In contrast, the Ensemble Approach combines the strengths of an incremental base classifier and our previously introduced clustering algorithm. We adopt the idea by Shan *et al.* (2019)[52] of using a pair of models where one is a single base classifier, learning the overall context while the other is an ensemble assembled by individually trained models. In contrast, our ensemble granularity is not based on different horizons but on spatial distinctions, using our clustering structure. Since our clustering includes labeled instances, we can form class predictions based on the class distributions within a cluster, as done with k-Nearest Neighbour Classification [15].

For each new instance x_i we identify its closest micro-cluster MC_k. We calculate the class prediction probability output from our clustering model C for a given instance x_i as:

$$p_C(y|x_i) = \frac{LI_k}{w}, \tag{4.3}$$

where DI_k is the class distribution stored in the cluster features of the closest cluster MC_k to x_i. Thus, the paired ensemble compromising the base classifier M and the clustering model C, makes its final class prediction for an instance by integrating the class probabilities from both sources. More precisely, the class probability decision of our ensemble E for an instance x_i is made using an equally weighted ensemble prediction as

$$p_E(y|x_i) = 0.5 * p_M(y|x_i) + 0.5 * p_C(y|x_i). \tag{4.4}$$

The final class prediction $\hat{y}_i$ is then determined by

$$\hat{y}_i = arg\,max_y\, p_E(y|x_i). \tag{4.5}$$

Each source contributes equally to the final prediction, ensuring a balanced decision that is informed by both detailed instance-level data and broader cluster-level patterns. While the base classifier provides the prediction using the overall data space context, the clustering assesses local patterns. This, furthermore also influences the labeling strategy as the ensemble prediction $p_E(y|x_i)$ represents the model uncertainty. By incorporating the localized information of the clustering, uncertain regions can be effectively localized and provided with the necessary labels to boost the learning trajectory in problematic regions.

This ensemble approach is an example of the possibilities of the framework. The clustering can not only be used for local change adaption, its stored labeling information can be incorporated in other parts of the framework such as the class predictions or potentially in the labeling strategy.

4.3. Adaption Mechanism

In this section, we outline our adaptation mechanism, the component that enables our framework to respond locally to changing data trends. This mechanism is activated by signals from the change detector ensemble, setting off a sequence of actions that recalibrate both the base classifier and the clustering structure. We divide the mechanism in the procedure of detecting changes, and the triggered adaption routine.

4.3.1. Change Detection

We chose the Drift Detection Method (DDM) as our change detector due to its simplicity and flexibility. Originally designed for detecting changes by monitoring error rates, it can be used to monitor any quantity where a rise in value indicates changes [23]. We distribute a DDM on every existing cluster upon initialization and evaluate two different approaches to detect changes.

We illustrate the change detection procedure in Figure 4.2. It is invoked if the label strategy queries the true class label, otherwise, we proceed with the standard online classification. If a label is available, we find the closest micro-cluster MC_k to x_i in the current clustering structure, highlighted as red boundaries in the figure. Based on the index, we identify the responsible

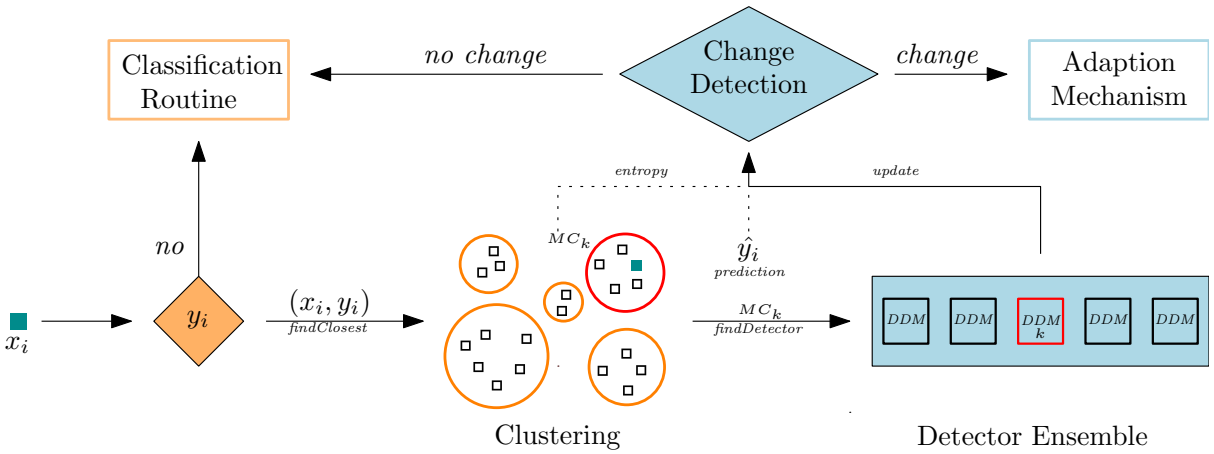


Figure 4.2.: Illustration of our change detection procedure

Drift Detection Module DDM_k from the change detector ensemble. DDM_k is updated with the designated drift indication value. We evaluate two quantities to indicate concept drifts for our framework:

1. Local Prediction Error.

2. Cluster Entropy.

We update the change detector with either of those values for the current step. If the change is significant enough, the detector signals a drift and triggers the adaption mechanism. Otherwise, the standard updates of the classifier and the clustering proceed. In the following, we present the details of the two design choices for change detection.

4.3.1.1. Local Prediction Error

Our first design decision for monitoring involves tracking local prediction errors as an indicator of potential concept drift. Increasing prediction errors within a specific cluster are interpreted as a signal that the underlying concept that this cluster encloses might be shifting. In Algorithm 6 we illustrate the prediction error based detection mechanism, after the responsible change detector is identified. Firstly, the class prediction $\hat{y}_i$ of the current classifier on x_i is obtained. In lines 2-3 we update the change detector by adding the prediction error:

$$e_i = \begin{cases} 0, & \text{if } \hat{y}_i = y_i \\ 1, & \text{if } \hat{y}_i \neq y_i. \end{cases} \tag{4.6}$$

Subsequently we check the change state of the detector. If the error rate in the area changed significantly over a period of time, the detector will trigger the adaption mechanism and inform the algorithm of the location of the drift.

4.3.1.2. Class Entropy

Our second design decision involves tracking changes in the class entropy within each cluster as a potential signal of concept drift. Class entropy is a measure of the class inhomogeneity in a dataset. A significant change in the entropy within a cluster suggests a possible alteration in

Algorithm 6: CORA: Local change detection on prediction error

Input: Labeled instance (x_i, y_i), Micro-Cluster MC_k, Classifier M
Output: Change detected $\{True, False\}$, Corresponding Cluster MC_k

```
1 ŷ_i = M.predictClass(y_i)                    /* get class prediction */
2 e_i ← ŷ_i ≠ y_i
3 DDM_k.update(e_i)
4 if DDM_k.changeDetected then
5 |   return True, MC_k
6 else
7 |   return False
```

Algorithm 7: CORA: Local change detection on class entropy

Input: Labeled instance (x_i, y_i), Micro-Cluster MC_k, Detector DDM_k
Output: Change detected $\{True, False\}$, Corresponding Cluster MC_k

```
1 LD_k ← MC_k.LD                    /* get labeled instances in cluster */
2 LD'_k ← LD_k[y_i] + 1
3 h'_k ← entropy(LD'_k)                        /* calculate new entropy */
4 DDM_k.update(h'_k)
5 if DDM_k.changeDetected then
6 |   return True, MC_k
7 else
8 |   return False
```

the class distribution of that region. Let LD_k be the distribution of labels in cluster MC_k and c the number of classes, the normalized class entropy of that cluster can be calculated as:

$$H(LD_k) = -\sum_{j=0}^{c} p_j \log_c(p_j), \tag{4.7}$$

where p_j is the proportion of labels belonging to class j in LD_k [53]. We update the change detector on every newly assigned labeled instance with the entropy H'_k of the assigned cluster MC_k. We illustrate that procedure in Algorithm 7. Regarding the input, in this design choice the indication quantity doesn't rely on the classifier prediction to update the change detectors but on the corresponding micro-cluster. We acquire the labeled distribution LD_k from the cluster (line 1). We instantiate an updated version of the current class distribution by adding the class label and calculate the new class entropy h'_k, according to Equation 4.7. Subsequently, the corresponding change detector is updated with the new class entropy, that would emerge if we add the instance to that cluster. If the entropy changes significantly, the detector will indicate a drift in that region and we return the corresponding cluster.

Since entropy is a measure that can change rapidly with incoming data and is not produced by a secondary component, monitoring it allows for potentially faster and more robust detection of drifts compared to monitoring prediction errors. This could enable more agile responses to changes in the data distribution. Furthermore, homogeneous clusters are beneficial for class predictions as the complexity of the classification problem within a cluster rises with its class entropy. Thus, homogeneous clusters elevate the effectiveness of the ensemble classifier

Algorithm 8: CORA: Adaption mechanism

Input: Changed MC index k, Classifier M, Clustering C, Detection Ensemble D, (x_i, y_i)

1 $C.remove(k)$
2 $D.remove(k)$
3 $t \leftarrow \{\}$
4 **for** MC_i ***in*** C **do**
5 $\quad t_i \leftarrow MC_i.LI$ `/* get labeled instances from every cluster */`
6 $\quad t \leftarrow t \cup t_i$ `/* gather training set */`
7 $M.retrain(t)$
8 $processInstance(x_i, y_i)$ `/* process instance as usual */`

approach. By deleting clusters once their class entropy rises significantly, we inherently increase the ability of the clustering to predict correct class labels.

4.3.2. Change Adaption

Upon detecting a change, our adaptation mechanism is activated. We present the procedure in Algorithm 8. The steps are as follows:

1. **Localization**: We obtain the index of the affected micro-cluster and its change detector. (Input)

2. **Selective Forgetting**: The micro-cluster is promptly removed from the clustering model and the change detector is deleted from the ensemble. (lines 1-2)

3. **Retraining**: After removing of the affected cluster, we unify the labeled instances from all other remaining clusters into a single training set. The base classifier is then retrained using this consolidated set of labeled instances, which were unaffected by the detected change. (lines 3-7)

4. **Assignment**: The instance that triggered the adaption is added to the updated framework according to the online classification routine (line 8).

5. **Re-initialization**: Once a new instance arrives that is not in the boundaries of the remaining clusters, a new micro-cluster is initialized containing this data point and a new detector.

Our adaptation mechanism is built with the central goal of preserving the model's capacity to learn from stable concepts, while swiftly discarding knowledge about the drifting concepts. We only discard information related to the affected cluster to avoid a complete 'reset', thereby allowing the model to retain valuable knowledge from other, stable regions of the feature space. Thus, our approach achieves quicker adaptation to local concept drifts. In environments such as active learning where labeled data is sparse, our mechanism ensures that training data is effectively used. With explicit change detection and a trigger-based partial forgetting mechanism, we avoid unnecessary model adjustments and overreactions. Hence, we strike a balance in the *stablility-plasticity dilemma* to provide reliable class predictions in stable and changing stream states.

5. Evaluation

In this chapter, we evaluate our CORA framework. First, we provide a detailed overview of the experimental setup, complemented by a selection of real-world and artificial datasets. We then investigate CORA's sensitivity, focusing specifically on its responsiveness to two pivotal parameters: the number of clusters and the detection threshold. In our core performance analyses, we benchmark CORA, in its various configurations, against three established baselines. Through this evaluation, we highlight the characteristics of CORA's change detection, the efficacy of our local adaption mechanism, and specific scenarios where it excels or requires enhancement.

5.1. Setup

This section provides a comprehensive overview of the experimental setup, detailing possible design choices under evaluation, the strategic selection of the framework's modular components, and a description of the evaluation datasets. With this setup, we aim to ensure a systematic and robust assessment of our proposed methods and their performance under various conditions.

5.1.1. Framework Configuration

We examine a variety of configurations that reflect distinct design options and our choice for interchangeable components: The base classifier and the labeling strategy. In Chapter 4 we presented that our framework offers multiple configuration alternatives, including the choice for a classifier model and the indicator for change detection. Each offers two design decisions, resulting in four possible configurations. We denote them as:

- **CORA-SP**: Single classifier and change detection on local prediction error.
- **CORA-SE**: Single classifier and change detection on class entropy.
- **CORA-EP**: Ensemble classifier and change detection on local prediction error.
- **CORA-EE**: Ensemble classifier and change detection on class entropy.

In Table 5.1 we provide an overview of the four design options under evaluatoin, including our choices of interchangeable components. We chose these components as follows:

- **Base Classifier**: We chose Hoeffding's Trees (HT) [6] as base model due to its adaptability and efficiency. The trees inherently adapt to data trends and incrementally process every incoming instance. This becomes particularly vital for configurations employing a single classifier model, where the need for stability against noise and subtle gradual drifts is accentuated. The Hoeffding's Tree's resilience to such data trends makes them generally suitable for the framework.

Config	Prediction Model	Detection Indicator	Base Classifier	Labeling Strategy
CORA-SP	**S**ingle	**P**rediction Error	HT	OPAL + BIQF
CORA-SE	**S**ingle	**E**ntropy	HT	OPAL + BIQF
CORA-EP	**E**nsemble	**P**rediction Error	HT	OPAL + BIQF
CORA-EE	**E**nsemble	**E**ntropy	HT	OPAL + BIQF

Table 5.1.: Overview of CORA configurations

- **Labeling Strategy**: For identifying instances to label, we rely on the Online Probabilistic Active Learning (OPAL) approach as labeling strategy and combine it with the Balanced Incremental Quantile Filter (BIQF) to manage the budget [35][37]. Our empirical studies have consistently shown this strategy achieves a desirable balance between exploitation and exploration. It prioritizes uncertain regions yet ensures a broad exploration of the instance space. Thus, the learning trajectory is guided efficiently and changes can be detected in every region. The BIQF employs an adaptive threshold mechanism, optimizing the usage of the labeling budget.

5.1.2. Datasets

This section describes the datasets used for our evaluation. Our collection includes four real-world datasets and four artificial data streams, each showcasing various potential concept drift characteristics and different structural properties. Table 5.2 summarizes these datasets, highlighting the number of features, classes, number of categorical features, and stream lengths. Datasets with a length denoted as Inf indicate implemented generators that can produce data streams of any desired length. Below, we provide further characteristics of each dataset, categorizing them into real-world and artificial datasets.

5.1.2.1. Real World Data Sets

These datasets, widely recognized in evaluations of data stream classification algorithms, are believed to exhibit dynamic behaviors. However, the exact nature of their concept drifts remains unknown. Certain examples are modified by value-specific sorting to infuse dynamic behaviors.

- **Electricity**: This dataset contains data from the Australian New South Wales Electricity Market, where prices are influenced by supply-demand dynamics. Each entry is distinguished by attributes such as the day, time, market demand, and more, and represents a 30-minute interval. The class label denotes the relative price change from the previous day. While this dataset, introduced by Harris *et al.* (1998) [27], is a favored benchmark for concept drift analysis, Zliobaite *et al.* (2013) [72] raised concerns about its efficacy for drift adaptation assessment.

- **Airlines**: This dataset from [29] comprises data related to the prediction of flight delays based on several input features like the source airport, destination airport, scheduled departure time, and so forth. Given the nature of airline operations, various factors such as weather, airport traffic, and changes in airline operations policies can introduce drifts over time, affecting the prediction accuracy. Thus, when the dataset is processed as a

Real World					
Dataset	**Features**	**Classes**	**Cat. Feature**	**Length**	**Drift Type**
Electricity	8	2	1	45,312	Unknown
Airlines	7	2	2	539,383	Unknown
Covertype	54	7	44	581,012	Unknown
Pokerhand	10	10	5	829,201	Unknown
Artificial					
Dataset	**Features**	**Classes**	**Cat. Feature**	**Length**	**Drift Type**
Hyperplane	2	2	0	Inf	Increm. + Global
SEA	3	2	0	100,000	Abrupt + Global
RBF	2	15	0	Inf	Abrupt + Local
Chessboard	2	8	0	200,000	Abrupt + Global

Table 5.2.: Real-world and artificial datasets for evaluation

stream, it emulates a realistic environment where conditions can change, necessitating models that can adapt to such changes. Notably, this dataset contains 2 categorical attributes with roughly 200 possible values each.

- **Covertype**: This dataset associates 30 x 30 meter land segments with specific forest cover types. It leverages cartographic attributes like elevation, slope, and soil type in order to predict the individual type of forest cover. The focus is on undisturbed forests, ensuring the cover types mirror natural ecological patterns. This dataset has been frequently utilized in drift algorithm evaluations [74][21][46].

- **Pokerhand**: This dataset describes a variety of poker hands. Each record comprises five cards from a standard 52-card deck, specified by the card's suit and rank. The classification represents the hand type, such as a pair or full house. While the original set lacks inherent concept drifts, we adopt the modified variant proposed by Bifet *et al.* (2013) [5], where drifts are incorporated by sorting instances based on rank and suit.

5.1.2.2. Artifical Data Sets

While real-world data streams showcase performances in real-world applications there's a need for controlled experimentation. Artificial data streams can synthetically generate different type of concept drifts in a controlled manner. Thus, we can control the type, magnitude, frequency, and nature of drift, in order to evaluate adaption capabilities and robustness. We evaluate our approach on the following artificial data sets:

- **Hyperplane**: We use the Hyperplane Generator from the Scikit-Multiflow Library [44] to generate a 2-dimensional stream where a hyperplane acts as decision boundary between two classes. Each instance is generated and labeled depending on its position in relation to the hyperplane. It introduces the drift by subtly altering the position and orientation of the hyperplane over time. Thus, an incremental drift of the whole decision boundary is induced.

- **RBF Generator**: Based on the generator described by Losing *et al.* (2016) [41], fifteen Radial Basis Functions (RBF) with randomized covariance matrices populate a two-dimensional feature space $[0, 1]$. Each RBF corresponds to a unique class. Samples are drawn from a randomly chosen RBF at every step. After every 2000 samples, a subset of the RBFs interchange, emulating local abrupt drifts. This setup ensures only a randomly choisen fraction (2-14) of the RBFs change, while the rest remain consistent.

- **Chessboard**: Another dataset from Losing *et al.* (2016) [41] divides the data space into squares, with each square representing a distinct concept. Drift is simulated by sequentially uncovering these squares every 1000 instances. After unveiling four squares, samples spanning the entire chessboard are displayed, penalizing algorithms that prematurely discard earlier concepts

- **SEA**: Introduced by Nick *et al.* (2001) [56], the SEA (Streaming Ensemble Algorithm) dataset is a renowned artificial data stream benchmark. It comprises three numeric attributes, of which only two are relevant for classification. Data points are uniformly distributed over the data space. The classification boundary is defined by the equation $x_1 + x_2 = b$, where x_1, x_2 are the relevant attributes. Abrupt drift is induced by changing b every 12500 samples. In addition, the dataset also contains a 10% noise.

5.2. Sensitivity Analysis

In this section, we assess our framework's responsiveness to key hyperparameters: the drift detection threshold δ and the cluster count n_c. Incorrect threshold settings can either inhibit adaptation or cause overadaption, both compromising the ability to accurately predict the current concept. Furthermore, the choice of clusters plays a crucial role, as it influences the granularity of our adaption mechanisms and the efficiency of our predictions. We'll examine how the framework performs under various parameter settings and assess the prediction accuracy across all four CORA configurations. Recognizing the significant influence of label availability, our sensitivity analysis comprises three labeling budgets ($b = 0.1, b = 0.4, b = 0.7$), from which we derive the average accuracy. Additionally, we analyze performance over random stream partitions containing 5000 instances, averaged over 30 repetitions.

5.2.1. Detector Thresholds

The detection threshold balances correct and timely adaptation against avoiding unnecessary overreaction. A properly set threshold ensures the system can detect serious drifts in the data stream. Simultaneously it guards against false alarms that could emerge from present noise or minor drifts. In our study, represented in Figure 5.1, we explore average accuracies across thresholds within the range $\delta \in [0.1, 3]$, tested on two datasets and three varying cluster counts $n_c \in \{3, 10, 20\}$. We evaluate the sensitivities on the Hyperplane data stream (upper row) and the RBF generator (lower row). The former encapsulates global incremental drifts, while the latter is characterized by local abrupt shifts. Notably, the sensitivity of CORA on the RBF Generator is significantly than on the Hyperplane stream. For instance, with 3 clusters, accuracy varies between 30% to 90% on the RBF's, while the Hyperplane reveals a steadier performance, deviating by merely 10%. This discrepancy arises from the higher complexity of the RBF generator, given its 15 classes compared to Hyperplane's binary

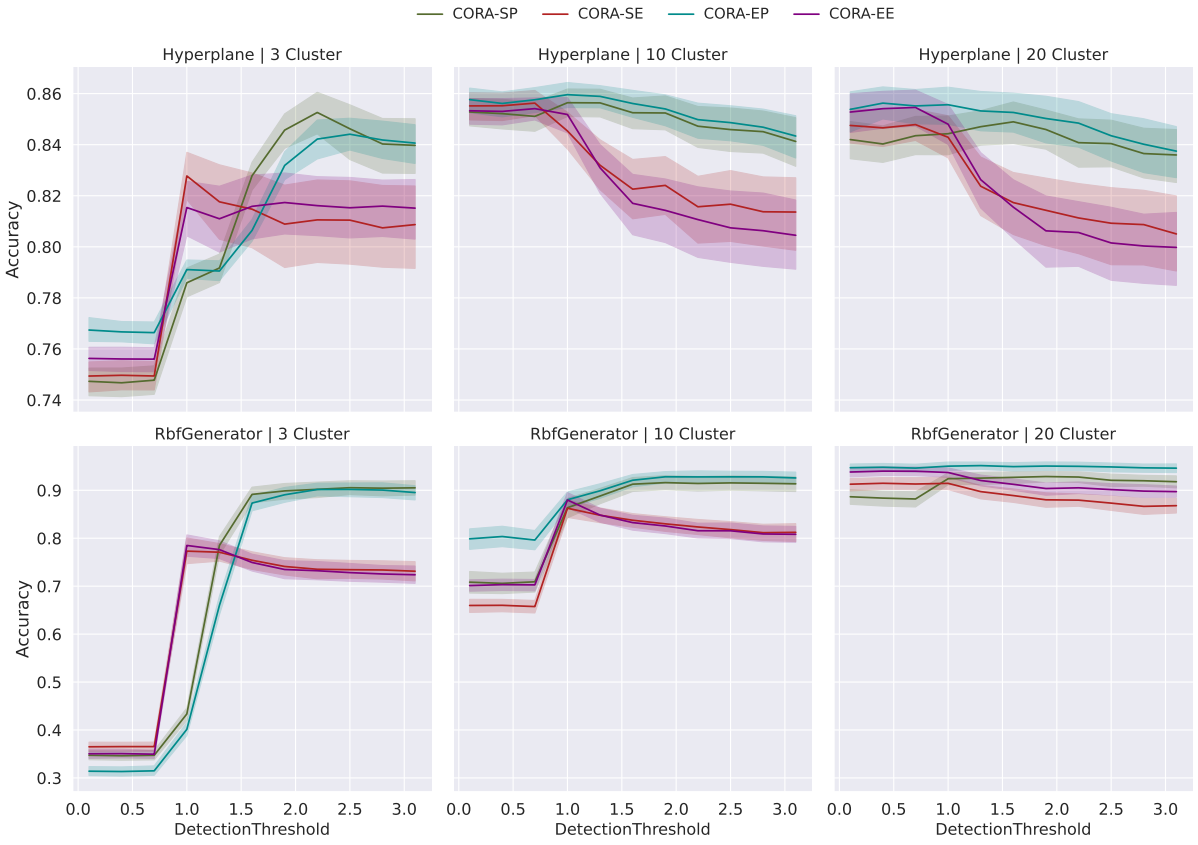
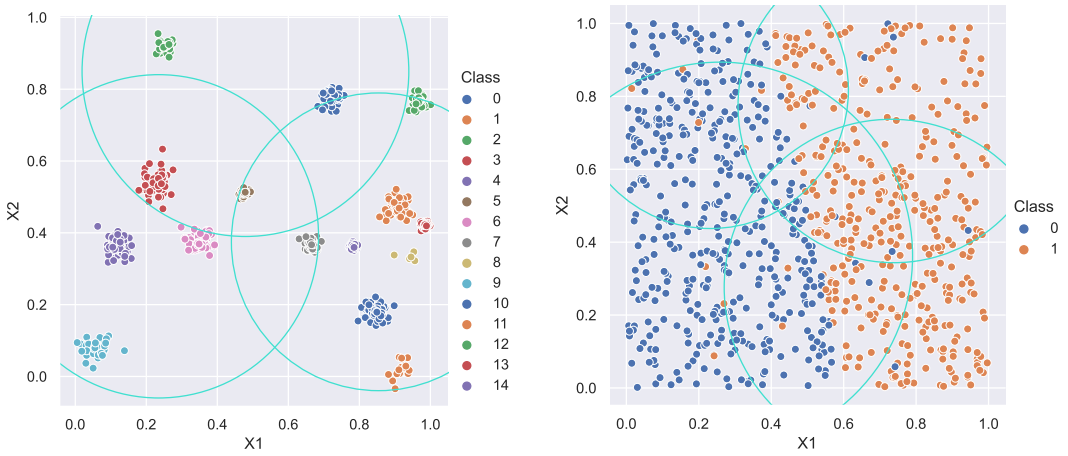


Figure 5.1.: Accuracy of all CORA configurations over the detector threshold and 3 number of clusters



(a) RBF Generator

(b) Hyperplane Generator

Figure 5.2.: Two datasets, clustered with CORA and 3 Cluster

distinction. Consequently, a small number of clusters is not sufficiently detailed enough to isolate the region of concept drifts.

Figure 5.2a illustrates the effect of using just three clusters on the RBF stream. Here, individual clusters (circles) span multiple RBFs, each designating a unique class. The clustering is too coarse for this dataset, as detected changes would remove all observations in the cluster, from which many may not be affected by induced changes. This is particularly vital with low detection thresholds, as the detectors classify changes in the prediction error or class entropy more frequently as concept drifts. Thus, the overreaction happens more often, resulting in low accuracies. However, as the cluster count increases (visible in subsequent columns of the accuracy grid), this effect diminishes. With a larger cluster count (e.g., 20), the system can allocate each RBF its own cluster. This ensures that our adaption mechanism can precisely forget observations from the affected areas while preserving unaffected observations. Here, the adaption mechanism is more accurate, resulting in more stability against varying detection thresholds.

When comparing configurations, prediction error detection techniques like CORA-SP and CORA-EP consistently outperform entropy-based methods when using rather high detection thresholds, regardless of the dataset. On the Hyperplane stream, we observe that with higher cluster numbers, the detection threshold has noticeably less impact on the performance of prediction-error techniques. This superior performance of CORA-SP and CORA-EP at elevated thresholds stems from the spatial structure of the Hyperplane stream. In Figure 5.2b we present the clustered dataspace, using a cluster number of 3. When the hyperplane slightly rotates, the class entropy within the clusters won't change significantly. The change in entropy is not enough to reliably trigger the change adaption mechanism. This is especially evident with higher detection thresholds, as the performance of CORA-SE und CORA-EE begins to degrade significantly upon a threshold of $\delta = 1$. The prediction-based detection is more sensitive to the change by the rotating hyperplane, as CORA-EP and CORA-EE remain relatively stable with only a slight degradation. Similar behavior can be observed on the RBF Generator too.

In conclusion, the importance of the detector threshold is enhanced in scenarios with a vast class count and cluster-like data distributions. In instances with anticipated noise or significant fluctuations, a rather high threshold is advisable to avoid false alarms and unnecessary forgetting. We observed that entropy-based detection provides more robustness

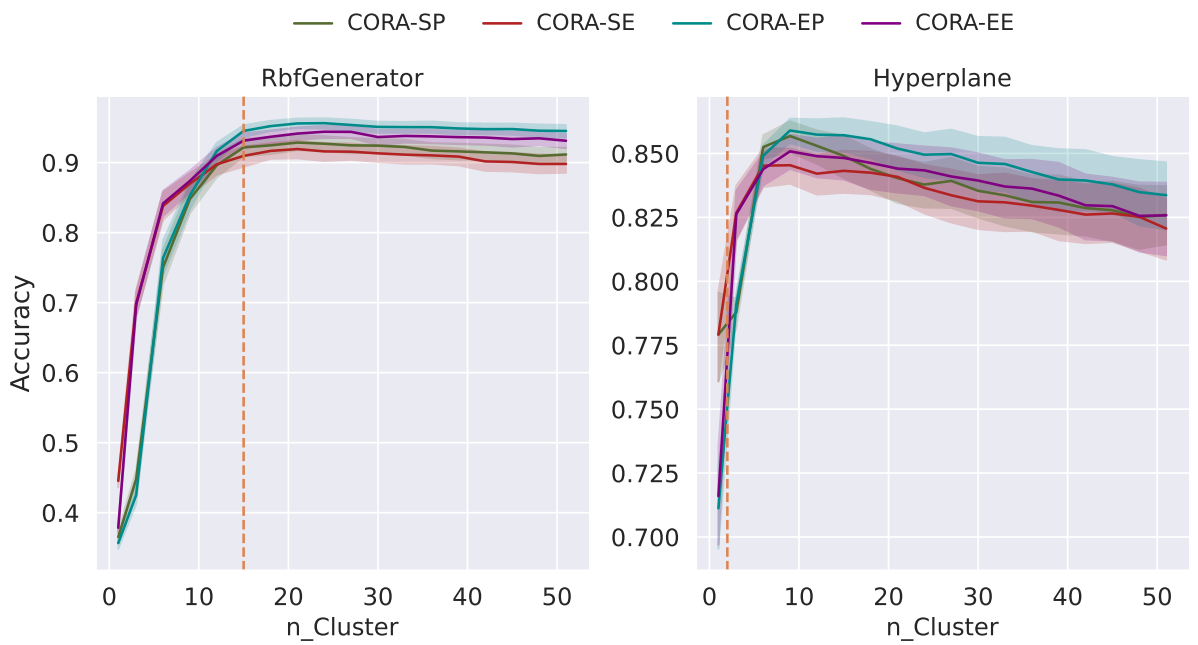


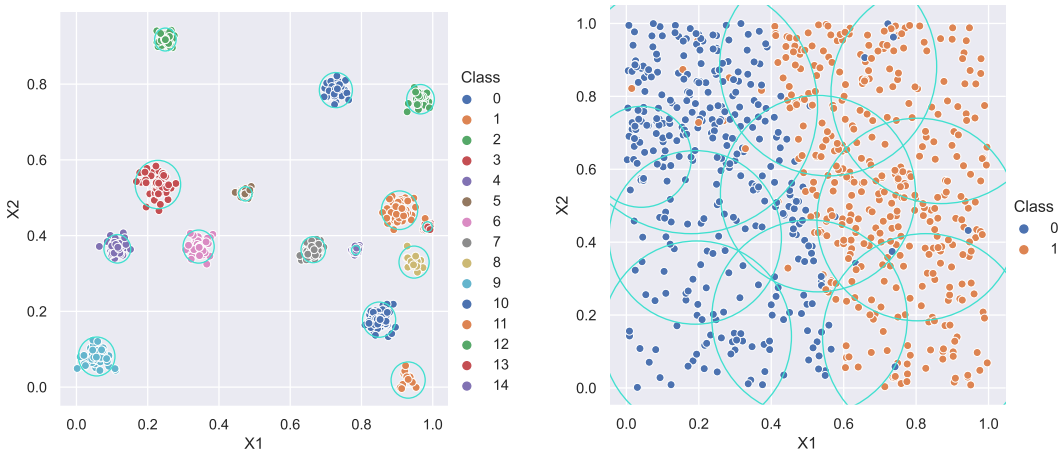
Figure 5.3.: Accuracy of CORA over the number of used clusters

towards the threshold, compared to CORA-SE and EE. However, the performances of all configurations decline past a certain peak on the Hyperplane generator. Based on our findings, a threshold value of 1 offers a balanced performance across the tested scenarios. Thus, we choose $\delta = 1$ as the default detection threshold for all further experiments.

5.2.2. Number of Clusters

In this section, we evaluate the sensitivity of CORA towards the number of employed clusters n_c. Using the determined default detection threshold, $\delta = 1$, we inspect a range of cluster numbers $n_c \in [1, 50]$. In Figure 5.3 we present the relationship between the performance of the four CORA configurations and n_c. Individual datasets are symbolized by columns, and the vertical dotted line indicates their respective class counts. An evident trend is the low performance with limited cluster counts for both datasets. However, as clusters increase, there is an ascent in performance, peaking around $n_c = 10$ for the hyperplane stream and nearing $n_c = 20$ for the RBF generator. Beyond these optimal points, we see a subtle decline in performance across the board. We observe a significant difference between the performance spread on the two tested streams. While the accuracies on Hyperplane begin at around 70% and peak shortly over 85%, the accuracies variance on the RBF Generator is roughly 50%. These deviations in value range and peak point stem from different inherent complexities of the respective classification problems. The RBF dataset generates 13 classes more than Hyperplane. Thus, it demands a greater cluster spread to encapsulate its generated class distributions and thereby optimize both adaptation and prediction.

The entropy-based configurations (CORA-SE and CORA-EE) exhibit a slight edge, particularly when cluster counts are below the class numbers on both datasets. Beyond the optimal cluster range, ensemble entropy techniques tend to be superior. In Figure 5.4a we present our clustering technique on the RBF set, configured with 20 clusters. This contrasts the 3-cluster configuration from the previous section. Now, each RBF is distinctly embedded into individual clusters, ensuring homogeneous segmentation. This clear and homogeneous cluster structure



(a) RBF Generator (b) Hyperplane Generator

Figure 5.4.: CORA Clustering with 10 and 20 cluster

enables our cluster ensemble to support the base classifier with confident class predictions, gaining an edge over the single classifier configurations.

The diminishing returns observed post the optimal peak can be assigned to the limited data available within each cluster. In Figure 5.4b we present our clustering strategy on the hyperplane set using 10 clusters. Here, clusters span the entirety of the data space, exhibiting minimal overlaps and providing a handy spatial separation, which can efficiently support the other components of our framework. However, as the number of clusters further rises, the overlap will increase too, causing individual clusters to encapsulate fewer instances. Given the nature of our active learning framework where labeled instances are inherently scarce, an excessive number of clusters can lead to situations where clusters possess insufficient or even no label-related data. Such sparsely populated clusters not only hinder the predictions when using ensemble classification but also lack information for timely drift detection and adaption.

To summarize, we observed it is beneficial to select a cluster count that surpasses the class count. However, striking the right balance is crucial as too many clusters will diminish the prediction quality. In our experiments, the formula $n_c = c + 8$, where c is the class count of the given classification problem, emerged as an optimal choice. Thus, we choose it as the default value for the following experiments.

5.3. Performance Evaluation

In this section, we compare CORA against three techniques for AL on data streams, namely:

1. **OPAL-NA**: This employs the OPAL [35] labeling strategy alongside the BIQF for budget management. Using no adaptation mechanism, it incrementally learns from the stream. It's a baseline that tests the value of our adaptive mechanisms in the presence of concept drifts.

2. **Zliobaite**'s Adaption Framework (2014) [74]: We've integrated this adaptive approach with the OPAL labeling strategy and BIQF for budget management, ensuring parity in comparison. Similar to our implementation design choices, it employs DDM for drift detection.

Baseline	Labeling Strategy	Change Detection	Classifier
OPAL-NA	OPAL + BIQF	-	Hoeffding's Tree
Zliobaite	OPAL + BIQF	DDM	Hoeffding's Tree
PEFAL	VarUncertainty	Implicit	Hoeffding's Tree

Table 5.3.: Baseline configuration

3. **PEFAL** by Xu *et al.* (2016) [65]: A more evolved version of Zliobaite's adaption approach. PEFAL does not provide the flexibility to align the active learning strategy with ours. Thus, we use the Variable Uncertainty Strategy from its original publication, a specialized version of uncertainty-based sampling combined with strategic randomizations. Furthermore, the framework structure has an implicit change detection mechanism, eliminating the requirement for an external detection module.

For all strategies, Hoeffding's Trees act as the base classifier. Any hyperparameters are set based on the respective original studies. We've selected the latter two strategies because of their similarities with our work: both emphasize explicit (or implicit) change detection and both use incremental learning combined with trigger-based adjustments of the classification model. OPAL-NA serves as a non-adaptive baseline to benchmark the efficacy of employed adaption mechanisms. We provide a brief overview of these benchmark models in Table 5.3.

For all following experiments, we evaluate the performance on stream partitions of 10,000 instances over budgets, ranging from $b = 0.01$ to $b = 1$. We divide the evaluation of prediction accuracies into artificial- and real-world streams. As we can evaluate the approaches in a controlled environment and know the type and position of drifts, we can conclude valuable insights about the capabilities of our framework. This knowledge helps to interpret the results on real-world data sets. The detailed performances on all datasets can be observed in Appendix A.1.

5.3.1. Artificial Data streams

In Figure 5.5, we present the comparative performance across different labeling budgets among all contenders. The results are on four artificial datasets: Hyperplane, Chessboard, Rbf Generator, and SEA. Firstly, we observe that all adaptive approaches demonstrate a significant enhancement in performance compared to the non-adaptive OPAL-NA strategy on all datasets but SEA.

CORA-EP outperforms all other candidates on both the Chessboard and RBF Generator datasets, closely followed by CORA-EE. These two datasets are distinct among the four under consideration due to their clustered data space. The superiority of both ensemble-based techniques indicates the effectiveness of our modified clustering, which substantially contributes to prediction performance. The RBF dataset, in particular, is characterized by clearly outlined data clusters, a trait that is reflected in its performance metrics. All four CORA configurations significantly surpass the baseline approaches. Moreover, in this dataset, concept drifts occur only locally by exchanging random RBFs, while the rest remain unchanged. This scenario is ideal for our local partial forgetting mechanism. While other adaptive strategies like Zliobaite and PEFAL discard all information upon detecting a change, our approach retains the unchanged information.

Both the Chessboard and the RBF Generator datasets exhibit the most frequent and pronounced concept drifts among the datasets in focus. For the RBF Generator, local drifts occur

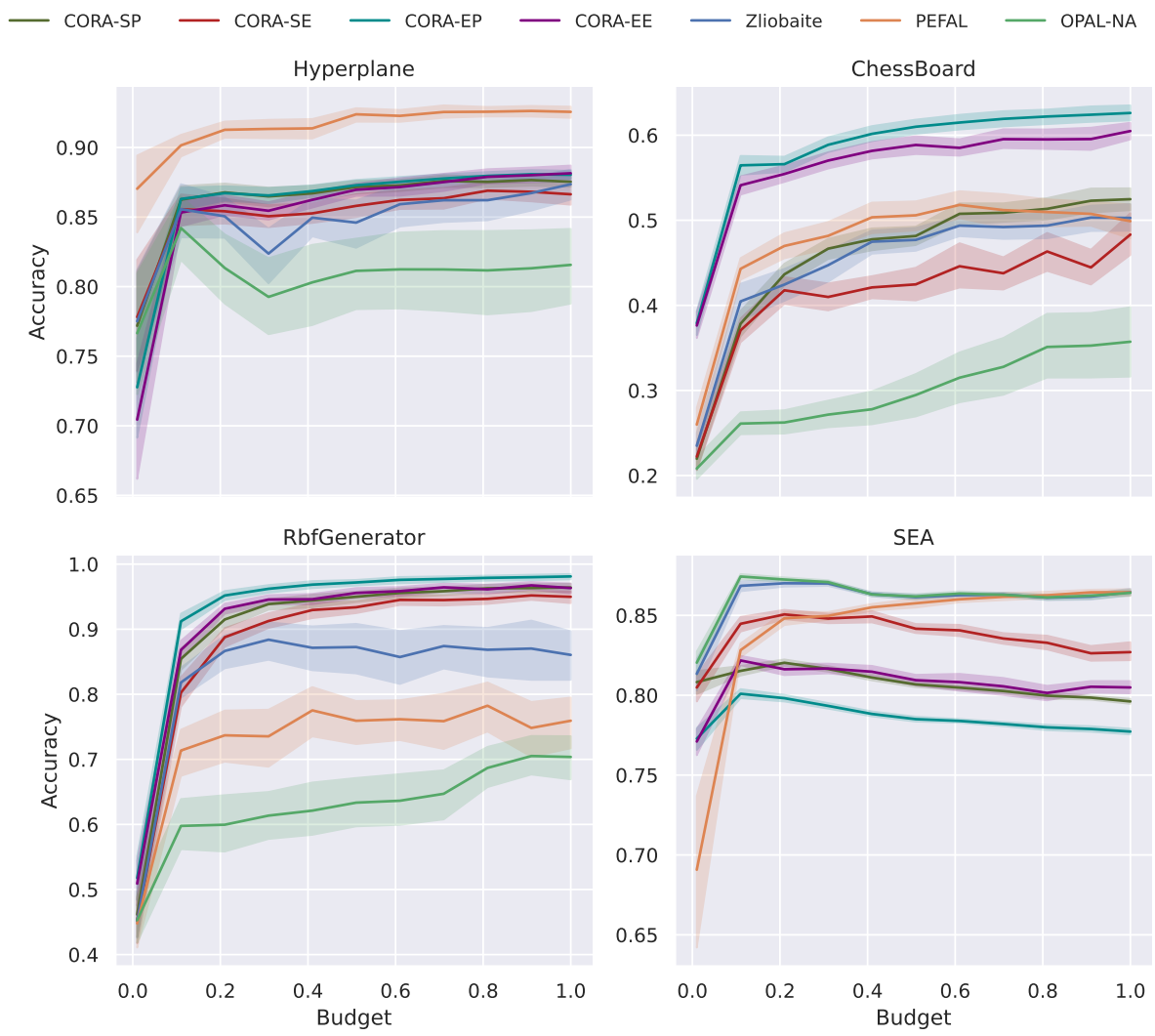
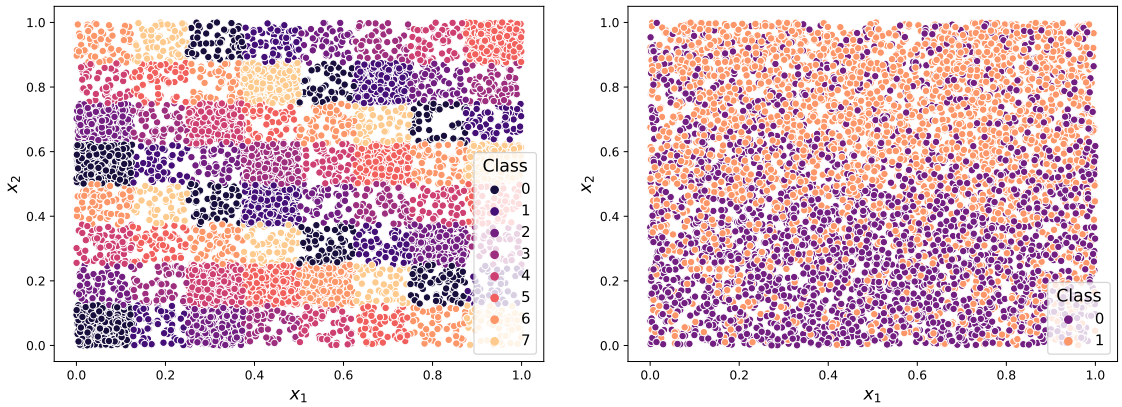


Figure 5.5.: Accuracy comparison over budgets $b \in [0.01, 1]$ on artificial datasets

every 2000 instances, whereas the Chessboard dataset witnesses a global abrupt concept drift every 1000 instances through the unveiling of a new concept. The frequency and intensity of these drifts are manifested in the poor performance of the non-adaptive OPAL method in comparison to the adaptive techniques.

In contrast to the RBF stream, single classifier strategies, specifically CORA-SE, stumble with the Chessboard dataset. We present a snapshot of the stream after 4000 instances in Figure 5.6a. This stream is inherently difficult, with seven classes emerging at different locations within the data space. Both of our ensemble methods seem to better capture the difficulty of the classification problem, as they both outperform all other techniques. Yet, the adaption mechanisms of all adaptive techniques showcase their value as they improve the performances in contrast to the non-adaptive baseline.

When examining the performances on the Hyperplane stream, it's evident that most techniques demonstrate relatively similar accuracies across all budgets. Only two techniques distinctly stand out: the non-adaptive approach OPAL-NA and the PEFAL method. Figure 5.6b displays the stream state after 4,000 instances. Even though we can vaguely estimate the boundary separating the two classes, instances from one class appear in both halves of the data space. This scenario arises due to the continuously rotating hyperplane that distinguishes



(a) Chessboard (b) Hyperplane

Figure 5.6.: Stream snapshots after 4000 Instances

the two classes. Without an adaptation mechanism, classification models struggle to learn the current boundary swiftly enough to keep pace with the rotation. They base their predictions on old states of the hyperplane, as indicated by the subpar performance of OPAL-NA. In contrast, PEFAL seems to be ideally suited for this context. The paired-ensemble approach, which trains one classifier on uncertainty-based sampling and another on random sampling, swiftly recognizes the incremental changes. Through their adaptation mechanism, a model trained on the latest randomly selected instances replaces the other. This enables the algorithm to respond effectively to drifts in such mixed stream states since the new classifier understands the global context. All other methods oscillate between these two baselines. The prediction error-based drift detection configurations, CORA-SP, and CORA-EP, have a slight advantage over entropy-based methods. This can be attributed to the characteristics of local entropies in this classification problem, a phenomenon we previously discussed in Section 5.2.1. In general, all CORA configurations seem to be advantageous compared to the adaptive Zliobaite technique. However, the differences are not statistically significant on Hyperplane.

The results on the SEA dataset differ significantly from those observed in the other streams. OPAL-NA exhibits the most accurate predictions, especially at lower label budgets, whereas CORA-EP, which performed exceptionally well on all other datasets, ranks at the bottom in this context. The challenge associated with the SEA dataset comes from its structure: an abrupt concept drift is only introduced every 12,500 instances. As we evaluate stream partitions of 10,000 instances, each partition comprises at most one major change. The stream complexity is primarily attributed to the added 10% noise. Consequently, the accuracy of the non-adaptive OPAL approach is highest in this context. The balanced labeling strategy of OPAL exhibits a commendable resilience to noise, providing an optimal setting in the absence of major changes. The Zliobaite method generates almost identical accuracies, which indicates that the framework might not be detecting changes. This is the correct behavior in this instance, as all detections besides the single major concept drift, are false alarms triggered by noise. Initially, PEFAL struggles but approaches similar accuracies at higher budgets.

Our CORA configurations underperform in this context, Especially, the CORA-EP approach exhibits noticeably poorer performance in contrast to the non-adaptive approach. An unusual observation is the declining performance with increasing budgets. We assume this trend is induced by excessive adaptation processes and enhanced sensitivity to noise. As the volume of labels increases, the possibility of false alarms rises, which might negatively impact our

accuracy with higher labeling budgets. Nevertheless, entropy-based configurations appear to be more resilient to noise compared to prediction-driven approaches.

In summary, we found significant distinctions in the performances on artificial streams. While adaptive strategies consistently outperformed the non-adaptive OPAL-NA, there were exceptions, notably the SEA dataset. The CORA-EP approach performed superior on both Chessboard and RBF, showcasing the efficacy of our clustering-based ensemble techniques, especially in naturally clustered data spaces. Datasets with frequent and abrupt concept drift, such as Chessboard and RBF Generator, highlight the strengths of our framework. The naturally clustered RBF data space with local concept drifts in particular, as every CORA configuration surpassed all baselines significantly. On the Hyperplane stream, techniques exhibited comparable accuracies, with OPAL-NA and PEFAL as outliers for differing reasons. The SEA dataset presented unique challenges, with OPAL-NA excelling due to the stream structure and inherent noise, while some previously top-performing techniques like CORA-EP struggled. Across datasets, we also observed the impact of local prediction error and cluster entropy in drift detection, with the former showing a slight edge. These insights provide a foundation for our following evaluation on real-world datasets.

5.3.2. Real-World Data Streams

In the following, we evaluate the performance on selected real-world datasets. We present the experimental results in Figure 5.7

We see that the differences in performance largely depend on the evaluation set. For instance, while accuracy for the Electricity dataset ranges between 65% to 85%, the Airlines dataset exhibits a much lower variance of roughly 5%. The shape of the performance curves with respect to the budget also varies considerably between datasets. Both Electricity and PokerHand datasets reveal a positive correlation between increasing labeling budgets and enhanced performance across all approaches. In contrast, datasets such as Covertype and Airlines show negligible to no influence to the number of acquired labels.

Remarkably, one CORA configuration emerges as a top performer in every dataset under consideration. Especially notable are the ensemble variants, CORA-EP and CORA-EE, which demonstrate significantly high accuracies in the Electricity, CoverType, and PokerHand datasets. This superiority in performance, particularly evident in the Pokerhand, showcases the value of incorporating our clustering in class predictions. The paired-ensemble of a base classifier and clustering seems to better capture the complexities of most data streams, compared to a single classifier model. The clustering breaks down the classification problem into multiple smaller decision boundaries, which simplifies the local prediction tasks in the cluster. This might elevate the model's ability to predict the correct classes.

Single-classifier CORA configurations struggled in these three streams, as we observe similar performances to OPAL-NA or even sometimes lagging behind. Inspecting the Covertype dataset, it's evident that this underperformance is caused by not detected changes. Zliobaite outperforms both CORA-SP and CORA-SE while their outcomes converge to similar results as PEFAL and OPAL-NA. This suggests that in contrast to Zliobaite, these techniques fail to detect changes and apply adaption mechanisms, as these reactions are the reason that Zliobaite surpasses the non-adaptive baseline. Equally notable is the near-identical performance of the ensemble strategies CORA-EE and CORA-EP. This pattern further enhances the potential issues in our framework's capacity to identify changes within this dataset. Nevertheless, the high accuracy rates achieved by the ensemble versions, occasionally surpassing 95%, underscore the potency of merging our labeled clustering with the base classifier.

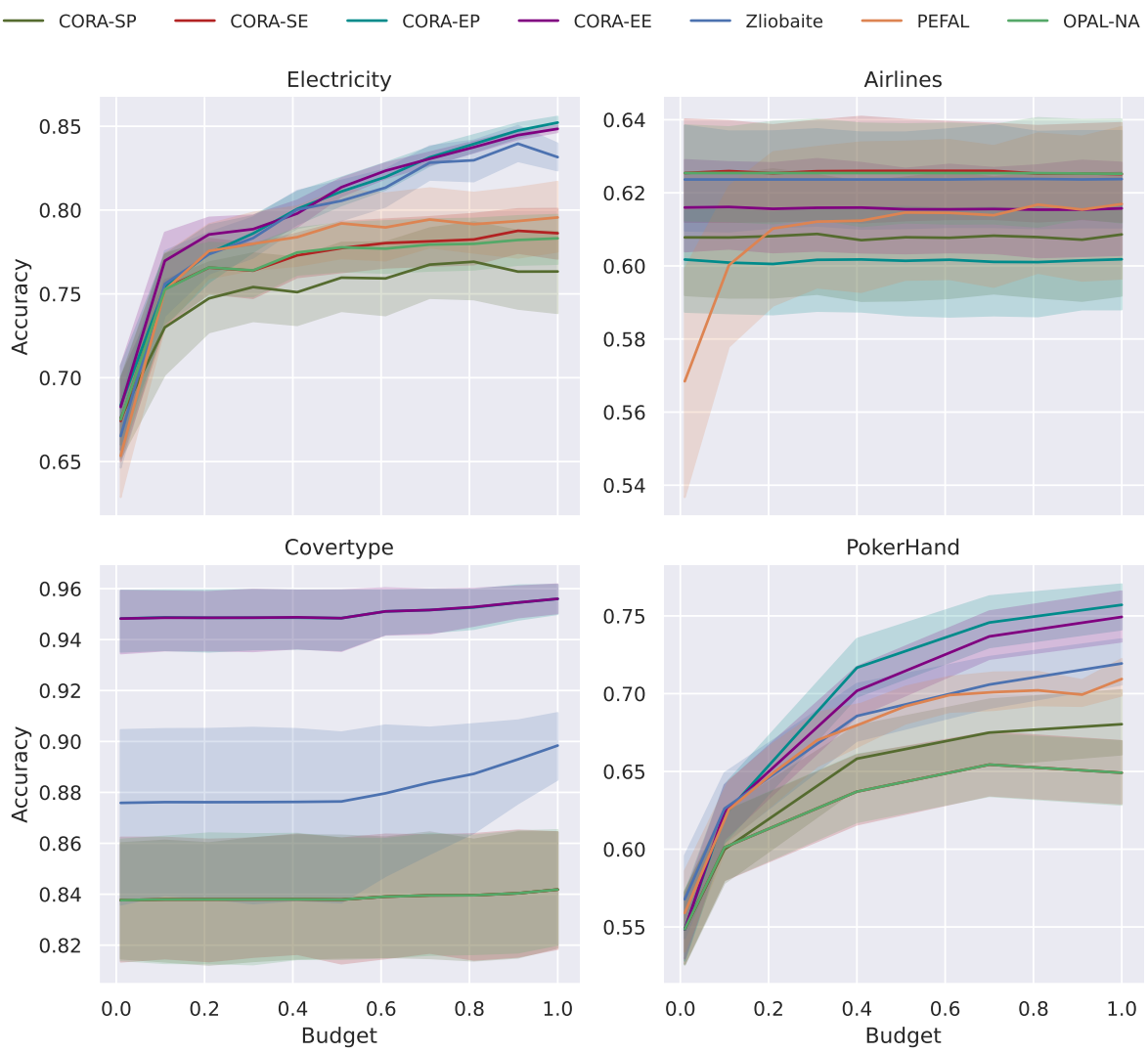


Figure 5.7.: Accuracy comparison over all budgets on real-world datasets

The analysis of the Airlines dataset presents a contrasting circumstance. Here, CORA-SP dominates the performance together with Zliobaite and OPAL-NA. The supremacy of our ensemble frameworks is not present in this context. The superior performance by the non-adaptive baseline suggests possible over-adaptations in some techniques due to misidentified drifts. A distinct performance gap emerges between prediction-based and entropy-based change detection. CORA-SP and CORA-EP are overshadowed by their entropy-focused counterparts. This pattern aligns with the insights we conducted in Sections 5.2.1 and 5.3.1, suggesting that prediction error-based mechanisms might be more sensible to noise or minor drifts. Furthermore, the integration of clustering into the classification model does not yield the same advantages, as seen in other datasets. This inconsistency is particularly evident when observing CORA-EE, which falls short compared to its single classifier variant, CORA-SE. Considering the dataset's attributes, we recognize challenges in clustering: two of the seven attributes are categorical, and some exhibit a range of over 200 potential values. Such complexity hinders effective clustering, which, in turn, influences the ensemble predictions reliant on clustering decisions.

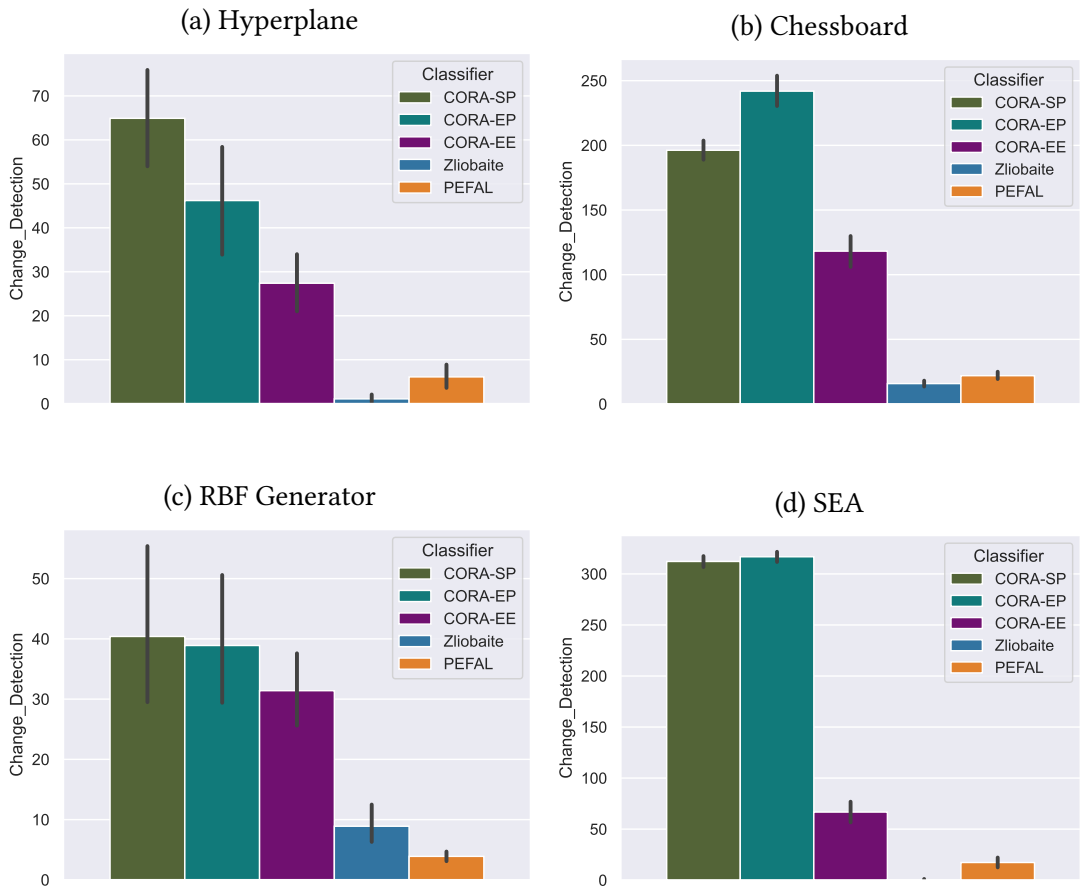


Figure 5.8.: Two datasets, clustered with CORA and 3 Cluster

5.3.3. Change Adaption

In order to gain a deeper understanding of the experimental results, we further investigate the change adaption capabilities of CORA and baseline approaches. This extended evaluation includes the framework's abilities to both detect changes correctly and react to them efficiently.

5.3.3.1. Detection

In Figure 5.8, we showcase the average number of total change detections by the adaptive approaches across the artificial datasets. It is important to note that we omitted CORA-SE from this representation, as its detection mechanisms are identical to those of CORA-EE. In the subsequent sections, we analyze these detected changes and draw correlations with the performance results we discussed earlier.

The standout observation is that all CORA variants register a significantly higher number of detections compared to the other two baselines. This disparity comes from our strategy of conducting local detections within clusters, in contrast to the global approach adopted by both Zliobaite and PEFAL. This implies that a single change in the data stream could activate multiple clusters to recognize and register a change. For instance, consider the RBF dataset: each stream partition inherently contains 4-5 changes. Each of these changes corresponds to a swapping of positions by 2-14 of the RBFs in the data space. Following our derived formula for the number of clusters, we use 23 in this context. This leads to a scenario where multiple

clusters encapsulate the same RBF. Consequently, when a change occurs, ideally all clusters covering one of the mutating RBFs should detect this shift. This explains the observable average detection count of up to 40 changes for CORA-SP, in contrast to PEFAL and Zliobaite, which identify fewer than 10 changes. The efficacy and correctness of our approach in this context is demonstrated by the significant superiority we achieve for all four CORA variants in comparison to Zliobaite, PEFAL, and the non-adaptive baseline.

However, CORA does not consistently identify the correct changes across all streams. Taking the SEA stream as an example, we observe an excessive number of detections (over 300) when using prediction accuracy as change indicator with CORA-SP and EE. Given that the complexity of the SEA stream is primarily influenced by noise rather than the potentially singular global change, many of these detections turn out to be false alarms, triggering unnecessary adaptations. This observation is reflected in the accuracies of the CORA approaches on the SEA stream, where they all fall short of the performance of OPAL-NAN. Zliobaite, which registers minimal to no changes in the SEA stream, matches the performance of OPAL-NA. In general, we observe a pattern that entropy-based designs, signal notably fewer changes than their prediction accuracy-based counterparts. This supports our previous hypothesis that entropy-based approaches, namely CORA-SE and EE, are less sensitive to variations, thereby providing stability to noise. As a result, they outperform the prediction error configuration in streams characterized by minor changes and noise, such as SEA or Airlines.

Nonetheless, this reduced sensitivity comes with its own set of challenges, particularly in streams with several significant changes. In the other three artificial datasets, which involve a variety of drift types, the prediction-based drift detection configurations consistently surpass their entropy-based counterparts.

The contrast between class entropy and local prediction error as indicators for changes becomes further evident when considering the average detected changes in real-world datasets, as presented in Figure 5.9. Apart from negligible detections in the Electricity dataset, CORA-EE largely failed to identify any changes, which negatively reflected in its performance on certain datasets. Both entropy-based strategies, with the exception of the outlier Airlines, which presumably lacks significant changes, tend to perform at the lower spectrum compared to other contenders.

In general, we observe that the ratio of detected changes between CORA and the baseline approaches on real-world streams differs considerably. Whereas CORA consistently registered the highest number of detections across artificial streams, its change detections on real-world streams significantly deviate between the data streams. Notably, the Covertype dataset stands out where none of the CORA configurations detect any change. Yet, the ensemble CORA approaches outperform others. In this context, Zliobaite, despite detecting numerous changes, does not come close to matching the performance of CORA-EP and CORA-EE. This shortfall in detections can likely be attributed to the nature of the instance space. Given that the Covertype dataset comprises 54 features, its dimensionality is substantially higher than the other datasets. As dimensionality increases, identifying coherent subsets that share the same concept becomes more challenging [54]. Consequently, change detection within these clusters becomes more difficult, given that the contained instances might not necessarily exhibit the desired class similarities. As evidenced by Zliobaite's performance, a change detection based on global prediction error seem to be more suitable for such data streams.

Examining the Electricity and Pokerhand datasets, the CORA approaches outperform even though they have fewer detections compared to PEFAL and Zliobaite. This superior performance can be attributed to the efficiency of our local adaptation mechanism, which we further discuss in the following.

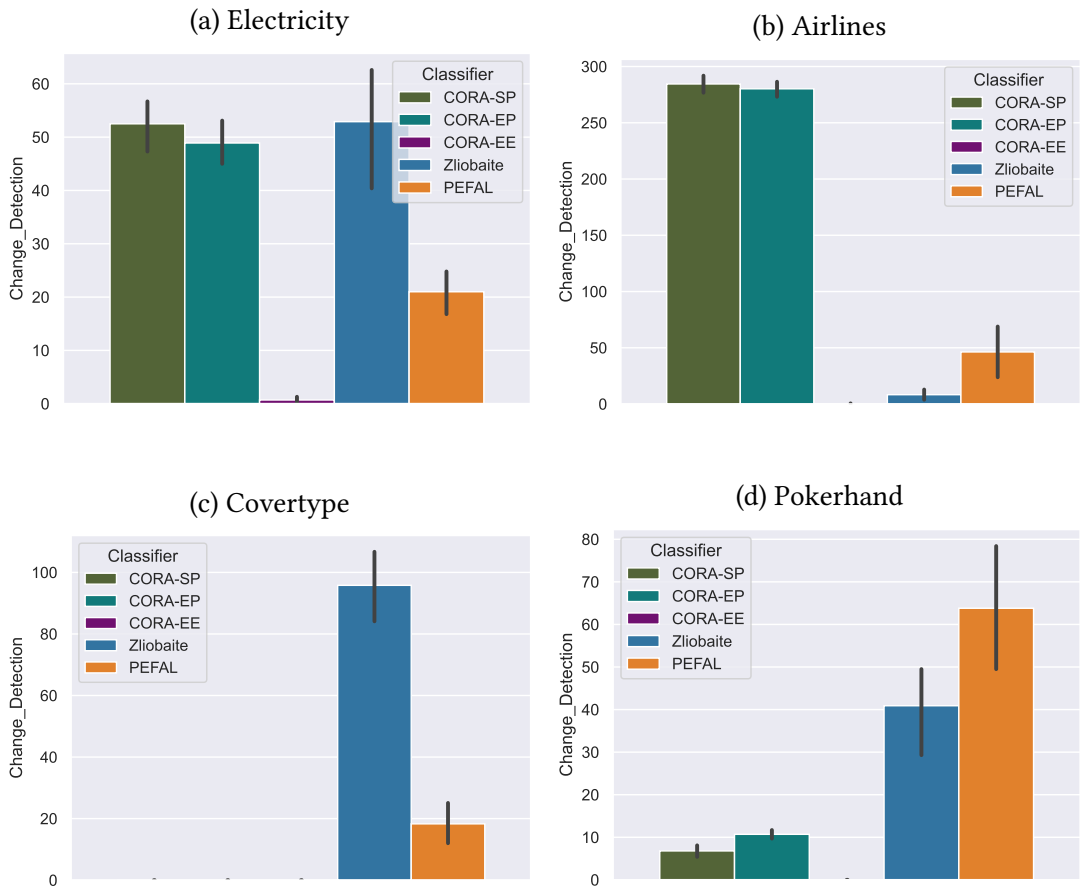


Figure 5.9.: Two datasets, clustered with CORA and 3 Cluster

5.3.3.2. Adaption

We observed that the CORA approaches, particularly with prediction accuracy as a change indicator, tend to register a higher average of detections across certain datasets. Taking the Chessboard dataset as an example, CORA-EP averages nearly 250 detections per 10,000 stream instances, which is over 200 more than both Zliobaite and PEFAL. Nonetheless, CORA-EP surpasses the baselines significantly. A similar trend is evident on datasets such as the Rbf Generator, Electricity, Hyperplane, and Pokerhand. This performance advantage stems from our partial forgetting mechanism.

Figure 5.10 offers snapshots of instances from the Rbf Generator that the current classifier has learned under various frameworks. This snapshot captures a moment 20 instances after RBFs interchanged their positions, with a labeling budget of $b = 0.5$. For clarity, we generated only five distinct Rbfs in this experiment, each corresponding to its own class. Observing the snapshot from the OPAL-NA approach in Figure 5.10a, the nature of the change becomes evident. Given that OPAL-NA lacks an adaptive technique, it clearly showcases the concept before and after the change. The RBFs generating classes 0 (Blue) and 3 (Red) swap positions, as can be identified from the newly labeled instances, surrounded by different labels. Since the OPAL-NA approach lacks direct adaptation techniques, the classifier learns the new concept on top of the old, significantly hindering its performance.

In contrast stand the adaptive methods: CORA, Zliobaite, and PEFAL. Examining the post-drift data in Zliobaite in Figure 5.10b, we notice that the entirety of the learned state prior to

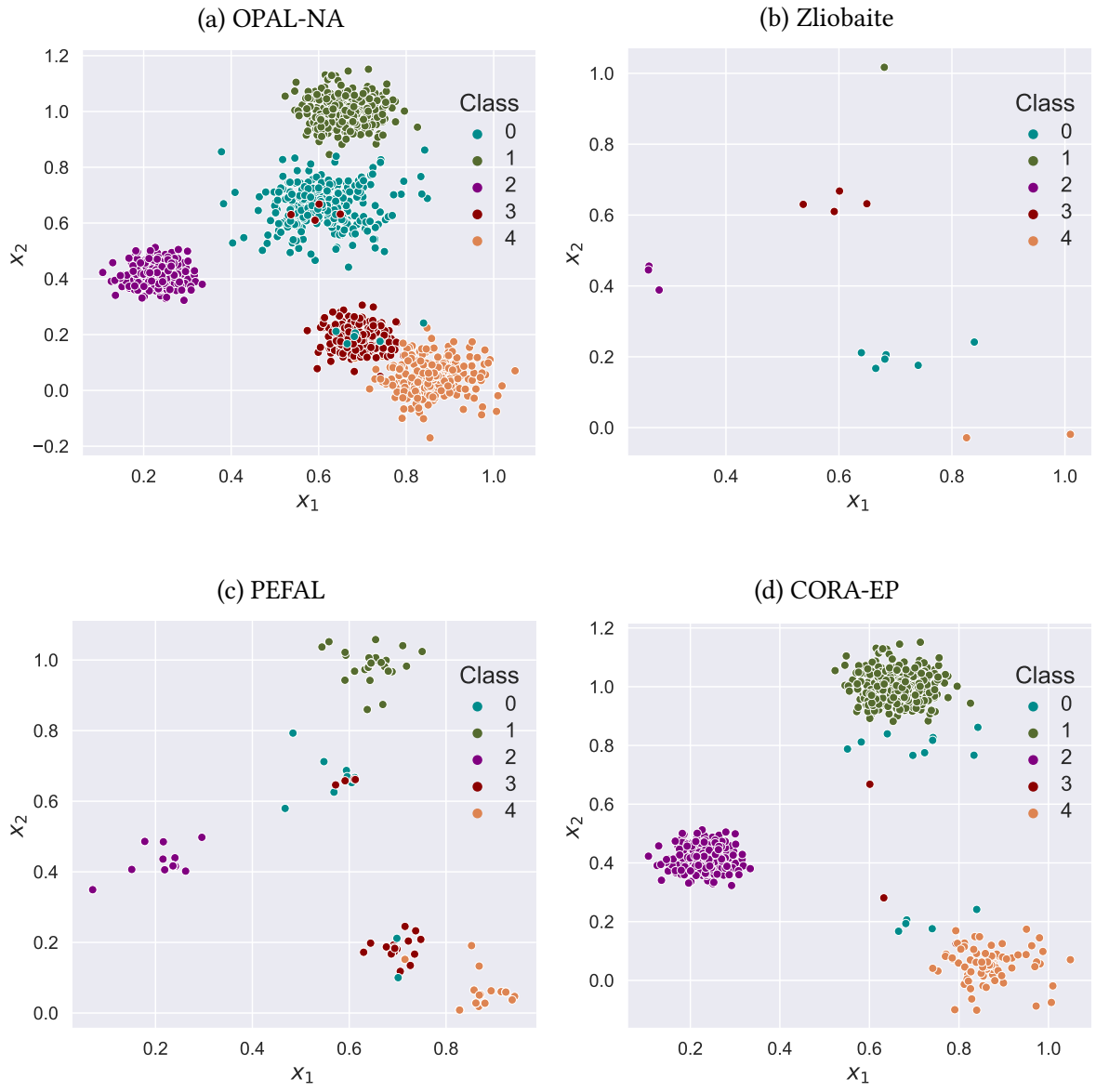


Figure 5.10.: Snapshots of learned instances, 40 timesteps after concept drift occured in RBF

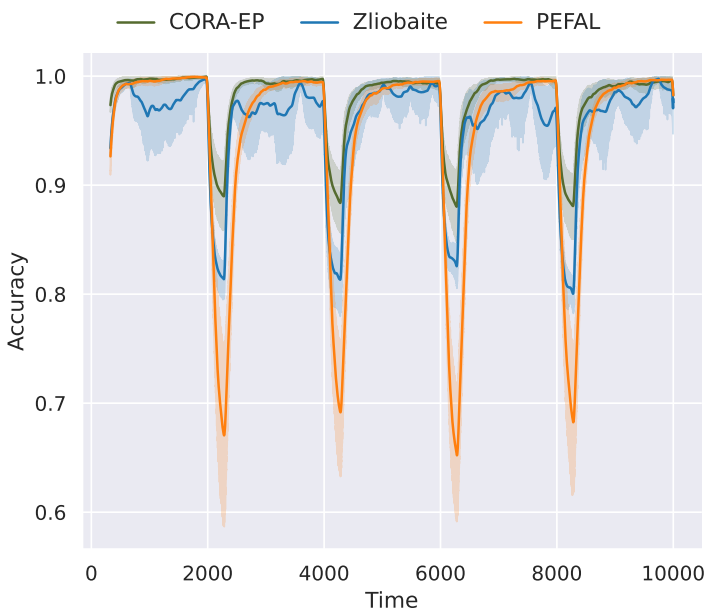


Figure 5.11.: Accuracies of adaptive approaches over time on RBF Generator

the drift is discarded. The learned knowledge comprises solely the new instances post-change detection. This approach offers the advantage of a fresh start, enabling quicker adaption to the new concept than if outdated information persisted. However, the downside is that still-relevant knowledge was discarded since the other three Rbfs represented the current concept. In the realm of active learning, where labels might be sparse, relearning a concept from scratch often results in delayed responses.

PEFAL, on the other hand, retains pre-drift data as visible in Figure 5.10c. Yet, since the new classifier was trained randomly on the most recent instances, the learned instances encapsulate Rbfs that correspond to the outdated concept. As opposed to the Zliobaite approach, old and new concepts mix, similar to the OPAL-NA behavior. However, since only a few old instances are present, PEFAL will swiftly overwrite the old concept with new training data as the stream progresses.

In Figure 5.10d, we observe the state of our CORA-EP approach after the concept drift. The effectiveness of our local forgetting mechanism is evident. Except for a few outliers that have been miss-clustered, our adaptation mechanism identified and discarded the altered RBFs while retaining the unaltered ones. Consequently, the classifier does not forget valuable training data, which is reflected by its performance.

In Figure 5.11, we present accuracies over 10,000 timestamps for the RBF generator, averaged across 30 repetitions and with a label budget $b = 0.5$. The moments of change are clearly displayed by the performance dip across all included frameworks. It is clear that all candidates identify the drift almost simultaneously and compensate through their adaptation strategies. However, the striking difference lies in the recovery time. CORA-EE exhibits a notably quicker return to its pre-drift performance levels compared to other methods, due to our partial forgetting mechanism. Following CORA-EE, the Zliobaite approach takes longer to recover and generally doesn't achieve stable accuracies as CORA-EP due to frequent false alarms. This extended recovery period can be attributed to its strategy of not retaining any pre-drift information. Hence, the classifier learns the new concept after a full reset and, given our label budget of 50%, it learns with only every second instance on average.

PEFAL, on the other hand, takes the longest time to return to its previous accuracy levels. Yet, unlike Zliobaite, it doesn't suffer from false alarms, as evident from its consistent accuracy during stable stream states. The further delayed reaction time stems from its model's strategy of retaining pre-drift information. However, this retained data is not selectively chosen but randomly distributed across the data space. This requires the new concept to be learned on top of a few instances from the old model.

In summary, our CORA configurations showcase their capability to address the stability-plasticity dilemma in stream learning. In a series of tests, especially evident in the RBF Generator dataset, CORA's local forgetting mechanism stands out. Unlike other methods that either discard entire learned knowledge post-drift or retain data randomly across the space, CORA identifies and discards only the altered concepts while preserving the unchanged ones. This ensures that the classifier doesn't lose valuable training data, allowing for both stability (retaining useful old knowledge) and plasticity (adapting to new patterns swiftly).

5.3.4. Summary

In our evaluation, we tested the four CORA configurations against three baselines across both real-world and artificial datasets, spanning various labeling budgets and multiple repetitions. We also analyzed our framework's capacity to detect changes across diverse data streams and a particular focus on the efficiency of our novel local adaption mechanism.

From our experiments, at least one CORA configuration emerged as the best performer in six out of the eight datasets, with the ensemble versions especially standing out. The superior performance across these datasets can be attributed to two primary factors:

1. The collaboration between the base classifier and our distinct clustering algorithm to predict the current class labels.

2. The effectiveness of our partial forgetting adaption mechanism.

Our investigations revealed that our adaption mechanism, when applied to datasets with cluster-like structures and pronounced changes, achieves significantly faster recovery post-drifts. Furthermore, CORA's approach exhibited resilience against false alarms, discarding only minimal segments of previous information, thus mitigating the impact of these alarms better than other methods.

We observed that using prediction error as a change indicator tends to be more sensitive compared to entropy-based detection. This sensitivity proves advantageous in datasets characterized by numerous substantial drifts. However, in scenarios with minor drifts and noise, the robustness of entropy-based detection is more favorable. Within such noisy datasets without significant drifts, our CORA-SP approach stands out among the other configurations, although it may not always outperform the baselines.

In a general context, CORA configurations, especially the ensemble variants, encountered challenges with datasets possessing high dimensions or categorical attributes with a substantial number of possible values. The inherent difficulty arises from the diminished ability of the clustering to form subsets of neighboring instances with similar concepts. This impacts both the local change detection and the added value for predictions in ensemble configurations.

5.3.4.1. Complexity

The computational and memory complexity of CORA is defined by the following components:

- Labeling strategy
- Base classifier
- Clustering
- Adaption mechanism

As the labeling strategy and base classifier are interchangable components and present in every AL approach, the additional complexity of CORA comes from the clustering and adaption mechanism. Note that our ensemble configuration causes no additional complexities as it is directly leveraged from the existing clustering model.

Our clustering stores CFs of n_c clusters. The CFs comprise 6 aggregated statistics and a fixed size sliding window of w instances. As n_c and w are stationary parameter of fixed size, the memory complexity is finite in $O(1)$.

Regarding the computational complexity, the major factors are the maintenance of the clustering model and the adaption mechanism. The online clustering process involves adding new instances to the CFs, merging CFs and deleting cluster. Adding instances to the CF is straightforward as the aggregated statistics are additive and thus the features and labels can be simply incremented with the instance. The sliding window LI is updated by inserting the new datapoint and its timestamp. Merging clusters is similar. Only the merge of LI requires additional consideration as we draw the w most relevant instances of both clusters. Thus, it involves a comparison of each timestamp in both LI ($O(w^2)$). To delete clusters, we need to consider each of the n_c cluster and compare their relevance stamp to the relevance threshold ($O(n_c)$). The adaption mechanism deletes the changed cluster and retrains the classifier model. Retraining involves gathering the training data from LI of all n_c cluster ($O(n_c * w)$ and the training process of the classier. The complexity of the training depends on the chosen base classifier and the amount of training data that is stored in the clusters which is bound to $n_c * w$. As w and n_c are both constants, defined by the user, the additional computational complexity of CORA is independent of time.

5.3.4.2. Requirements

In Chapter 1 we defined requirements for adaptive AL algorithms in data streams and further refined them in Chapter 3 to address all types of concept drifts and strike a balance in the *stability-plasticity-dilemma*. In summary, we demand: **R1** a balanced labeling strategy, **R2** continous learning, **R3** detecting changes, **R4** forgetting changed instances and **R5** preserving unaffected knowledge.

Our flexible framework allows for any online labeling strategy to be deployed. Thus, we chose OPAL as it effectively balances exploration and exploitation (**R1**). We employ an incrementally learning base classifier that is only rebuild if changes are identified (**R2**). Our detector ensemble, monitoring local prediction errors or class entropies inside the clusters ensures that concept drifts are detected in any area of the feature space (**R3**). As the activated detector also provides the area of change, our partial adaption mechanism retrains the classifier model to forget instances from the changed cluster (**R4**) while preserving the observation from the unaffected cluster (**R5**)

6. Conclusion and Future Work

Datastreams present unique challenges for active learning, particularly when faced with changes in data behavior, known as concept drifts. Online active learning frequently employs incremental classifiers to continuously learn incoming instances and train robust models. However, they often suffer from delayed reactions to concept drifts due to reliance on outdated instances. To adequately respond to changes, they need to implement active adaption mechanisms. Previous adaptive online active learning approaches use profound forgetting mechanisms to discard outdated information after drifts. Yet, concept drifts can be local, manifesting only in certain regions of the instance space. Their adaption approach leads to inadvertently discarding valuable training data, inhering current concepts. In this thesis, we identified five essential requirements that adaptive active learning algorithms should meet to address the stability-plasticity dilemma effectively.

We introduced the adaptive active learning framework CORA. This solution enhances online clustering with labeling statistics and recent training data. We combine this with an incremental learning classifier. An ensemble of change detectors continually monitors cluster statistics to detect regional changes anywhere in the feature space. If a change is detected, the affected clusters are discarded and the classifier retrains on the remaining clusters. This novel adaption mechanism ensures that the model forgets outdated instances while preserving intact knowledge. We explored various design choices for CORA, including a promising ensemble variant that further leverages the enhanced clustering for class predictions. Furthermore, we evaluated two different change indicators: Local prediction error and class entropy. A significant strength of our approach lies in its flexibility, allowing for arbitrary online labeling strategies and incremental classifiers to be seamlessly integrated into the adaption framework.

Our experiments showcased the efficacy of CORA's change adaptation mechanism. We achieved significantly faster recovery from local concept drifts than previous adaption approaches. Especially in data streams with naturally clustered instances and frequent severe drifts, our ensemble approach consistently surpassed other established online active learning techniques. Both, monitoring local prediction error and the local class entropy in clusters showed promising results concerning the efficiency of drift detection. Local prediction error change detection proved superior in data streams marked by significant changes, while the entropy-based approach showcased significant stability toward data streams with noise or subtle incremental drifts.

Our framework faced problems with datasets characterized by high dimensionality. Furthermore, datasets with nominal attributes that inherit a wide range of potential values challenge CORA's clustering capabilities. These characteristics often hinder its ability to spatially group observations sharing classes. Thus, the adaption mechanism loses precision and the contribution of the clustering to the class predictions is limited.

In future research, we aim to further leverage our clustering structure in the overall process. For instance, the labeling statistics in the cluster can be used to employ a balanced labeling strategy, that strategically obtains labels over the feature space. An alternative direction could be to improve the computational efficiency of the adaption mechanism. Rather than solely

retraining a new classifier on the remaining clusters post-change detection, future works could focus on refining and enhancing the efficiency of the adaptation mechanism.

Bibliography

[1] Charu C. Aggarwal et al. “A Framework for Clustering Evolving Data Streams”. In: *VLDB*. Morgan Kaufmann, 2003, pp. 81–92.

[2] Ethem Alpaydin. *Introduction to machine learning*. Adaptive computation and machine learning. MIT Press, 2004.

[3] Vladimir Araujo et al. “Entropy-based Stability-Plasticity for Lifelong Learning”. In: *CVPR Workshops*. IEEE, 2022, pp. 3720–3727.

[4] Manuel Baena-Garcıa et al. “Early drift detection method”. In: *Fourth international workshop on knowledge discovery from data streams*. Vol. 6. Citeseer. 2006, pp. 77–86.

[5] Albert Bifet et al. “Efficient data stream classification via probabilistic adaptive windows”. In: *SAC*. ACM, 2013, pp. 801–806.

[6] Albert Bifet et al. “Ensembles of Restricted Hoeffding Trees”. In: *ACM Trans. Intell. Syst. Technol.* 3.2 (2012), 30:1–30:20.

[7] Albert Bifet et al. *Machine Learning for Data Streams with Practical Examples in MOA*. Mar. 2018. ISBN: 9780262037792. DOI: 10.7551/mitpress/10654.001.0001.

[8] Albert Bifet et al. “MOA: Massive Online Analysis”. In: *J. Mach. Learn. Res.* 11 (2010), pp. 1601–1604.

[9] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[10] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[11] Davide Cacciarelli and Murat Kulahci. “A survey on online active learning”. In: *CoRR* abs/2302.08893 (2023).

[12] Feng Cao et al. “Density-Based Clustering over an Evolving Data Stream with Noise”. In: *SDM*. SIAM, 2006, pp. 328–339.

[13] Olivier Chapelle. “Active Learning for Parzen Window Classifier”. In: *AISTATS*. Society for Artificial Intelligence and Statistics, 2005.

[14] David A. Cohn, Les E. Atlas, and Richard E. Ladner. “Improving Generalization with Active Learning”. In: *Mach. Learn.* 15.2 (1994), pp. 201–221.

[15] Thomas M. Cover and Peter E. Hart. “Nearest neighbor pattern classification”. In: *IEEE Trans. Inf. Theory* 13.1 (1967), pp. 21–27.

[16] Gregory Ditzler et al. “Learning in Nonstationary Environments: A Survey”. In: *IEEE Comput. Intell. Mag.* 10.4 (2015), pp. 12–25.

[17] Pedro M. Domingos and Geoff Hulten. “Mining high-speed data streams”. In: *KDD*. ACM, 2000, pp. 71–80.

[18] Hermann Ebbinghaus. “Memory: A contribution to experimental psychology”. In: *Annals of neurosciences* 20.4 (2013), p. 155.

[19] Yifan Fu, Xingquan Zhu, and Bin Li. "A survey on instance selection for active learning". In: *Knowl. Inf. Syst.* 35.2 (2013), pp. 249–283.

[20] Mohamed Medhat Gaber. "Advances in data stream mining". In: *WIREs Data Mining Knowl. Discov.* 2.1 (2012), pp. 79–85.

[21] João Gama, Ricardo Rocha, and Pedro Medas. "Accurate decision trees for mining high-speed data streams". In: *KDD*. ACM, 2003, pp. 523–528.

[22] João Gama et al. "A survey on concept drift adaptation". In: *ACM Comput. Surv.* 46.4 (2014), 44:1–44:37.

[23] João Gama et al. "Learning with Drift Detection". In: *SBIA*. Vol. 3171. Lecture Notes in Computer Science. Springer, 2004, pp. 286–295.

[24] Lukasz Golab and M. Tamer Özsu. "Issues in data stream management". In: *SIGMOD Rec.* 32.2 (2003), pp. 5–14.

[25] Heitor Murilo Gomes et al. "Adaptive random forests for evolving data stream classification". In: *Mach. Learn.* 106.9-10 (2017), pp. 1469–1495.

[26] Stephen Grossberg. "Nonlinear neural networks: Principles, mechanisms, and architectures". In: *Neural Networks* 1.1 (1988), pp. 17–61.

[27] Michael Harries. *Splice-2 Comparative Evaluation: Electricity Pricing*. Technical Report. The University of South Wales, 1999.

[28] Dino Ienco et al. "Clustering Based Active Learning for Evolving Data Streams". In: *Discovery Science*. Vol. 8140. Lecture Notes in Computer Science. Springer, 2013, pp. 79–93.

[29] "KDD-Cup 1999 Data". In: *SIGKDD Explorations*. Accessed: [your access date]. 1999. URL: http://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data.

[30] Mark G. Kelly, David J. Hand, and Niall M. Adams. "The Impact of Changing Populations on Classifier Performance". In: *KDD*. ACM, 1999, pp. 367–371.

[31] Ralf Klinkenberg and Thorsten Joachims. "Detecting Concept Drift with Support Vector Machines". In: *ICML*. Morgan Kaufmann, 2000, pp. 487–494.

[32] J. Zico Kolter and Marcus A. Maloof. "Dynamic Weighted Majority: An Ensemble Method for Drifting Concepts". In: *J. Mach. Learn. Res.* 8 (2007), pp. 2755–2790.

[33] Łukasz Korycki and Bartosz Krawczyk. "Concept drift detection from multi-class imbalanced data streams". In: *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE. 2021, pp. 1068–1079.

[34] Daniel Kottke. *Budget Optimization for Active Learning in Data Streams*. Master's thesis. Available at http://www.daniel.kottke.eu/add/master-thesis.pdf. Oct. 2014.

[35] Daniel Kottke, Georg Krempl, and Myra Spiliopoulou. "Probabilistic Active Learning in Datastreams". In: *IDA*. Vol. 9385. Lecture Notes in Computer Science. Springer, 2015, pp. 145–157.

[36] Bartosz Krawczyk et al. "Ensemble learning for data stream analysis: A survey". In: *Inf. Fusion* 37 (2017), pp. 132–156.

[37] Georg Krempl, Daniel Kottke, and Myra Spiliopoulou. "Probabilistic Active Learning: Towards Combining Versatility, Optimality and Efficiency". In: *Discovery Science*. Vol. 8777. Lecture Notes in Computer Science. Springer, 2014, pp. 168–179.

[38] Georg Krempl et al. “Open challenges for data stream mining research”. In: *SIGKDD Explor.* 16.1 (2014), pp. 1–10.

[39] Patrick Lindstrom, Sarah Jane Delany, and Brian Mac Namee. “Handling Concept Drift in a Text Data Stream Constrained by High Labelling Cost”. In: *FLAIRS Conference*. AAAI Press, 2010.

[40] Sanmin Liu et al. “Online Active Learning for Drifting Data Streams”. In: *IEEE Trans. Neural Networks Learn. Syst.* 34.1 (2023), pp. 186–200.

[41] Viktor Losing, Barbara Hammer, and Heiko Wersing. “KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift”. In: *ICDM*. IEEE Computer Society, 2016, pp. 291–300.

[42] Jie Lu et al. “Learning under concept drift: A review”. In: *IEEE transactions on knowledge and data engineering* 31.12 (2018), pp. 2346–2363.

[43] James MacQueen et al. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[44] Jacob Montiel et al. “Scikit-Multiflow: A Multi-output Streaming Framework”. In: *Journal of Machine Learning Research* 19.72 (2018), pp. 1–5. URL: http://jmlr.org/papers/v19/18-251.html.

[45] S. Muthukrishnan. “Data Streams: Algorithms and Applications”. In: *Found. Trends Theor. Comput. Sci.* 1.2 (2005).

[46] Nikunj C. Oza and Stuart Russell. “Experimental comparisons of online and batch versions of bagging and boosting”. In: *KDD*. ACM, 2001, pp. 359–364.

[47] Martin Scholz and Ralf Klinkenberg. “An ensemble classifier for drifting concepts”. In: *Proceedings of the Second International Workshop on Knowledge Discovery in Data Streams*. Vol. 6. 11. Porto, Portugal. 2005, pp. 53–64.

[48] Burr Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[49] Burr Settles. “Active learning literature survey”. In: (2009).

[50] Burr Settles, Mark Craven, and Soumya Ray. “Multiple-Instance Active Learning”. In: *NIPS*. Curran Associates, Inc., 2007, pp. 1289–1296.

[51] H. Sebastian Seung, Manfred Opper, and Haim Sompolinsky. “Query by Committee”. In: *COLT*. ACM, 1992, pp. 287–294.

[52] Jicheng Shan et al. “Online Active Learning Ensemble Framework for Drifted Data Streams”. In: *IEEE Trans. Neural Networks Learn. Syst.* 30.2 (2019), pp. 486–498.

[53] Claude E. Shannon. “A mathematical theory of communication”. In: *Bell Syst. Tech. J.* 27.3 (1948), pp. 379–423.

[54] Michael Steinbach, Levent Ertöz, and Vipin Kumar. “The challenges of clustering high dimensional data”. In: *New directions in statistical physics: econophysics, bioinformatics, and pattern recognition*. Springer, 2004, pp. 273–309.

[55] W. Nick Street and YongSeog Kim. “A streaming ensemble algorithm (SEA) for large-scale classification”. In: *KDD*. ACM, 2001, pp. 377–382.

[56] W. Nick Street and YongSeog Kim. "A streaming ensemble algorithm (SEA) for large-scale classification". In: *KDD*. ACM, 2001, pp. 377–382.

[57] Alaa Tharwat and Wolfram Schenck. "A Survey on Active Learning: State-of-the-Art, Practical Challenges and Research Directions". In: (2023).

[58] Katrin Tomanek and Udo Hahn. "Semi-Supervised Active Learning for Sequence Labeling". In: *ACL/IJCNLP*. The Association for Computer Linguistics, 2009, pp. 1039–1047.

[59] Alexander Vezhnevets, Joachim M. Buhmann, and Vittorio Ferrari. "Active learning for semantic segmentation with expected change". In: *CVPR*. IEEE Computer Society, 2012, pp. 3162–3169.

[60] Peng Wang, Peng Zhang, and Li Guo. "Mining Multi-Label Data Streams Using Ensemble-Based Active Learning". In: *SDM*. SIAM / Omnipress, 2012, pp. 1131–1140.

[61] Geoffrey I. Webb et al. "Characterizing concept drift". In: *Data Min. Knowl. Discov.* 30.4 (2016), pp. 964–994.

[62] Gerhard Widmer and Miroslav Kubat. "Learning in the Presence of Concept Drift and Hidden Contexts". In: *Mach. Learn.* 23.1 (1996), pp. 69–101.

[63] Gerhard Widmer and Miroslav Kubat. "Learning in the Presence of Concept Drift and Hidden Contexts". In: *Mach. Learn.* 23.1 (1996), pp. 69–101.

[64] Yi Wu et al. "Sampling Strategies for Active Learning in Personal Photo Retrieval". In: *ICME*. IEEE Computer Society, 2006, pp. 529–532.

[65] Wenhua Xu, Fengfei Zhao, and Zhengcai Lu. "Active learning over evolving data streams using paired ensemble framework". In: *ICACI*. IEEE, 2016, pp. 180–185.

[66] Yiming Yang and Jan O. Pedersen. "A Comparative Study on Feature Selection in Text Categorization". In: *ICML*. Morgan Kaufmann, 1997, pp. 412–420.

[67] Chunyong Yin, Shuangshuang Chen, and Zhichao Yin. "Clustering-based Active Learning Classification towards Data Stream". In: *ACM Trans. Intell. Syst. Technol.* 14.2 (2023), 38:1–38:18.

[68] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. "BIRCH: An Efficient Data Clustering Method for Very Large Databases". In: *SIGMOD Conference*. ACM Press, 1996, pp. 103–114.

[69] Ye Zhang, Matthew Lease, and Byron C. Wallace. "Active Discriminative Text Representation Learning". In: *AAAI*. AAAI Press, 2017, pp. 3386–3392.

[70] Xingquan Zhu and Xindong Wu. "Scalable Representative Instance Selection and Ranking". In: *ICPR (3)*. IEEE Computer Society, 2006, pp. 352–355.

[71] Xingquan Zhu et al. "Active Learning from Data Streams". In: *ICDM*. IEEE Computer Society, 2007, pp. 757–762.

[72] Indre Zliobaite. "How good is the Electricity benchmark for evaluating concept drift adaptation". In: *CoRR* abs/1301.3524 (2013).

[73] Indre Zliobaite. "Learning under Concept Drift: an Overview". In: *CoRR* abs/1010.4784 (2010).

[74] Indre Zliobaite et al. "Active Learning With Drifting Streaming Data". In: *IEEE Trans. Neural Networks Learn. Syst.* 25.1 (2014), pp. 27–39.

[75] Indrė Žliobaitė, Mykola Pechenizkiy, and Joao Gama. “An overview of concept drift applications”. In: *Big data analysis: new algorithms for a new society* (2016), pp. 91–114.

A. Appendix

A.1. Accuracy Comparison

Dataset	CORA-EE	EP	SE	SP	OPAL-NA	PEFAL	Zliobaite
Airlines	0.6157	0.6013	**0.6256**	0.6079	0.6253	0.6086	0.6236
ChessBoard	0.5625	**0.5832**	0.4128	0.4580	0.2981	0.4736	0.4498
Covertype	**0.9506**	**0.9506**	0.8389	0.8389	0.8389	0.8389	0.8817
Electricity	**0.8020**	0.7999	0.7659	0.7491	0.7646	0.7728	0.7932
Hyperplane	0.8536	0.8598	0.8525	0.8619	0.8085	**0.9146**	0.8477
PokerHand	0.7171	**0.724**	0.6361	0.6603	0.6361	0.6715	0.6950
RbfGenerator	0.9064	**0.9251**	0.8778	0.8970	0.6270	0.7253	0.8267
SEA	0.8068	0.7854	0.8364	0.8072	**0.8615**	0.8402	0.8599

Table A.1.: Accuracy comparison averaged over all budgets and 30 repetitions