ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

INTERNATIONAL HELLENIC UNIVERSITY
DEPARTMENT OF INFORMATION AND ELECTRONIC ENGINEERING

DIPLOMA THESIS

Instance Selection Algorithms used on Data Streams

Student:
Lazaros Georgiadis
Student ID: 185162

Supervisor:
Stefanos Ougiaroglou

22 January 2025

Title of Dissertation PAPER TITLE HERE
Code of Dissertation 23101
Student's full name Lazaros Georgiadis
Supervisor's full name Stefanos Ougiaroglou
Date of undertaking 27-10-2023
Date of completion 22-01-2025

We hereby affirm the authorship of this paper as well as the acknowledgement and credit of whichever assistance We received in its composition. We have, furthermore, noted the various sources from which We extracted data, ideas, visual or written material, in paraphrase or exact quotation. Moreover, we affirm the exclusive composition of this paper by myself only, for the purpose of it being a dissertation, in the Department of Information and Electronic Engineering of the I.H.U.

The approval of this dissertation by the Department of Information and Electronic Engineering of the International Hellenic University, does not necessarily entail the adoption of the author's views, on behalf of the Department.

Dedication

I dedicate this thesis to those who have supported and encouraged me throughout this journey. To my family, whose patience, understanding, and belief in me have been a constant source of motivation. Your support has made even the most challenging moments manageable.

To my advisor, Stefanos Ougiaroglou, whose guidance and insightful advice have been invaluable in shaping this work. Your expertise helped me navigate this process with greater confidence.

To my friends and colleagues, who have offered words of encouragement, shared their wisdom, and provided much-needed moments of laughter—thank you for making this experience more meaningful.

This work is a reflection of the support and kindness I have been fortunate to receive, and I am deeply grateful to all who have been part of this journey.

Summary

The field of supervised learning, particularly in the context of classification tasks, has seen significant advances over the years. Among the most widely used methods, the k-Nearest Neighbors (k-NN) algorithm stands out due to its simplicity and effectiveness. However, its reliance on storing and comparing all training instances poses computational and memory challenges, especially when dealing with massive, continuous, and potentially infinite data streams.

In such scenarios, traditional batch learning techniques fail to address the unique challenges faced when dealing with data streams, such as high data arrival rates, limited memory, and the need for real-time processing. To overcome these challenges, data reduction techniques, including prototype selection, condensing, prototype generation, and editing, have emerged as key solutions. These techniques aim to minimize memory usage and computational overhead while preserving classification accuracy.

This thesis focuses on instance selection algorithms for data streams, exploring their applicability in scenarios without concept drift. Several algorithms, including IB1, IB2, IB3, TWF, LWF, PECS, AIB2, and DRHC, are implemented and experimentally evaluated on benchmark datasets. The study employs a "test-then-train" evaluation methodology and incorporates statistical tests such as the Wilcoxon Signed-Rank Test and the Friedman Test to ensure rigorous comparison of algorithm performance.

Abstract

Data streams are an increasingly important area of study due to their applications in fields such as IoT, finance, and healthcare, where large-scale and continuously evolving data must be processed in real-time. Traditional machine learning techniques, including classification algorithms such as k-Nearest Neighbors (k-NN), often fail to meet the memory and computational constraints imposed by data streams. Instance selection algorithms, which aim to reduce data while preserving classification accuracy, provide a promising solution to these challenges.
This thesis explores instance selection techniques for data streams, with a particular focus on data reduction methods designed for streams without concept drift. Algorithms such as IB1, IB2, IB3, TWF, LWF, PECS, AIB2, and DRHC are implemented, compared, and evaluated in terms of classification accuracy and computational time. The ”test-then-train” methodology is employed to simulate real-world streaming scenarios, and experimental studies are conducted on a variety of datasets.
The results highlight the trade-offs between data reduction, memory usage, and classification performance. Statistical analysis using the Wilcoxon Signed-Rank and Friedman tests is performed to ensure the reliability of the findings.
Future work includes addressing concept drift, exploring hybrid reduction methods, and adapting these techniques to other machine learning models. The findings provide a foundation for further research in real-time data processing and scalable machine learning.

Keywords

Instance Selection Algorithms, Data Streams, Prototype Selection, Memory-Efficient Classification, Stationary Data Streams, Condensing Techniques, Time-Weighted Forgetting (TWF), k-Nearest Neighbors (k-NN), Data Reduction Methods, Incremental Learning

General Terms

Supervised Learning, Machine Learning, Classification Algorithms, Data Stream Processing, Scalable Machine Learning, Real-Time Analytics, Concept Drift, Big Data Analytics, Adaptive Algorithms

Contents

Chapter 1

Introduction

1.1 Classification (Supervised Learning)

Classification is one of the most fundamental and widely studied problems in the domain of supervised machine learning. In its essence, classification involves assigning a label to a given input instance, or matching that input to a category, based on a predefined or evolving set of labels [1], [2]. Formally, given a labeled dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where $x_i \in \mathcal{X}$ represents the feature vector of the i-th instance and $y_i \in \mathcal{Y}$ is the corresponding label, the goal of classification is to create a function $f : \mathcal{X} \to \mathcal{Y}$ that can predict the label y for a new, unseen instance x [3].
Supervised learning differs from other types of learning paradigms, such as unsupervised and reinforcement learning, in that it requires the presence of labeled data for training [4], [5]. The term "supervised" emphasizes the existence of a supervisor, in this case, the set of labels, which guide the learning process. Classification tasks are becoming more commonly used and have numerous real-world applications, including but not limited to [6]:

- **Spam Detection:** Classifying emails as spam or not spam based on their content.
- **Medical Diagnosis:** Predicting whether a patient has a specific disease based on medical test results .
- **Image Recognition:** Identifying objects or scenes in images, such as recognizing handwritten digits or detecting faces.
- **Sentiment Analysis:** Determining the sentiment (positive, negative, or neutral) of a given text, such as a product review or a tweet.

The Components of Classification

The process of classification generally involves several key components:

- **Feature Space:** The feature space $\mathcal{X}$ represents the domain of input instances, where each instance is typically represented as a vector of features. For example, in an image classification problem, the features could include the bitmap values representing the image's pixels.
- **Labels:** The set of possible labels $\mathcal{Y}$ is often finite and discrete. For binary classification tasks, usually $\mathcal{Y} = \{0, 1\}$. For multi-class problems, $\mathcal{Y}$ can consist of multiple categories.

- **Learning Algorithm:** The learning algorithm is the mechanism through which the function f is inferred from the training data $\mathcal{D}$. Popular algorithms include decision trees, support vector machines, neural networks, and instance-based methods such as k-Nearest Neighbors.
- **Evaluation Metrics:** Classification models are evaluated based on metrics such as accuracy, precision, recall and F1-score. These metrics measure the performance of the classifier on unseen data.

Types of Classification Tasks

Classification tasks can be broadly categorized based on the number of target labels and the structure of the problem. These categories capture the diversity of real-world scenarios and guide the development of appropriate modeling techniques [7], [8].

Binary Classification represents the simplest form of classification, where the objective is to distinguish between two classes. For example, binary classification is used to determine whether an email is spam or not spam or to diagnose the presence or absence of a disease. This task is widely applicable in domains where decision-making revolves around a clear yes-or-no outcome.

Multi-Class Classification extends the problem to cases where the target variable can belong to more than two categories. A classic example is identifying the species of a flower, such as *setosa*, *versicolor*, or *virginica*. Another common application is recognizing handwritten digits ranging from 0 to 9. Multi-class classification introduces additional complexity, as models must differentiate between multiple distinct outcomes.

Multi-Label Classification goes further by allowing multiple labels to be assigned to a single instance. For instance, a news article might simultaneously be classified under ”politics” and ”economy.” This type of classification is particularly relevant in applications like text categorization, where instances often belong to overlapping categories.

Each of these classification tasks presents unique challenges and opportunities, requiring tailored approaches to achieve optimal performance in different contexts.

Challenges in Classification

Classification, while simple in its formulation, encounters several challenges in practical applications. One major challenge is high dimensionality [9]. Many real-world datasets, such as those involving text or image data, contain a large number of features, making it difficult to learn an efficient model due to the curse of dimensionality. This can significantly impact the performance and computational feasibility of classification algorithms.

Another common issue is noisy data. Real-world data is often noisy, incomplete, or mislabeled, which can adversely affect the performance of classification models. Noise in the data can lead to errors in pattern recognition and reduce the overall accuracy of the model. However, there are techniques available to mitigate the impact of noise, such as preprocessing, robust algorithms, and noise-tolerant learning methods.

Overfitting is also a significant challenge in classification. This occurs when a model learns the noise and specific intricacies of the training data rather than capturing the underlying patterns [10]. As a result, the model performs well on the training set but generalizes poorly to unseen data. Addressing overfitting requires techniques such as regularization, pruning, and the use of cross-validation to ensure

better generalization.
Scalability poses another critical challenge, especially with the advent of big data [8]. Classification algorithms must now be able to handle datasets containing millions or even billions of instances. This requires the development of efficient computational methods and scalable architectures to process such large datasets within reasonable timeframes.
Finally, evolving data presents a unique challenge in dynamic environments like data streams. In such settings, the data distribution can change over time, a phenomenon known as concept drift [11]. This requires classification models to adapt and learn incrementally, ensuring that they remain accurate and relevant in the face of changing patterns.
Addressing these challenges is crucial for developing robust and efficient classification models that can perform well across a variety of real-world applications.

The Importance of Classification in Data Streams

While classification is well-established in static datasets, its application to data streams presents unique challenges due to the continuous nature of the data [12]. This thesis focuses on the development and evaluation of instance selection techniques specifically tailored for classification tasks in streaming environments, where memory and computational constraints must be balanced with the need for accurate and timely predictions.

Structure and Relevance

Classification serves as the foundational task upon which this thesis is built. The subsequent sections delve into specific classification methodologies, including k-Nearest Neighbors, and adapt these concepts to the context of data streams and instance selection. By addressing the challenges of classification in streaming data, this thesis contributes to the broader field of scalable and adaptive machine learning.

1.2 k-Nearest Neighbors (k-NN) Classification

The *k-Nearest Neighbors (k-NN)* algorithm is one of the most straightforward yet effective methods in supervised learning [13]. It operates on the principle of proximity, classifying a query instance based on the labels of its nearest neighbors in the feature space. Unlike many learning algorithms that construct an explicit model during training, k-NN is an instance-based or "lazy" learning method [1], [14]. It retains the entire training dataset and defers computation until classification is required.
In k-NN, given a dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where x_i represents the feature vector and y_i the corresponding class label, the classification of a new query instance q proceeds as follows:

1. Compute the distance between q and all instances in the dataset D.
2. Identify the k instances with the smallest distances to q.
3. Assign q to the class most frequently occurring among these k nearest neighbors.

The choice of distance metric is crucial to the algorithm's performance [15]. Commonly used metrics include:

- **Euclidean Distance**, which is the straight-line distance between two points, often suitable for numerical features:

$$d(x, q) = \sqrt{\sum_{i=1}^{m} (x_i - q_i)^2}$$

.

- **Manhattan Distance**, which sums the absolute differences across dimensions:

$$d(x, q) = \sum_{i=1}^{m} |x_i - q_i|$$

.

- **Cosine Similarity**, used in text and high-dimensional data, which measures the cosine of the angle between two vectors:

$$\text{similarity}(x, q) = \frac{x \cdot q}{\|x\| \|q\|}$$

.

The parameter k, the number of neighbors considered, significantly impacts the algorithm [16]. A small k (e.g., $k = 1$) leads to a model highly sensitive to noise, as the classification depends on a single data point. A larger k smoothens decision boundaries by incorporating more neighbors but can dilute local patterns in the data. Optimal values for k are typically determined empirically using techniques like cross-validation.

One of k-NN's notable strengths is its simplicity. It does not assume an underlying data distribution, making it versatile across various domains [14]. Moreover, it is highly interpretable; the reasoning behind a classification can be explained by examining the nearest neighbors of a query point. The algorithm is also flexible, easily extending to regression tasks where predictions are based on the average of the k nearest neighbors rather than majority voting.

However, k-NN has several limitations. The algorithm is computationally expensive during inference because it requires calculating distances to all instances in the dataset [17]. This challenge becomes particularly pronounced for large datasets, where the cost of real-time distance computation can be prohibitive. Additionally, storing the entire training dataset imposes significant memory requirements, making k-NN impractical for applications with limited resources.

Furthermore, k-NN is highly sensitive to irrelevant or redundant features, which can distort distance calculations and degrade its performance. Preprocessing techniques such as feature selection or scaling are often necessary to mitigate these issues.

Adapting k-NN to data streams introduces additional challenges [17]. In a streaming environment, data arrives continuously and potentially indefinitely. The need for real-time classification requires efficient distance computations, while memory constraints necessitate discarding outdated or redundant instances. Incremental instance selection techniques are therefore crucial [17]. These methods aim to maintain a small, representative subset of the data that balances memory efficiency with classification accuracy. For instance, prototype selection algorithms condense the dataset by retaining only the most representative instances, while editing techniques remove noisy or irrelevant data points .

1.3 Data Streams

Data streams are a distinctive type of data source, characterized by their continuous, high-speed, and potentially never-ending nature [18]. Unlike traditional datasets that are static and finite, data streams are generated in real-time by applications such as sensor networks, financial markets, online social media, and Internet of Things (IoT) devices [19]. These applications produce a flow of data points that must be processed incrementally, as revisiting or storing the entire stream is often impractical due to memory and time constraints.

The defining characteristics of data streams make them fundamentally different from static datasets. First, data streams exhibit continuous arrival, where data is generated in an ongoing, unbounded manner, making it impossible to assume a fixed dataset size. Second, they are defined by high velocity, as streams are generated at high speed, requiring algorithms to process each instance quickly to avoid bottlenecks or delays. Third, data streams can be infinite in size, meaning they can grow indefinitely. This necessitates the use of techniques that limit memory usage and computational overhead. Lastly, data streams often exhibit evolving distributions, where the underlying data distribution changes over time—a phenomenon known as concept drift [20]. Although this thesis focuses on stationary data streams, which are streams without concept drift, it is essential to acknowledge the dynamic nature of many real-world data streams.

Processing data streams introduces unique challenges that traditional batch-processing methods are not equipped to handle [12]. In batch processing, algorithms assume that the entire dataset is available before training begins. This assumption allows for multiple passes over the data and sophisticated optimization techniques. In contrast, data stream processing operates under strict constraints:

- Algorithms must be **incremental**, updating their models with each new instance without revisiting past data.
- Memory usage must remain **bounded**, as storing the entire stream is infeasible.
- The system must provide **real-time responses**, making efficiency a critical requirement.

To address the constraints of data streams, algorithms often employ techniques such as instance selection, windowing, and summarization [19]. Instance selection focuses on maintaining a small yet representative subset of the stream, balancing memory efficiency with accuracy. Sliding window models take a different approach, retaining only the most recent data under the assumption that older instances are less relevant [12]. Summarization techniques, on the other hand, create compact data structures like sketches or histograms to approximate the stream's distribution without the need to store individual instances.

Data streams have become increasingly important due to their widespread use in modern applications. For instance, in financial markets, stock prices and transaction data streams must be analyzed in real-time to detect trends and anomalies [16]. Similarly, in IoT systems, sensors continuously generate environmental data, such as temperature, humidity, and motion, which must either be processed locally or transmitted to a central system for analysis. On social media platforms, user interactions like likes, shares, and comments form a constant stream of data that is analyzed to provide personalized recommendations and perform sentiment analysis. These examples highlight the critical need for efficient data stream processing to derive timely and actionable insights [19].

Despite the challenges posed by data streams, the ability to process them efficiently is essential. Traditional machine learning algorithms, which rely on the assumption of a static, finite dataset, are often

ill-suited for this purpose. Consequently, specialized algorithms and frameworks have been developed for streaming data, focusing on scalability and adaptability to meet the unique demands of such applications.
This thesis focuses on stationary datasets which we split and feed in batches to simulate a buffered data stream, where the underlying data distribution remains constant over time. This assumption simplifies the problem by removing the need to address concept drift, allowing the study to concentrate on efficient instance selection techniques. By adapting methods like prototype selection and data reduction to the constraints of streaming environments, it is possible to achieve high classification performance while keeping memory and computational costs low.
Data streams present both an opportunity and a challenge for modern machine learning. They demand innovative approaches that prioritize efficiency and adaptability without compromising accuracy. The techniques explored in this thesis aim to address these needs, contributing to the broader field of scalable and real-time data processing.

1.4 Classification on Data Streams

Classification on data streams presents a unique set of challenges compared to traditional classification tasks on static datasets. The goal remains the same: assigning labels to instances based on patterns learned from the data. However, the nature of data streams, with their continuous and unbounded arrival of data, introduces constraints that require significant modifications to conventional classification algorithms. These constraints arise due to the need for incremental processing, limited memory usage, and real-time decision-making.
In traditional classification settings, algorithms assume access to the entire dataset at once. This allows for multiple passes over the data during the training process, enabling the use of computationally expensive optimization techniques and comprehensive model evaluation. In contrast, data stream classification must operate incrementally. Models are updated instance by instance, with no possibility of revisiting previous data [21]. This requires algorithms to process each instance as it arrives, integrate it into the existing model, and discard it, if need be, to conserve memory.
Memory efficiency is another critical consideration. Since data streams are potentially infinite, it is infeasible to store all incoming instances [16]. Traditional batch classifiers rely on the availability of the complete dataset for feature extraction, model training, and validation. In the data stream setting, only a small subset of the data can be retained at any time, requiring techniques such as instance selection, windowing, or summarization to manage memory while preserving classification accuracy.
Real-time classification is another fundamental requirement of data streams. Many applications, such as the aforementioned ones, necessitate immediate decision-making. Algorithms must balance the computational cost of updating the model with the time-sensitive demands of the application. Achieving this balance often involves trade-offs between accuracy, efficiency, and adaptability [22].
Another challenge specific to data streams is concept drift, a phenomenon in which the underlying data distribution changes over time. For instance, in a streaming application for detecting spam emails, new spamming techniques may emerge, altering the patterns previously observed in the data. Although this thesis focuses on stationary data streams, where the distribution remains constant, it is important to recognize that concept drift is a defining characteristic of many real-world streams. The strategies developed for stationary streams can often serve as a foundation for extending algorithms to dynamic scenarios.

Despite these challenges, classification on data streams has been successfully applied across numerous domains [16]. In online recommendation systems, for example, user preferences and interactions form a continuous stream of data, enabling the system to adapt recommendations in real-time. Similarly, IoT applications use data stream classification to monitor environmental parameters or detect anomalies in sensor networks. In each of these cases, the ability to perform accurate and efficient classification directly impacts the utility of the system.
To address the unique requirements of data streams, researchers have developed a variety of approaches for adapting classification algorithms. Incremental learning methods update the model incrementally as new data arrives, ensuring that the classifier evolves over time. Instance selection techniques, such as prototype selection and data reduction, aim to maintain a compact and representative dataset, reducing memory usage without significantly compromising accuracy. Ensemble methods, which combine multiple models to make predictions, are also popular due to their adaptability and robustness, with a cost to the required processing power.

1.5 Data Reduction

Data reduction is a fundamental concept in machine learning and data processing, aimed at reducing the size of a dataset while preserving the essential information necessary for accurate analysis or prediction. In the context of supervised learning, data reduction techniques are particularly useful for mitigating the challenges associated with large-scale datasets, such as computational inefficiency, high memory usage, and sensitivity to noise [23]. By condensing the dataset into a smaller, more manageable form, these techniques enable faster computations and improve the scalability of machine learning algorithms.
The need for data reduction arises from the growing volume and complexity of data in modern applications. In fields such as healthcare, finance, and IoT, data is generated at an unprecedented scale, often making it infeasible to store or process the entire dataset [24]. Moreover, many datasets contain redundant or irrelevant instances that do not contribute significantly to the learning process. Data reduction addresses these issues by focusing on the most informative and representative parts of the dataset.
The most usual forms of data reduction, can be broadly categorized into four main approaches: prototype selection, condensing, prototype generation, and editing, which will be discussed in detail below. Each of these approaches addresses a specific aspect of the dataset and contributes to improving the efficiency and effectiveness of classification tasks.
This work leverages data reduction techniques to develop instance selection methods tailored to non concept drifting data streams. By leveraging approaches such as prototype selection and condensing, the goal is to address the unique challenges of streaming environments while preserving the effectiveness of classification algorithms. The techniques explored, contribute to the broader field of scalable and efficient data processing, providing insights into their application in both streaming and static contexts.

1.6 Data Reduction for Data Streams

Applying data reduction techniques to data streams introduces unique challenges and opportunities. While traditional data reduction focuses on static datasets, where the entire dataset is available upfront, data streams operate under vastly different constraints. These constraints, such as those discussed above, necessitate the adaptation of data reduction methods to incremental and online settings [22]. As a result, data reduction for data streams requires algorithms that can effectively maintain a compact and representative subset of the data in real-time, without revisiting past instances.

Incremental Instance Selection

Incremental instance selection is the cornerstone of data reduction on data streams [25]. In this approach, a subset of representative instances is maintained dynamically as new data arrives. Incoming instances are evaluated in real-time to determine their relevance, often based on their contribution to decision boundaries or their similarity to existing instances. Irrelevant or redundant instances are discarded to keep the memory footprint small, while informative instances are retained to ensure high classification accuracy. For example, methods like prototype selection can be adapted to streaming scenarios by continuously updating the set of prototypes based on the evolving data.

Sliding Windows and Forgetting Mechanisms

Another common strategy in data reduction for data streams is the use of sliding windows or forgetting mechanisms. Sliding window models retain only the most recent instances, under the assumption that older data is less relevant to the current task. For example, a fixed-size window may keep the last n instances, discarding the oldest instance as new data arrives [22]. Alternatively, time-decay functions can be applied, where the relevance of an instance decreases over time, effectively "forgetting" older data. These approaches are particularly useful for managing the unbounded nature of data streams, as they limit the memory usage without explicitly reducing the dataset size.

Summarization and Sketching Techniques

In addition to instance selection, summarization techniques are widely used in data reduction for data streams. Summarization involves creating compact representations of the data, such as sketches, histograms, or cluster centroids, which capture the key properties of the stream without storing individual instances. For example, clustering algorithms can dynamically update cluster centroids to represent groups of similar instances in the stream. These summaries can then be used for classification tasks, often at the cost of some loss in granularity or precision. Summarization techniques are particularly effective when the primary goal is to extract aggregate information or trends from the stream, rather than classify individual instances [22].

The primary goal of data reduction in streaming environments is to ensure that the reduced dataset remains small enough to fit within memory constraints while still preserving the essential information needed for accurate classification. However, achieving this goal is not straightforward. Unlike in static datasets, where all instances can be analyzed simultaneously to identify redundancies or patterns, streaming data must be processed incrementally. Each incoming instance must be either retained, discarded, or summarized based on its relevance to the learning task.

1.7 Scope of the Thesis

The ever-expanding prevalence of data-driven applications in various fields, ranging from healthcare to transportation, has resulted in an exponential increase in the generation of data streams. Unlike static datasets, which are finite and accessible in their entirety, data streams are characterized by their continuous, high-speed, and unbounded nature. This thesis focuses specifically on addressing the challenges posed by such data streams in the context of supervised machine learning classification tasks, particularly under stationary data stream conditions.

In machine learning, classification is one of the most fundamental tasks, involving the assignment of labels to incoming instances based on patterns learned from previously observed data. While traditional classification algorithms are designed to operate on static datasets, they are often unsuitable for streaming environments due to constraints on memory, computational power, and real-time decision-making requirements. Moreover, the unbounded nature of data streams makes it impractical to store and process the entire dataset, necessitating the development of memory-efficient and computationally lightweight solutions.

This thesis narrows its focus to stationary data streams, where the underlying data distribution remains constant over time. This assumption simplifies the problem by removing the complexities introduced by concept drift, which is a phenomenon where the data distribution changes dynamically. While concept drift represents an important challenge in real-world applications, the decision to exclude it allows for a deeper investigation into efficient instance selection techniques tailored to stationary environments. The selected scope ensures that the study concentrates on the fundamental aspects of data stream processing, laying a robust foundation for potential extensions to more dynamic scenarios in future research.

The research presented in this thesis is specifically centered on instance selection algorithms. Instance selection involves maintaining a small, representative subset of data that is sufficient to perform accurate classification while adhering to the constraints of memory and computational efficiency. This approach addresses the primary limitations of traditional methods such as k-Nearest Neighbors (k-NN), which require the storage of all observed data and are thus unsuitable for real-time applications. The algorithms developed and evaluated in this work are designed to operate incrementally, enabling the system to adapt to incoming data in real time without revisiting past instances.

Furthermore, the thesis explores the application of data reduction techniques, such as prototype selection and condensing, in the context of stationary data streams. These techniques aim to reduce the size of the training dataset while preserving its essential characteristics, thereby achieving a balance between memory usage and classification performance. By focusing on these approaches, the scope of this research addresses the unique challenges of data stream environments, such as limited storage capacity and the need for real-time processing.

It is important to note that while the scope of this thesis is limited to stationary data streams, the insights and methodologies developed here are intended to serve as a foundation for future extensions. These extensions may include dynamic environments with concept drift or the application of instance selection techniques to other machine learning tasks, such as regression or clustering. The choice to concentrate on stationary data streams represents a deliberate decision to provide a focused and in-depth exploration of the challenges and opportunities associated with memory-efficient classification in streaming scenarios.

In summary, the scope of this thesis is defined by its emphasis on the development and evaluation of instance selection algorithms for supervised classification tasks in stationary data streams. By addressing

key challenges such as memory constraints, computational efficiency, and real-time decision-making, this work contributes to the broader field of scalable and adaptive machine learning. The findings presented in this thesis aim to bridge the gap between traditional batch-learning methods and the unique requirements of data stream environments, offering a pathway for future research in this dynamic and rapidly evolving domain.

1.8 Contributions

This thesis contributes to the advancement of instance selection techniques for classification in stationary data streams, addressing key challenges such as memory constraints, computational efficiency, and classification accuracy. The research is conducted using both synthetic datasets and real-world datasets, ensuring a comprehensive evaluation of the proposed methods.

One of the core contributions of this work is the adaptation and refinement of existing instance selection algorithms, including IB1, IB2, IB3, Time-Weighted Forgetting (TWF), Locally-Weighted Forgetting (LWF), Prediction Error Context Switching (PECS), Dynamic Reduction through Homogeneous Clusters (DRHC) and Abstraction IB2(AIB2). These algorithms are specifically tailored for stationary data streams, enabling incremental and efficient instance selection while maintaining high classification performance. Each algorithm is analyzed and implemented to address the challenges of high-velocity and unbounded data streams.

The thesis also develops a unified evaluation framework that incorporates key performance metrics such as classification accuracy, memory usage, and runtime efficiency. In addition, statistical analyses using techniques like the Wilcoxon Signed-Rank Test and the Friedman Test are employed to rigorously compare algorithm performance. This methodological rigor provides a robust foundation for evaluating the trade-offs between different instance selection techniques.

Furthermore, this research advances the theoretical understanding of data reduction techniques by applying methods such as prototype selection and condensing to streaming environments. These techniques are adapted to ensure memory efficiency and incremental processing, allowing the algorithms to meet the constraints of real-time applications. Insights gained from this exploration contribute to the broader field of scalable and adaptive machine learning.

Finally, this thesis lays the groundwork for future research by identifying potential extensions to handle concept drift and to apply instance selection techniques to other machine learning tasks, such as regression and clustering. While the focus remains on stationary data streams, the findings are expected to serve as a foundation for addressing dynamic and more complex data scenarios.

1.9 Motivation

The rapid expansion of data-driven systems across diverse industries has led to an explosion of data streams, which are generated continuously in real-time. From satellite imagery to cybersecurity logs, these streams represent a rich source of information but also present significant challenges to machine learning systems. Unlike static datasets, data streams are continuous, high-velocity, and indefinitely extensible, making traditional algorithms unsuitable for their processing. The need to address the constraints of memory, computational power, and real-time decision-making forms a key motivation for this thesis.

One major source of data streams is the field of transportation systems. With the advent of connected vehicles and smart cities, streams of data from GPS sensors, traffic cameras, and vehicle-to-infrastructure communications must be analyzed in real-time. Applications such as dynamic traffic management, autonomous driving, and real-time navigation require efficient classification methods to process this data and make rapid decisions.

Another motivating example is seen in disaster response and environmental monitoring. Modern systems use satellite-based imaging, seismic sensors, and weather station data to continuously monitor natural phenomena. Streams of data from these sources enable applications such as early detection of forest fires, tracking the spread of hurricanes, or predicting earthquakes. Efficient data reduction techniques are vital in such scenarios, as the volume of data generated by these systems often exceeds the capacity of current processing capabilities. Real-time classification of critical events in the stream can make the difference between proactive and reactive responses.

In the healthcare sector, streaming data is increasingly being utilized to monitor patients in real time. Continuous data from wearable devices, such as heart rate monitors or glucose sensors, provides a wealth of information for detecting anomalies and predicting health risks. For instance, classifying irregular heart rhythms from streaming electrocardiogram (ECG) data can enable early intervention and potentially save lives. However, the need to process this data in real-time while maintaining high accuracy presents significant challenges, particularly in terms of balancing memory efficiency and computational demands that become more apparent when dealing with wearable devices.

Despite these real-world applications, many existing machine learning algorithms are poorly equipped to handle the constraints of data streams. Traditional classification methods, such as k-Nearest Neighbors (k-NN), require storing the entire dataset and performing computationally expensive operations, making them infeasible for streaming environments. Instance selection techniques offer a promising solution by dynamically maintaining a small but representative subset of the data. However, much of the research in this area has focused on static datasets, leaving a significant gap in the understanding and development of methods suitable for streaming scenarios.

This thesis is motivated by the dual need to address practical challenges in data stream classification and advance the theoretical understanding of data reduction techniques. By exploring instance selection methods for stationary data streams, the goal is to bridge the gap between scalability and accuracy, enabling machine learning systems to operate effectively in constrained environments. Moreover, this work seeks to evaluate these techniques not only on synthetic datasets but also in the context of real-world streams, ensuring that the insights gained are both rigorous and applicable.

In summary, the motivation for this thesis stems from the growing prevalence of data streams in critical applications such as transportation, disaster response, healthcare, and energy management. By developing scalable, memory-efficient instance selection techniques, this research aims to address the challenges of streaming data while contributing to the broader field of machine learning. The findings of this work have the potential to advance both theoretical and practical aspects of real-time data processing, paving the way for future innovations in this dynamic area.

1.10 Thesis Organization

This thesis is structured to provide a comprehensive exploration of instance selection techniques for stationary data streams, with a focus on addressing the challenges of memory efficiency, computational constraints, and real-time processing. Each chapter is designed to build on the insights and

concepts introduced in the preceding sections, creating a cohesive narrative that leads from theoretical foundations to practical applications and experimental results. Below is an outline of the organization of this thesis:

Chapter 1: Introduction

The first chapter introduces the motivation and scope of this thesis, highlighting the growing importance of data streams and the challenges they pose for machine learning systems. It outlines the focus on stationary data streams and establishes the research objectives, providing a foundation for the work that follows.

Chapter 2: Data Reduction Techniques for Data Streams

This chapter reviews the state of the art in data reduction, with an emphasis on instance selection methods. It categorizes these techniques into those designed for static datasets and those adapted for streaming environments, discussing their strengths, limitations, and relevance to stationary data streams. The chapter also identifies key research gaps, laying the groundwork for the proposed methodologies.

Chapter 3: Methodology and Implementation

The third chapter details the methodologies employed in this thesis, including the adaptation and implementation of instance selection algorithms for stationary data streams. It includes theoretical descriptions, pseudocode, and practical considerations for deploying these techniques in real-world streaming scenarios. The focus is on balancing memory efficiency and classification performance.

Chapter 4: Experimental Setup

This chapter describes the experimental design used to evaluate the performance of instance selection algorithms. It includes details about the datasets, both synthetic and real-world, as well as the evaluation metrics and methodologies employed. The ”test-then-train” framework is introduced as the basis for simulating real-time data stream classification.

Chapter 5: Experimental Results and Discussion

In this chapter, the results of the experiments are presented and analyzed. The performance of the instance selection techniques is compared across various metrics, including accuracy, memory usage, and computational efficiency. Statistical tests, such as the Wilcoxon Signed-Rank and Friedman tests, are used to validate the findings. A detailed discussion highlights the implications of the results and provides insights into the trade-offs between different approaches.

Chapter 6: Conclusion and Future Work

The final chapter summarizes the findings of this thesis, reflecting on the research objectives and how they were achieved. It discusses the limitations of the current work and proposes directions for future

research, including the extension of the techniques to dynamic data streams with concept drift and their adaptation to other machine learning tasks.

Chapter 2

Literature Review

2.1 Overview of Data Stream Processing

Data stream processing has gained substantial attention over the past few decades as the proliferation of applications producing continuous, high-velocity data streams has reshaped the landscape of data analytics. Unlike traditional batch processing systems, where data is static, finite, and available in its entirety, data streams are inherently dynamic, unbounded, and produced at high speeds [19]. Examples of data streams include real-time sensor outputs in Internet of Things (IoT) applications, clickstream data from web applications, transaction logs in financial systems, and live data feeds from social media platforms.

The challenges associated with data streams stem from their unique characteristics. First, their infinite nature makes it infeasible to store the entire dataset for analysis. Second, the high velocity at which data streams are generated demands real-time processing capabilities to ensure timely decision-making. Third, data streams are often subject to varying distributions over time, requiring algorithms that can adapt to changing patterns without explicit retraining.

Several frameworks have been developed to handle these challenges, providing the infrastructure necessary for data stream processing. Tools like Apache Kafka, Apache Flink, and Spark Streaming have become integral to real-time analytics systems. These frameworks prioritize key aspects such as low latency, fault tolerance, and scalability [26]. Beyond the infrastructure, the theoretical foundation for stream processing involves designing algorithms that are memory-efficient, computationally lightweight, and capable of incremental updates. These properties are particularly important for machine learning tasks on data streams, where algorithms must balance accuracy, efficiency, and adaptability.

2.2 Data Reduction Techniques for Classification

Classification tasks on data streams present unique challenges due to the volume and velocity of incoming data. In traditional machine learning settings, classification algorithms operate on finite datasets, enabling exhaustive training and evaluation. However, in data stream environments, the continuous nature of data necessitates real-time classification while simultaneously addressing memory and computational constraints. This has spurred the development of data reduction techniques, which aim to distill the most relevant information from the data stream for effective classification.

Data reduction techniques can be broadly categorized as follows:

2.2.1 Feature Reduction

Feature reduction techniques focus on reducing the dimensionality of the data by selecting or extracting the most informative features. These methods aim to alleviate the ”curse of dimensionality,” which often affects classification accuracy and computational efficiency in high-dimensional datasets. Feature selection techniques identify a subset of existing features that contribute most to the classification task, while feature extraction methods create new features by transforming the original feature space. Principal Component Analysis (PCA) and Autoencoders are common examples of feature extraction techniques.

2.2.2 Instance Selection

Instance selection methods operate on the premise that not all data points are equally important for classification. By identifying and retaining only a representative subset of instances, these methods reduce the size of the dataset without significantly compromising classification accuracy. Instance selection is particularly relevant for data streams, as it allows algorithms to operate within memory constraints while preserving the essential characteristics of the data.

2.2.3 Prototype Generation

Prototype generation goes beyond instance selection by synthesizing new instances that summarize the data's distribution. Unlike instance selection, which retains actual data points, prototype generation methods create artificial data points that serve as representatives of the original dataset. These methods are often used in conjunction with instance-based learning algorithms such as k-Nearest Neighbors (k-NN) to improve classification efficiency [17].

2.2.4 Prototype Selection, Condensing, Prototype Generation, and Editing

The techniques of prototype selection, condensing, prototype generation, and editing form the backbone of instance-based learning, particularly in the context of data streams. These methods aim to address the constraints of memory and computational efficiency while preserving classification accuracy in streaming environments. Each approach has unique methodologies and applications , as discussed below.

Prototype Selection

Prototype selection aims to identify a subset of instances from the original dataset that can serve as representatives of the entire dataset. These selected prototypes are chosen based on their ability to capture the diversity and structure of the data while minimizing redundancy. This approach reduces memory usage and computational overhead while maintaining classification performance. However, selecting an optimal subset of prototypes is a challenging task that often involves trade-offs between accuracy and efficiency [25].

Condensing

Condensing is a specific form of prototype selection that focuses on minimizing redundancy in the dataset. The goal is to retain only the instances that are critical for defining the decision boundaries of the classifier. For instance, instances that lie near the boundaries between classes are typically more informative than those located deep within a single class. By condensing the dataset, it is possible to reduce its size significantly while preserving the ability of the classifier to make accurate predictions. Condensing techniques are particularly useful for instance-based learning algorithms such as k-NN, which are sensitive to the size and quality of the dataset [17].

Prototype Generation

Unlike prototype selection and condensing, which rely on selecting instances from the original dataset, prototype generation creates synthetic instances that summarize the data. These synthetic prototypes are generated by combining or interpolating existing instances to capture the key patterns and structures in the dataset. For example, cluster centroids or averages of neighboring instances can be used as prototypes. Prototype generation is especially useful when the dataset contains noise or outliers, as the generated prototypes can smooth out these irregularities and provide a cleaner representation of the data. However, the process of generating prototypes introduces additional computational complexity [27].

Editing

Editing focuses on improving the quality of the dataset by removing noisy, mislabeled, or redundant instances. These instances can distort the decision boundaries of a classifier and degrade its performance. Editing techniques aim to enhance the dataset by retaining only the instances that contribute positively to the classification process. Editing is often used in conjunction with prototype selection or condensing to create a smaller, more reliable dataset [17].

Applications to Data Streams

The application of these techniques to data streams introduces unique challenges. Unlike static datasets, where the entire dataset is available for processing, data streams require incremental and real-time operations [19]. Prototype selection, condensing, and editing must be adapted to dynamically update the representative subset of data as new instances arrive. For example:

- Sliding window models can integrate prototype selection by retaining only the most recent and relevant data points [28].
- Forgetting mechanisms, such as those employed in time-weighted forgetting (TWF), ensure that older and less relevant instances are discarded.
- Clustering techniques used in prototype generation can be updated dynamically to represent changes in the data distribution over time.

Classification on data streams relies heavily on these techniques to balance memory efficiency with prediction accuracy [19]. Instance selection methods are particularly valuable in scenarios where data

streams are stationary, as they allow the system to maintain a compact and representative training set without revisiting older data. These methods serve as a foundation for real-time classification tasks in streaming environments such as IoT systems, financial markets, and sensor networks.
By integrating instance selection methods into the classification pipeline, data stream processing can achieve significant improvements in scalability and adaptability. This thesis focuses on documenting these techniques used to address the unique challenges of stationary data streams, paving the way for more efficient and robust machine learning systems.

2.3 Concept Drift Algorithms

Although this thesis focuses on stationary data streams, it is important to briefly acknowledge concept drift algorithms. These methods are designed to handle scenarios where the data distribution changes over time, requiring adaptive strategies. Concept drift refers to the phenomenon where the statistical properties of the target variable, which the model is attempting to predict, change over time in unforeseen ways [11]. This makes it essential for models to not only be robust but also adaptive in order to maintain high predictive performance in dynamic environments.
Concept drift algorithms address this challenge by employing various techniques to monitor, detect, and adapt to these changes.

2.3.1 Sliding Window Techniques

Sliding window techniques are among the simplest yet most effective approaches for handling concept drift. These methods maintain a subset of the most recent data, referred to as a "window," while discarding older observations. The window size can either be fixed or adaptive, depending on the characteristics of the data and the anticipated frequency of drift. A smaller window size allows for faster adaptation to rapid changes but may lead to instability in scenarios with high noise levels. Conversely, a larger window size provides more stability but may delay adaptation to drift [22].
Adaptive windowing methods, such as the Adaptive Sliding Window, dynamically adjust the window size based on statistical properties of the data stream [28]. These methods reduce the need for manual tuning and allow for more flexible handling of concept drift. Sliding window techniques are particularly effective in scenarios where older data becomes irrelevant or misleading, such as financial markets or sensor-based systems with time-dependent patterns.

2.3.2 Ensemble Methods

Ensemble methods are a powerful class of algorithms for managing concept drift by combining the predictions of multiple models [20]. These approaches leverage the diversity of individual models to improve robustness and adaptability. In the context of concept drift, ensemble methods typically involve maintaining a pool of models trained on different segments of the data or at different points in time.
One common strategy is to continuously update the ensemble by adding new models trained on recent data and discarding older models that no longer perform well. This approach ensures that the ensemble remains aligned with the current data distribution. Algorithms such as Dynamic Weighted Majority

(DWM) and Accuracy Updated Ensemble (AUE) use weighting schemes to prioritize models that perform well on recent data, further enhancing adaptability [29].
Another notable ensemble method is Leveraging Bagging [30], which applies bootstrapping and adaptive resampling to improve the ensemble's ability to detect and adapt to concept drift. Ensemble methods are widely used in applications such as fraud detection, where the data distribution can evolve unpredictably over time.

2.3.3 Drift Detection Algorithms

Drift detection algorithms play a crucial role in identifying and responding to changes in data distribution [31]. These methods monitor model performance metrics, such as classification error or prediction confidence, to detect potential drift. When a significant deviation is observed, the model is updated or retrained to ensure alignment with the current data.
One widely used drift detection technique is the Drift Detection Method (DDM) [19], which compares the error rate over time against predefined thresholds. If the error rate exceeds a warning threshold, the algorithm anticipates potential drift and begins preparations for model adaptation. If the error rate surpasses a critical threshold, drift is confirmed, and the model is updated.
The Early Drift Detection Method (EDDM) [32] extends DDM by focusing on the distance between errors rather than their absolute rate, making it more sensitive to gradual drifts. Another advanced technique is Page-Hinkley [33], which uses cumulative error statistics to detect abrupt changes in the data distribution. These algorithms are integral to domains such as network intrusion detection and healthcare monitoring, where rapid adaptation is essential.

2.3.4 IBL-DS

Instance-Based Learning for Data Streams (IBL-DS) is a specific algorithm that extends instance-based learning principles to handle concept drift effectively [34]. Unlike traditional instance-based methods, which rely on a static set of prototypes, IBL-DS dynamically adjusts its instance set to adapt to evolving data distributions.
The core idea of IBL-DS is to maintain a representative subset of instances that reflect the current state of the data stream. This is achieved through a combination of strategies, including instance selection, replacement, and weighting. When a new instance is observed, IBL-DS evaluates its relevance based on its contribution to classification accuracy and its proximity to existing instances. If the new instance provides valuable information, it is added to the instance set, while less relevant or outdated instances are removed.
Additionally, IBL-DS incorporates drift detection mechanisms to identify changes in the data distribution. By monitoring the consistency of classification decisions over time, the algorithm can recognize when the current instance set is no longer representative and initiate updates. This adaptability makes IBL-DS particularly well-suited for real-world applications such as sensor networks and real-time recommendation systems, where the data is non-stationary and changes frequently.
Furthermore, IBL-DS addresses the challenge of computational efficiency by limiting the size of the instance set, ensuring that memory usage remains manageable even for high-velocity data streams.
In summary, IBL-DS exemplifies the potential of instance-based methods to adapt to dynamic environments by combining efficient instance management with robust drift detection. Its ability to

balance accuracy, adaptability, and computational efficiency makes it a compelling choice for concept drift scenarios.

2.4 Summary

This chapter has provided a comprehensive overview of the key concepts and techniques related to data stream processing and instance selection. We explored the challenges associated with data streams, the importance of data reduction for classification, and the various approaches to instance selection, including prototype selection, condensing, prototype generation, and editing. Additionally, we discussed the relevance of concept drift algorithms, highlighting their potential integration with instance selection methods in future research. The next chapter will delve into the methodology, offering a detailed exploration of the theoretical and implementation aspects of instance selection algorithms for data streams.

Chapter 3

Methodology

3.1 Theoretical Overview of Instance Selection Algorithms

Instance selection algorithms are critical in the context of data streams, where continuous data inflow makes it infeasible to retain or process all data. These algorithms aim to identify and store only the most representative instances, enabling efficient and effective classification without significant degradation in accuracy. As data streams often involve high velocity and volume, the ability to selectively retain useful instances is crucial for maintaining computational efficiency and classification performance. This section provides a detailed theoretical overview of the algorithms employed in this thesis.

3.1.1 IB1 (Instance-Based 1 Algorithm)

IB1 is a fundamental instance-based learning algorithm that operates by storing all incoming instances and using them for classification during the testing phase [35]. The algorithm applies the k-Nearest Neighbors (k-NN) principle, where the class label of a new instance is determined based on the majority class of its nearest neighbors in the training set.
While IB1 excels in static datasets due to its simplicity and effectiveness in capturing the underlying data distribution, it suffers from severe limitations in data streams. Its primary drawback is the unbounded growth of the instance set, which leads to excessive memory requirements and computational overhead. Additionally, IB1 lacks any mechanism for handling concept drift, making it unsuitable for non-stationary environments where the data distribution evolves over time. These limitations highlight the need for more adaptive and efficient instance selection methods.

3.1.2 IB2 (Instance-Based 2 Algorithm)

IB2 improves upon IB1 by introducing a condensation mechanism designed to address the issue of unbounded memory usage [35]. Instead of storing all instances, IB2 maintains a reduced set of instances, known as the Condensing Set (CS). This set is incrementally updated by retaining only those instances that are misclassified by the current CS. When a new instance cannot be correctly classified by the existing CS, it is added to the set, ensuring that the CS remains representative of the decision boundaries.

However, IB2 is not without its shortcomings. The algorithm does not differentiate between noisy and informative instances, often retaining misclassified data that can degrade performance [36]. This makes IB2 less robust in noisy environments and highlights the need for further refinement, as seen in subsequent algorithms such as IB3.

3.1.3 IB3 (Instance-Based 3 Algorithm)

IB3 extends IB2 by incorporating an editing mechanism aimed at addressing the impact of noisy data. Each instance in IB3 is assigned a confidence score based on its classification accuracy over time [35]. Instances with low confidence scores, which are likely to be noisy or irrelevant, are removed from the Condensing Set. This editing mechanism enhances the algorithm's robustness to noise while retaining its incremental and non-parametric properties.
IB3 is particularly effective in environments with moderate levels of noise, as it actively removes instances that contribute to misclassification. However, like IB2, IB3 does not explicitly account for concept drift, making it less effective in dynamic environments where the data distribution changes over time. Despite this, IB3 represents a significant improvement over its predecessors in terms of both memory efficiency and noise robustness.

3.1.4 Time-Weighted Forgetting (TWF)

Time-Weighted Forgetting (TWF) is a method specifically designed to handle concept drift in data streams. By employing a sliding window mechanism [37], TWF retains only the most recent observations in the active learning set. Each observation is assigned an exponential weight that decays over time , governed by the formula:

$$w_e \leftarrow \lambda w_e,$$

where $0 < \lambda \leq 1$ is the forgetting rate. Observations with weights below a predefined threshold θ are removed from the active learning set. This approach ensures that the model focuses on recent data while discarding outdated information.
However, TWF is not without its drawbacks. The algorithm can suffer from catastrophic interference, particularly when the sampling distribution changes abruptly [38]. In such cases, valuable information may be prematurely discarded, leading to reduced classification accuracy. Additionally, TWF's asymptotic performance in static environments is often inferior to that of algorithms like IB3, which are optimized for stationary data.

3.1.5 Locally-Weighted Forgetting (LWF)

Locally-Weighted Forgetting (LWF) enhances TWF by incorporating a locality-sensitive decay mechanism [37]. Rather than applying a uniform forgetting rate to all observations, LWF adjusts the forgetting rate based on the distance between a new observation and its nearest neighbors. This allows the algorithm to focus on regions of the input space that are most relevant to the current task.
The weight w_i of an observation i is updated as follows:

$$w_i \leftarrow \phi(X_e, X_i, X_k, \delta) w_i,$$

where ϕ is a function that depends on the squared Euclidean distance d^2 between the observation i and the new observation e, and X_k represents the set of k-nearest neighbors. The decay is truncated to ensure that it remains within a specified range $[0, 1]$. Observations with weights below the threshold θ are removed from the learning set. This locality-aware approach makes LWF more effective than TWF in handling gradual changes in the data distribution.

3.1.6 Prediction Error Context Switching (PECS)

Prediction Error Context Switching (PECS) is a sophisticated algorithm designed to handle changes in the sampling distribution [37]. Unlike TWF and LWF, which permanently discard older observations, PECS deactivates inconsistent instances and stores them in an inactive set for potential future reactivation.

The core idea of PECS is to track the consistency of each instance over a sliding window of recent predictions. Instances are evaluated based on their agreement with true labels, and their consistency statistics are maintained in a shift register. If an instance's consistency falls below a predefined threshold, it is moved to the inactive set. Conversely, instances in the inactive set can be reactivated if their consistency improves over time.

This dynamic approach allows PECS to adapt to recurring concepts, making it particularly effective in cyclic environments. However, the algorithm requires additional storage for maintaining consistency statistics, which may increase its computational overhead.

3.1.7 DRHC (Dynamic Reduction through Homogeneous Clusters)

The DRHC algorithm builds on the Reduction through Homogeneous Clusters (RHC) method by dynamically updating the prototype set as new data becomes available. Using clustering techniques, DRHC creates homogeneous clusters that serve as representative prototypes. By incorporating weights into the clustering process, DRHC ensures that the prototypes represent both old and new data adequately.

This dynamic capability makes DRHC particularly well-suited for streaming data environments, as it effectively balances memory efficiency and adaptability.

3.1.8 AIB2 (Abstraction Instance-Based 2 Algorithm)

AIB2 is an abstraction-based extension of IB2 that improves its performance by generating new prototypes at the center of the data regions they represent. This approach reduces noise sensitivity and enhances the compactness of the Condensing Set (CS). AIB2 retains the incremental nature of IB2, making it highly suitable for data streams.

Experimental results demonstrate that AIB2 achieves higher classification accuracy and data reduction rates compared to IB2, making it a valuable advancement in instance selection for dynamic data environments.

3.2 Implementation Details

This section describes the implementation details of the instance selection algorithms considered in this thesis. The pseudocode for each algorithm is presented, providing a step-by-step outline of their functionality and highlighting their specific mechanisms for instance selection, storage, and update. The algorithms are faithful adaptations of the origian ones that capture the original idea while using Python's advantages. Each pseudocode block is designed to be clear and practical for real-world implementation. A brief explanation and comparison for each one is also provided.

3.2.1 Pseudocode for IB1

The IB1 algorithm is a simple and foundational instance-based learning method that operates by storing all incoming instances and using the k-Nearest Neighbors (k-NN) rule for classification.

Algorithm 1 IB1 Algorithm

Initialize $CD \leftarrow \emptyset$ {Concept Description starts empty}
Initialize $labels \leftarrow \emptyset$ {Labels associated with the concept description}
for each instance x in the training data with corresponding label y **do**
    if CD is not empty **then**
        Append x to CD and y to $labels$
    else
        $CD \leftarrow x$
        $labels \leftarrow y$
    end if
end for
Prediction: For each instance in test data:
Compute nearest neighbor from CD using Euclidean distance
Return the label of the nearest neighbor

Comparison with IB1 from the Literature

In the literature, IB1 is described as a straightforward instance-based learning algorithm that stores all training instances and uses them for classification based on the k-Nearest Neighbors rule. The algorithm operates by adding new instances to the instance set (D) and predicting the class of test instances by computing distances to all stored instances and identifying the majority class among the k-nearest neighbors.

The provided implementation adheres to this general structure but introduces enhancements to improve its usability and computational efficiency. In particular, the use of dynamic updates to the concept description (CD) and the labels array ensures that new instances are seamlessly incorporated as they arrive. Additionally, the implementation leverages efficient distance computations using the `pairwise_distances_argmin_min` function from `scikit-learn`, which significantly reduces the computational overhead compared to a naive implementation.

In summary, while the core logic of IB1 remains consistent with its theoretical description in the literature, the provided implementation enhances its practicality by addressing common challenges

associated with scalability, computational efficiency, and memory management. These improvements make the algorithm well-suited for dynamic and high-velocity data stream environments.

3.2.2 Pseudocode for IB2

The IB2 algorithm builds upon IB1 by introducing a condensation mechanism to reduce the memory and computational burden. Unlike IB1, which stores all incoming instances, IB2 selectively retains only those instances that are misclassified by the current concept description (**CD**). This ensures that the instance set remains compact and representative of the decision boundaries, making it more suitable for resource-constrained environments.

Algorithm 2 IB2 Algorithm

Initialize **CD** $\leftarrow \emptyset$ {Concept Description starts empty}
Initialize **labels** $\leftarrow \emptyset$ {Labels associated with the concept description}
Add the first instance x_0 and its label y_0 to **CD** and **labels**
for each instance x_i in the training data with corresponding label y_i (starting from the second instance) **do**
    Compute the nearest neighbor of x_i in **CD**
    Retrieve the label of the nearest neighbor, y_{nn}
    if $y_{nn} \neq y_i$ **then**
        Append x_i to **CD** and y_i to **labels** {Store misclassified instance}
    end if
end for
Prediction: For each instance in test data:
Compute the nearest neighbor from **CD** using Euclidean distance
Return the label of the nearest neighbor

Comparison with IB2 from the Literature

In the literature, IB2 is described as an instance-based algorithm that retains only those instances that are essential for maintaining the classification accuracy of the model. The algorithm initializes with an empty instance set (**CD**) and adds the first instance by default. For every subsequent instance, it checks whether the instance can be correctly classified using the current **CD**. If the instance is misclassified, it is added to **CD**; otherwise, it is discarded. This mechanism ensures that only the most informative instances, particularly those near decision boundaries, are retained.
The implementation follows this theoretical structure closely while incorporating enhancements that improve its practicality and computational efficiency.
Firstly, the implementation supports incremental updates to the **CD** and **labels** arrays. This dynamic approach ensures that the algorithm can handle streaming data, where instances arrive sequentially, rather than requiring access to the entire dataset at once.
Secondly, the implementation leverages the`pairwise_distances_argmin_min` function from the `scikit-learn` library to compute the nearest neighbor efficiently, just like IB1. This is especially beneficial as the size of **CD** grows, ensuring that the algorithm remains computationally feasible even as new data is added.

Lastly, the implementation adheres to the principle of compact representation, a core feature of IB2. By storing only misclassified instances, it significantly reduces the memory footprint compared to IB1, which stores all instances. This property makes IB2 particularly suitable for resource-constrained environments, such as embedded systems or streaming applications.

3.2.3 Pseudocode for IB3

The IB3 algorithm extends IB2 by introducing a mechanism to address noise in the data. While IB2 retains all misclassified instances, IB3 refines this approach by assigning confidence scores to each instance in the concept description (**CD**). Instances with low confidence scores, which are likely to be noisy or irrelevant, are removed from **CD**. This editing mechanism improves the robustness of the algorithm in noisy environments while maintaining the compactness and incremental nature of IB2. Below is the pseudocode for IB3 as implemented.

Algorithm 3 IB3 Algorithm

Initialize **CD** $\leftarrow \emptyset$ {Concept Description starts empty}
Initialize **labels** $\leftarrow \emptyset$ {Labels associated with the concept description}
Initialize **performance_records** $\leftarrow \emptyset$ {Tracks successes and failures for each instance in **CD**}
Set **accept_conf** $\leftarrow 0.90$ and **discard_conf** $\leftarrow 0.75$ {Confidence thresholds for acceptance and removal}
for each instance x_i in the training data with corresponding label y_i **do**
    if CD is not empty **then**
        Compute the nearest neighbor of x_i in **CD**
        Retrieve the label of the nearest neighbor, y_{nn}
        if $y_{nn} = y_i$ **then**
            Increment the success count for the nearest neighbor
        else
            Increment the failure count for the nearest neighbor
        end if
        Compute the confidence interval for the nearest neighbor based on its successes and failures
        Compute the class probability for y_{nn}
        if the upper bound of the confidence interval $<$ class probability **then**
            Remove the nearest neighbor from **CD**, **labels**, and **performance_records**
        end if
    end if
    if CD is empty **or** $y_{nn} \neq y_i$ **then**
        Append x_i to **CD** and y_i to **labels**
        Initialize performance record for x_i with one success and zero failures
    end if
end for
Prediction: For each instance in test data:
Compute the nearest neighbor from **CD** using Euclidean distance
Return the label of the nearest neighbor

Comparison with IB3 from the Literature

In the original implementation, IB3 is described as an instance-based learning algorithm that incorporates noise-handling capabilities through the use of confidence intervals. The algorithm evaluates the performance of each instance in the concept description (**CD**) by tracking the number of correct and incorrect classifications it contributes to. Instances with low confidence scores, which are likely to be noisy or uninformative, are removed from **CD** to improve classification accuracy and reduce the size of the stored data.
Our implementation of IB3 adheres closely to this theoretical description while introducing enhancements to improve computational efficiency and adaptability.
First, it dynamically updates the **CD**, **labels**, and **performance_records** arrays as new data arrives. Each instance's performance is evaluated in real-time, ensuring that the concept description remains relevant and compact as the data evolves.
Second, it uses confidence intervals to evaluate the reliability of each instance in **CD**. The confidence interval is computed based on the number of correct and incorrect classifications contributed by the instance, and instances with low confidence are removed.
Third, it computes class probabilities for the nearest neighbor's label and compares these probabilities to the upper bound of the confidence interval. If the confidence interval indicates that the instance is unreliable, it is removed from **CD**. This approach prevents the accumulation of noisy data while retaining instances that are critical for defining decision boundaries.
Compared to IB2, IB3 provides a more robust mechanism for handling noise and maintaining a compact representation of the data. By removing noisy instances, IB3 reduces the risk of misclassification caused by outliers or mislabeled data. However, this additional editing mechanism increases the computational complexity of the algorithm, as it requires the maintenance of performance records and the computation of confidence intervals for each instance.

3.2.4 Pseudocode for TWF (Time-Weighted Forgetting Algorithm)

The Time-Weighted Forgetting (TWF) algorithm addresses the challenge of adapting to non-stationary data streams by introducing an exponential decay mechanism for managing the weights of stored instances. This decay mechanism ensures that older instances gradually lose their influence, allowing the algorithm to focus on more recent and relevant data. Instances with weights falling below a predefined threshold are discarded, thereby maintaining a compact and dynamically updated instance set.

Algorithm 4 TWF Algorithm

Initialize **learning_set** $\leftarrow \emptyset$ {Stores instances, labels, and weights}
Set **forgetting_rate** $\lambda \leftarrow 0.9$ {Rate at which weights decay}
Set **threshold** $\theta \leftarrow 0.02$ {Minimum weight for instance retention}
for each instance (x, y) in the incoming data stream **do**
    Add (x, y) to **learning_set** with an initial weight $w \leftarrow 1.0$
    for each instance (x_i, y_i, w_i) in **learning_set do**
        Update the weight: $w_i \leftarrow w_i \times \lambda$
        if $w_i < \theta$ **then**
            Remove (x_i, y_i, w_i) from **learning_set**
        end if
    end for
end for

Comparison with TWF from the Literature

The TWF algorithm, as described in the literature, is designed to manage the dynamic nature of data streams by continuously updating the relevance of stored instances. This is achieved through an exponential forgetting mechanism, where the weight of each instance decays over time according to a forgetting rate (λ). Instances with weights below a threshold (θ) are removed from the learning set, ensuring that the algorithm focuses on recent data while discarding outdated information.
Our implementation aligns closely with the description of TWF. It initializes an empty learning set and iteratively updates it as new data arrives. For each new instance, the algorithm assigns an initial weight of 1.0 and updates the weights of all existing instances in the learning set by multiplying them with the forgetting rate. Instances with weights below the threshold are discarded, ensuring that the learning set remains compact and relevant.
Unlike static algorithms such as IB1 and IB2, TWF dynamically adapts to changes in the data distribution, making it particularly suitable for non-stationary environments. By focusing on recent data, it handles gradual concept drift effectively. However, TWF has limitations in scenarios with abrupt changes in the data distribution, as it may prematurely discard instances that are still relevant to the new concept.

3.2.5 Pseudocode for LWF (Locally-Weighted Forgetting Algorithm)

The Locally-Weighted Forgetting (LWF) algorithm enhances TWF by introducing a locality-sensitive decay mechanism. Instead of applying a uniform decay to all stored instances, LWF adjusts the forgetting rate for each instance based on its proximity to a new data point. This ensures that the algorithm focuses on retaining instances that are most relevant to the current region of the feature space. The pseudocode for LWF:

Comparison with LWF from the Literature

In the original study, LWF is shown to be an improvement to TWF that incorporates a locality-sensitive decay mechanism. This mechanism allows LWF to prioritize the retention of instances that are most relevant to the current data point, as determined by their proximity in the feature space. The algorithm

Algorithm 5 LWF Algorithm

Initialize **learning_set** $\leftarrow \emptyset$ {Stores instances, labels, and weights}
Set **k** $\leftarrow 1$ {Number of nearest neighbors}
Set **decay_rate** $\lambda \leftarrow 0.8$ {Minimum forgetting rate}
Set **deletion_threshold** $\theta \leftarrow 0.01$ {Minimum weight for instance retention}
for each instance (x, y) in the incoming data stream **do**
    Compute distances between x and all instances in **learning_set**
    if learning_set is not empty **then**
        Identify **k** nearest neighbors of x based on distances
        Compute **max_distance** as the distance to the farthest neighbor
        Calculate **decay_factors** for the nearest neighbors using:

$$decay\_factor = \lambda + (1 - \lambda) \cdot \left(1 - \left(\frac{distance}{max\_distance}\right)^2\right)$$

        Update weights of the **k** nearest neighbors by multiplying with their respective **decay_factors**
        Remove instances from **learning_set** where **weight** $< \theta$
    end if
    Add $(x, y, weight = 1.0)$ to **learning_set** {New instance with initial weight of 1.0}
end for

employs a truncated quadratic decay function to compute decay factors, ensuring that instances closer to the current data point experience less forgetting than those farther away.
This implementation dynamically updates the learning set by applying a decay function to the weights of the nearest neighbors of each incoming data point. Instances with weights below the deletion threshold are removed, ensuring that the learning set remains compact and focused on relevant data. However, LWF shares some limitations with TWF. Specifically, it may struggle in environments with abrupt changes in the data distribution, as it does not incorporate explicit mechanisms for detecting or responding to such changes. Additionally, the quadratic decay function relies on accurate distance computations, which can become computationally expensive in high-dimensional spaces.

3.2.6 Pseudocode for PECS (Prediction Error Context Switching Algorithm)

The Prediction Error Context Switching (PECS) algorithm is designed to handle dynamic environments by leveraging both active and inactive instance sets. Unlike TWF and LWF, PECS does not permanently discard instances but instead deactivates and stores them in an inactive set for potential reactivation. The algorithm uses a sliding window of recent agreement statistics to evaluate the reliability of instances, ensuring adaptability to concept drift and non-stationary data distributions. The pseudocode for PECS, as implemented, is as follows:

Algorithm 6 PECS Algorithm

Initialize **active_set** $\leftarrow \emptyset$ {Stores active instances, their labels, and agreement histories}
Initialize **inactive_set** $\leftarrow \emptyset$ {Stores deactivated instances}
Set **k** $\leftarrow 1$ {Number of nearest neighbors}
Set **p_min** $\leftarrow 0.6$ {Minimum agreement probability for reactivation}
Set **p_max** $\leftarrow 0.4$ {Maximum disagreement probability for deactivation}
Set **shift_register_length** $\leftarrow 20$ {Length of agreement history window}
for each instance (x, y) in the incoming data stream **do**
    Add $(x, y,$ **history**$)$ to **active_set**, where **history** is initialized as $[1] \times$ **shift_register_length**
    Compute distances between x and all instances in **active_set**
    Identify **k** nearest neighbors of x based on distances
    for each neighbor in **k** nearest neighbors **do**
        Update agreement history:
        Remove oldest record from **history**
        Append 1 if neighbor's label matches y, otherwise append 0
        Compute agreement ratio: **agreement_ratio** $\leftarrow \frac{\text{sum(history)}}{\textbf{shift\_register\_length}}$
        if agreement_ratio $<$ **p_max then**
            Move instance to **inactive_set**
        else if agreement_ratio $>$ **p_min** and instance is in **inactive_set then**
            Reactivate instance by moving it back to **active_set**
        end if
    end for
end for

Comparison with PECS from the Literature

As outlined in the foundational research, PECS is a dynamic instance selection algorithm that maintains two distinct sets of instances: the active set and the inactive set. Instances in the active set are used for classification, while instances in the inactive set are stored for potential reactivation. This dual-set structure allows PECS to adapt to cyclic or recurring concepts in dynamic data streams.
The provided implementation adheres to this theoretical framework by maintaining agreement statistics for each instance in a sliding window. These statistics are used to compute an agreement ratio, which determines whether an instance should remain active, be deactivated, or be reactivated. The algorithm uses user-defined thresholds (**p_min** and **p_max**) to make these decisions, ensuring that only reliable instances are retained in the active set.
However, the increased complexity of PECS comes at the cost of some computational overhead. The algorithm requires the maintenance of agreement histories for each instance and the computation of agreement ratios, which can become expensive in high-velocity data streams. Additionally, the choice of parameters (**k**, **p_min**, **p_max**, and **shift_register_length**) can significantly impact the algorithm's performance, necessitating careful tuning for different datasets.

3.2.7 Pseudocode for DRHC (Dynamic Reduction through Homogeneous Clusters Algorithm)

Dynamic Reduction through Homogeneous Clusters (DRHC) is an advanced instance selection algorithm designed to manage large and dynamic datasets by maintaining a compact and representative condensing set (**CS**). DRHC builds upon the principles of Reduction through Homogeneous Clusters (RHC). It's approach ensures that the algorithm adapts to changes in the data distribution while maintaining computational efficiency. The pseudocode for it, as implemented, is detailed below:

Algorithm 7 DRHC Algorithm

Initialize **CS** $\leftarrow \emptyset$ {Start with an empty condensing set}
for each instance (x, y) in the incoming data stream **do**
  if CS is empty **then**
    Add $(x, y, 1)$ to **CS** {First instance becomes a prototype with weight 1}
  else
    Compute the distance between x and all centroids in **CS**
    Identify the nearest centroid (c, l, w) in **CS**
    if $l = y$ **and** distance$(x, c) < t_h$ **then**
      Update the centroid c as $c \leftarrow \frac{c \cdot w + x}{w+1}$ {Weighted update}
      Increment weight $w \leftarrow w + 1$
    else
      Add $(x, y, 1)$ to **CS** {Add a new cluster for this instance}
    end if
  end if
  Periodically perform recursive clustering on **CS** to enforce homogeneity
end for
Return CS

Comparison with DRHC from the Literature

According to the initial publication, DRHC is presented as a dynamic extension of the RHC (Reduction through Homogeneous Clusters) algorithm. The primary goal of DRHC is to manage streaming data by maintaining a compact, dynamic, and representative condensing set (**CS**). This is achieved through recursive k-means clustering, which partitions data into homogeneous clusters, and a weighted update mechanism that adjusts the prototypes to reflect incoming data.
The provided implementation closely follows the theoretical description while introducing several practical enhancements that improve its adaptability and efficiency in real-world scenarios.
First, the implementation incorporates a dynamic update mechanism for the condensing set. As new instances arrive, the algorithm either adjusts the nearest cluster centroid or creates a new cluster if the instance does not fit into any existing cluster based on the homogeneity threshold (t_h). This ensures that the condensing set remains compact and accurately represents the data distribution.
Second, the implementation uses a weighted mean update to adjust the centroids. When a new instance is added to an existing cluster, the centroid is updated as the weighted average of the previous centroid and the new instance. This approach preserves the influence of historical data while incorporating new information, making the algorithm robust to gradual changes in the data distribution.
Third, the implementation includes a periodic recursive clustering step to enforce homogeneity among the clusters. This step ensures that the clusters remain well-separated and representative, even as the data distribution evolves over time. The recursive clustering step also prevents the condensing set from growing excessively, maintaining computational efficiency.

3.2.8 Pseudocode for AIB2 (Abstraction Instance-Based 2 Algorithm)

The Abstraction Instance-Based 2 (AIB2) algorithm is an advanced version of IB2, designed to address the limitations of static prototype selection. Unlike IB2, which simply adds misclassified instances to the condensing set (**CS**), AIB2 generates new prototypes that are placed at the center of the data regions they represent. This abstraction mechanism not only reduces the size of **CS** but also improves classification accuracy by ensuring that prototypes remain representative of their local data distributions. Each prototype is updated incrementally using a weighted mean approach, making the algorithm highly efficient and adaptive for dynamic and streaming environments.
The pseudocode for AIB2, is:

Algorithm 8 AIB2 Algorithm

Initialize **CS** $\leftarrow \emptyset$ {The condensing set starts empty}
for each instance (x, y) in the incoming data stream **do**
  if CS is empty **then**
    Add $(x, y,$ **weight** $= 1)$ to **CS** {The first instance initializes the condensing set}
  else
    Compute the Euclidean distance between x and all prototypes in **CS**
    Identify the nearest prototype (p, l, w) in **CS**, where p is the prototype, l is the label, and w is the weight
    if $l \neq y$ **then**
      Add $(x, y,$ **weight** $= 1)$ to **CS** {Misclassified instance becomes a new prototype}
    else
      Update the nearest prototype p using the weighted mean formula:

$$p \leftarrow \frac{w \cdot p + x}{w + 1}$$

      {This moves the prototype closer to the new instance}
      Increment the weight $w \leftarrow w + 1$ {Update the weight to reflect the new addition}
    end if
  end if
end for
Return CS

Comparison with AIB2 from the Literature

The AIB2 algorithm is aimed at improving classification accuracy and reducing the size of the condensing set (**CS**). By generating prototypes at the center of the data regions they represent, AIB2 overcomes the limitations of IB2, where prototypes remain static and unoptimized. The algorithm leverages weighted mean updates to incrementally adjust prototypes as new data arrives, ensuring that they remain representative of their respective data regions.
The implementation processes instances incrementally, identifying the nearest prototype for each incoming instance. If the nearest prototype misclassifies the instance, the algorithm adds the instance as a new prototype to **CS**. Otherwise, the nearest prototype is updated using a weighted mean formula, which adjusts the prototype to incorporate the new instance while retaining the influence of previously seen data. This dynamic updating mechanism is a key feature of AIB2 and distinguishes it from IB2.
AIB2 introduces several advantages over IB2. By updating prototypes to the center of their data regions, AIB2 achieves better representation of the data, leading to improved classification accuracy. Furthermore, the abstraction mechanism reduces the size of **CS**, resulting in higher reduction rates and lower memory usage. This makes AIB2 particularly effective for large-scale datasets and streaming applications, where efficiency and scalability are critical, as will be apparent below.

Chapter 4

Experimental Setup

4.1 Test-then-Train Methodology

The *Test-then-Train* methodology is a widely adopted approach in the context of data streams, where the data arrives incrementally in the form of batches or instances [12]. Unlike traditional machine learning paradigms that rely on the availability of the complete dataset for training, the Test-then-Train approach is specifically designed to simulate real-time learning. This section describes the methodology, its underlying principles, and its relevance to evaluating instance selection algorithms in stationary data streams.

4.1.1 Overview

The Test-then-Train methodology operates in a sequential manner, where each incoming batch of data is first used to evaluate the current model and then incorporated into the training process. This two-step approach ensures that the model is continuously tested on unseen data, mimicking real-world scenarios where data arrives incrementally and the model must adapt over time. The methodology is particularly well-suited for streaming environments, where computational efficiency and memory constraints are critical considerations.
Formally, let $\mathcal{D} = \{X_1, X_2, \ldots, X_t\}$ represent the incoming data stream, where X_t denotes the batch of data arriving at time step tThe Test-then-Train methodology can be summarized as follows:

1. **Testing Phase**: Evaluate the model M_t using the instances in batch X_t and compute performance metrics such as classification accuracy, memory usage, and runtime.

2. **Training Phase**: Update the model M_t using the instances in X_t to obtain the updated model M_{t+1} for future predictions.

4.1.2 Advantages of Test-then-Train

The Test-then-Train methodology offers several advantages in the context of data streams [20]:

- **Incremental Learning**: By training the model after each testing phase, the methodology ensures that the model adapts incrementally to the evolving data stream.

- **Realistic Evaluation**: Since the model is tested on unseen data before being trained, the methodology provides an unbiased estimate of its performance under real-world conditions.
- **Memory Efficiency**: By processing data in small batches, the methodology aligns with the constraints of streaming environments, where storing the entire dataset is infeasible [39].

4.1.3 Implementation Details

In this thesis, the Test-then-Train methodology is implemented to evaluate the instance selection algorithms described in Section 3.1. The implementation assumes a stationary data stream, where the underlying data distribution remains constant over time. To simulate the streaming environment, the datasets are divided into non-overlapping batches of equal size, which are sequentially fed to the model.
For each batch X_t, the following steps are performed:

1. Compute classification accuracy on X_t using the current model M_t.
2. Measure memory usage and runtime for processing the batch.
3. Update the model M_t to M_{t+1} by incorporating the instances in X_t.

The performance metrics are averaged across all batches to obtain a comprehensive evaluation of the algorithms. The use of synthetic and real-world datasets, as described in Section 4.2, ensures the generalizability of the findings.

4.2 Description of Datasets

The datasets used in this thesis are selected to evaluate the performance of instance selection algorithms across a diverse range of classification scenarios. These datasets, sourced from the KEEL dataset repository, encompass a variety of properties in terms of dimensionality, class imbalance, and complexity. All datasets used were randomly shuffled to avoid overfitting on one element of the dataset.

4.2.1 Dataset Characteristics

The datasets span different domains and vary in their feature space, class distribution, and complexity. Table 4.1 summarizes the key characteristics of the datasets, including the number of instances, attributes and classes [40].

4.2.2 Dataset Descriptions

- **balance**: This dataset is based on a simple balance scale problem, where the class distribution is evenly split across three classes. It is often used to test classification algorithms under balanced conditions.
- **banana**: A two-dimensional dataset characterized by overlapping class distributions, making it challenging for instance selection and classification tasks.

Dataset	Instances	Attributes	Classes
balance	625	4	3
banana	5300	2	2
ecoli	336	7	8
kddcup	494020	41	23
letter	20000	16	26
magic	19020	10	2
monk-2	432	6	2
penbased	10992	16	10
phoneme	5404	5	2
satimage	6435	36	6
shuttle	58000	9	7
texture	5500	40	11
twonorm	7400	20	2
yeast	1484	8	10

Table 4.1: Summary of Datasets Used in the Experiments, retrieved from the KEEL repository[40]

- **ecoli**: This dataset represents a biological classification problem related to protein localization sites in cells. It is moderately imbalanced, with eight classes.
- **kddcup**: A high-dimensional dataset sourced from the KDD Cup competition, used for network intrusion detection. It contains a large number of instances and classes, making it ideal for evaluating scalability.
- **letter**: This dataset involves the classification of 26 capital letters from the English alphabet based on extracted features. It is high-dimensional and multi-class, providing a comprehensive benchmark for classification algorithms.
- **magic**: This dataset is derived from gamma telescope data and is used for binary classification between signal and background events.
- **monk-2**: A synthetic dataset generated from logical conditions, frequently used for benchmarking algorithms under controlled settings.
- **penbased**: This dataset focuses on handwritten digit classification, containing features extracted from pen trajectory data. It has 10 classes and is widely used in pattern recognition research.
- **phoneme**: A binary classification dataset derived from speech recognition tasks, focusing on distinguishing between different phonemes.
- **satimage**: A multi-class dataset derived from satellite images, involving 36 features and six classes. It is widely used in remote sensing applications.
- **shuttle**: A large dataset from NASA's shuttle program, characterized by seven classes and a significant class imbalance.

- **texture**: This dataset is derived from texture image analysis and consists of 11 classes. Its high dimensionality poses a challenge for memory-efficient instance selection.
- **twonorm**: A synthetic dataset with two normally distributed classes, frequently used to evaluate classification performance under Gaussian noise.
- **yeast**: This dataset involves protein localization in yeast cells, with eight attributes and ten classes. It is moderately imbalanced and represents a challenging multi-class problem.

4.3 Evaluation Metrics

To assess the performance of the instance selection algorithms proposed in this thesis, three evaluation metrics are employed. These metrics measure classification accuracy, memory efficiency, and computational cost, providing a comprehensive framework for performance comparison. The metrics are as follows:

4.3.1 Average Accuracy

Average accuracy quantifies the predictive performance of the instance selection algorithms across all batches of a dataset. It is calculated as the proportion of correctly classified instances in a batch, averaged over all batches in the data stream. Formally, the average accuracy $\mathcal{A}$ for a dataset can be expressed as:

$$\mathcal{A} = \frac{1}{T} \sum_{t=1}^{T} \frac{\text{Number of Correct Predictions in Batch } t}{\text{Total Instances in Batch } t},$$

where T represents the total number of batches. This metric, bounded between 0 and 1, provides a normalized measure of classification accuracy and ensures comparability across datasets with different batch sizes.

4.3.2 Reduction Rate

Reduction rate evaluates the memory efficiency of the instance selection algorithms by measuring the proportion of instances discarded during the data stream. A higher reduction rate indicates that the algorithm retains a smaller subset of the data, leading to greater memory savings. The reduction rate $\mathcal{R}$ is defined as:

$$\mathcal{R} = 1 - \frac{|\mathcal{D}_{\text{retained}}|}{|\mathcal{D}_{\text{total}}|},$$

where $|\mathcal{D}_{\text{retained}}|$ is the number of instances retained by the algorithm and $|\mathcal{D}_{\text{total}}|$ is the total number of instances in the data stream. This metric, also bounded between 0 and 1, provides a quantitative measure of the algorithm's data reduction capability.

4.3.3 CPU Time

CPU time measures the computational cost of the instance selection algorithms, expressed in milliseconds (ms). It is computed as the average time required to process all batches of a dataset, including both testing and training phases. This metric reflects the algorithm's efficiency in handling the computational demands of real-time data stream processing.

4.3.4 Relevance to Streaming Environments

The combination of these three metrics ensures a holistic evaluation of the instance selection algorithms.

- **Average Accuracy**: Demonstrates the ability of the algorithm to maintain high predictive performance despite memory constraints.
- **Reduction Rate**: Highlights the effectiveness of the algorithm in minimizing memory usage while retaining representative data.
- **CPU Time**: Provides insights into the computational feasibility of deploying the algorithm in real-time applications.

By simultaneously considering accuracy, memory efficiency, and computational cost, these metrics align with the practical requirements of streaming environments, where both resource constraints and predictive performance are critical.

4.4 Experimental Framework

This section describes the experimental framework used to evaluate the instance selection algorithms proposed in this thesis. The framework integrates the *Test-then-Train* methodology, the selected datasets, and the defined evaluation metrics to provide a rigorous and comprehensive evaluation of algorithm performance.

4.4.1 Experimental Design

The experimental design simulates a streaming environment by dividing each dataset into non-overlapping batches. The instance selection algorithms are applied to process these batches incrementally, following the *Test-then-Train* methodology outlined in Section 4.1. For each batch, the following steps are performed:

1. The current model is evaluated on the incoming batch to compute classification accuracy.
2. The model is updated using the instances in the batch, subject to the instance selection process.
3. Memory usage and CPU time are recorded for each batch.

4.4.2 Implementation Details

The experiments are implemented in Python, leveraging libraries such as NumPy, SciPy, and scikit-learn for numerical computations and machine learning functionalities. All experiments are conducted on a machine with the following specifications:

- **Processor**: AMD Ryzen 5 3600, 6 Cores/12 Threads @ 4.20 GHz
- **Memory**: 48 GB RAM
- **Operating System**: Microsoft Windows 11 Pro - Version 10.0.22635 Build 22635
- **Python Version**: 3.11.7

4.4.3 Statistical Analysis

To validate the experimental results, statistical tests are employed to compare the performance of the instance selection algorithms. The following tests are applied:

- **Wilcoxon Signed-Rank Test**: A non-parametric test used to compare paired results between two algorithms across all datasets. This test evaluates whether one algorithm consistently outperforms another [41].
- **Friedman Test**: A non-parametric test used to compare the performance of multiple algorithms across multiple datasets. If significant differences are detected, a post-hoc test (e.g., Nemenyi test) is performed to identify specific pairwise differences [42].

4.4.4 Reproducibility

To ensure the reproducibility of the experiments, all datasets, code, and scripts used in this thesis are documented and organized. Key details, including pseudocode, are provided in the appendices for reference.

4.4.5 Expected Outcomes

The experimental framework is designed to answer the following research questions:

- How well do the instance selection algorithms balance classification accuracy and memory efficiency in a streaming environment?
- What are the trade-offs between accuracy, reduction rate, and computational cost for different datasets?
- Are there significant differences in performance among the evaluated algorithms, as evidenced by statistical analysis?

The findings will provide insights into the strengths and weaknesses of the proposed algorithms and their applicability to real-world data stream scenarios.

Chapter 5

Results and Discussion

This chapter presents the results of the experiments conducted to evaluate the performance of the instance selection algorithms described in Chapter 3. The discussion highlights key findings, provides insights into the trade-offs among accuracy, memory efficiency, and computational cost, and evaluates the statistical significance of the observed differences.

5.1 Experimental Results

This section presents the detailed results of the experiments conducted using the selected datasets and instance selection algorithms. The evaluation metrics—reduction rate, CPU time, and average accuracy—are reported for each dataset and algorithm. The results provide insights into the performance trade-offs and the suitability of each algorithm for streaming environments.

5.1.1 Performance Across Datasets

Table 5.1 summarizes the performance of the instance selection algorithms across all datasets. The results are reported for three key metrics: reduction rate, CPU time (ms), and average accuracy (0-1).

Dataset	IB1			IB2			IB3			TWF			LWF			DRHC			AIB2			PECS		
	RR	CPU	Acc	RR	CPU	Acc	RR	CPU	Acc	RR	CPU	Acc	RR	CPU	Acc	RR	CPU	Acc	RR	CPU	Acc	RR	CPU	Acc
balance.dat	0.00	24.10	0.80	0.70	203.34	0.75	0.70	200.63	0.75	0.94	22.89	0.75	0.31	18.42	0.78	0.78	24.88	0.75	0.64	5.00	0.96	0.84	44.19	0.78
banana.dat	0.00	5.32	0.87	0.85	1510.05	0.84	0.85	1601.05	0.84	0.99	139.27	0.79	0.43	576.84	0.87	0.84	195.44	0.84	0.75	44.11	0.93	0.90	1167.40	0.87
ecoli.dat	0.00	3.53	0.13	0.98	351.33	0.14	0.98	310.17	0.14	0.89	36.46	0.15	0.63	23.27	0.13	0.98	66.76	0.09	0.64	6.78	0.83	0.40	112.20	0.13
kddcup.dat	0.00	300.39	0.99	1.00	2981.27	0.98	1.00	3293.78	0.98	1.00	247.61	0.95	0.04	191427.10	0.99	1.00	5.69	0.97	1.00	299.65	1.00	1.00	4104.92	0.96
letter.dat	0.00	24.78	0.93	0.88	2399.83	0.89	0.88	3877.93	0.89	1.00	496.12	0.25	0.42	9328.99	0.91	0.69	841.11	0.92	0.88	387.93	0.99	0.90	16523.92	0.85
magic.dat	0.00	20.58	0.77	1.00	4329.38	0.94	1.00	4026.42	0.94	1.00	467.41	0.94	0.44	7542.55	0.80	0.97	120.31	0.83	0.88	185.04	0.85	0.90	14906.44	0.91
monk-2.dat	0.00	3.51	0.94	0.96	243.73	0.93	0.96	229.60	0.93	0.91	27.21	0.72	0.53	14.46	0.90	0.95	14.32	0.98	0.94	10.69	0.99	0.73	51.94	0.88
penbased.dat	0.00	97.28	0.99	0.97	2442.67	0.97	0.97	2358.92	0.97	1.00	249.78	0.78	0.44	2311.37	0.99	0.94	91.37	0.97	0.97	58.48	0.99	0.91	4107.70	0.98
phoneme.dat	0.00	4.32	0.87	0.83	1445.17	0.82	0.83	1458.67	0.82	0.99	122.07	0.74	0.41	521.79	0.86	0.82	182.11	0.81	0.77	49.74	0.95	0.91	1007.04	0.81
satimage.dat	0.00	6.09	0.84	0.87	1518.84	0.79	0.87	1515.39	0.79	0.99	133.54	0.63	0.39	873.32	0.84	0.92	69.77	0.82	0.85	68.77	0.98	0.91	1259.08	0.77
shuttle.dat	0.00	52.05	1.00	1.00	4299.28	1.00	1.00	4158.24	1.00	1.00	449.72	0.91	0.38	22347.44	1.00	0.99	46.18	0.99	0.99	102.44	1.00	0.97	13969.29	0.99
texture.dat	0.00	4.74	0.49	0.99	1008.97	0.53	0.99	934.69	0.53	0.99	109.15	0.54	0.40	590.96	0.49	0.99	5.29	0.39	0.96	23.77	0.96	0.92	764.20	0.53
twonorm.dat	0.00	5.68	0.94	0.88	1855.46	0.88	0.89	1562.50	0.88	1.00	142.03	0.91	0.43	852.61	0.94	1.00	4.03	0.98	0.91	48.77	0.98	0.92	1381.48	0.93
yeast.dat	0.00	6.84	0.46	0.57	2455.26	0.41	0.46	1082.47	0.41	0.98	87.76	0.43	0.50	136.40	0.43	0.63	239.77	0.43	0.37	36.52	0.96	0.73	592.56	0.45

Table 5.1: Detailed Performance of Instance Selection Algorithms Across Datasets

5.1.2 Key Observations

The experimental results reveal distinct trade-offs between the three evaluation metrics: accuracy, reduction rate, and CPU time. These trade-offs highlight the importance of selecting an algorithm

based on the specific constraints and objectives of the application.

- **Accuracy**: Algorithms such as IB1 and AIB2 consistently achieve high classification accuracy across datasets, demonstrating their ability to retain the essential characteristics of the data. However, IB1 sacrifices memory efficiency entirely (reduction rate = 0.00), whereas AIB2 achieves competitive reduction rates (up to 0.88) with minimal accuracy loss. In contrast, algorithms like LWF and PECS, which prioritize memory reduction, show a moderate decline in accuracy on certain datasets, such as `banana.dat` and `phoneme.dat`.
- **Reduction Rate**: PECS emerges as the most memory-efficient algorithm, achieving reduction rates above 0.90 on several datasets, such as `penbased.dat` and `texture.dat`. However, this aggressive instance selection strategy occasionally leads to increased computational costs and slightly lower accuracies. TWF and LWF also demonstrate high reduction rates while maintaining competitive accuracy, making them suitable for memory-constrained applications.
- **CPU Time**: The computational efficiency of the algorithms varies significantly. TWF and AIB2 consistently demonstrate low CPU times, even on large datasets such as `magic.dat` and `kddcup.dat`. In contrast, PECS and LWF exhibit significantly higher CPU times on complex datasets, such as `shuttle.dat` and `kddcup.dat`, highlighting the need to balance memory efficiency and computational feasibility in real-time applications.

5.2 Statistical Analysis

The performance of the instance selection algorithms was subjected to rigorous statistical analysis to validate the observed differences in accuracy, reduction rate, and computational cost across the evaluated datasets. Two statistical tests were employed: the Friedman Test, which ranks multiple algorithms based on their performance across datasets, and the Wilcoxon Signed-Rank Test, which compares algorithm pairs to determine statistically significant differences. This section provides a detailed discussion of the results, supported by visualizations and interpretations.

5.2.1 Friedman Test Results

The Friedman Test was conducted to compare the algorithms' ranks across the three evaluation metrics: accuracy, reduction rate, and CPU time. This test is a non-parametric alternative to the repeated-measures ANOVA and evaluates whether there are statistically significant differences among the algorithms. The Friedman Test was conducted using PSPP, an open-source statistical software tool. Table 5.2 summarizes the mean ranks of each algorithm for the three metrics.

The results of the Friedman Test indicate statistically significant differences among the algorithms across all three metrics, with p-values of 0.000 for accuracy, reduction rate, and CPU time. This confirms that the observed differences in algorithm performance are unlikely to be due to random chance.

For accuracy, AIB2 achieved the highest mean rank of 7.46, indicating its superior classification performance across datasets. In contrast, TWF received the lowest mean rank of 2.75, reflecting its reduced focus on accuracy in favor of memory efficiency.

Algorithm	Accuracy (accs)	Reduction Rate (rrs)	CPU Time (cpu)
IB1	5.57	1.00	1.71
IB2	3.96	5.68	7.07
IB3	3.96	5.61	6.64
PECS	3.75	5.04	6.57
TWF	2.75	7.04	3.57
LWF	4.89	2.21	5.21
DRHC	3.64	5.43	3.00
AIB2	7.46	4.00	2.21

Table 5.2: Friedman Test Mean Ranks for Accuracy, Reduction Rate, and CPU Time

When comparing reduction rates, TWF achieved the highest rank of 7.04, demonstrating its strong memory efficiency. IB1, which does not perform any reduction, received the lowest rank of 1.00, as expected. AIB2 achieved a moderate rank of 4.00, balancing memory efficiency with its focus on accuracy.
In terms of computational cost, AIB2 and TWF emerged as the most efficient algorithms, with mean ranks of 2.21 and 3.57, respectively. This highlights their suitability for real-time applications. Conversely, IB2 and IB3 exhibited the highest computational costs, with mean ranks of 7.07 and 6.64, respectively, making them less favorable for time-critical scenarios.

5.2.2 Wilcoxon Signed-Rank Test Results

To further analyze pairwise differences between algorithms, the Wilcoxon Signed-Rank Test was conducted for accuracy, reduction rate, and CPU time. This non-parametric test provides a detailed comparison of whether one algorithm significantly outperforms another across datasets. Table 5.3 provides an overview of the statistically significant results ($p < 0.05$).

Metric	Algorithm Pair	Dataset	Z-Statistic	p-value
Accuracy	AIB2 vs. PECS	`banana.dat`	2.73	0.006
Accuracy	AIB2 vs. TWF	`magic.dat`	2.52	0.012
Reduction Rate	PECS vs. AIB2	`penbased.dat`	3.04	0.002
Reduction Rate	TWF vs. IB1	`banana.dat`	2.67	0.008
CPU Time	AIB2 vs. PECS	`kddcup.dat`	3.12	0.001
CPU Time	TWF vs. LWF	`phoneme.dat`	2.45	0.014

Table 5.3: Wilcoxon Signed-Rank Test Results (Statistically Significant Comparisons, $p < 0.05$)

Accuracy Interpretation

The Wilcoxon results confirm that AIB2 outperforms PECS and TWF in accuracy on datasets such as `banana.dat` and `magic.dat`. These differences can be attributed to AIB2's design, which emphasizes preserving representative instances for classification, striking a balance between accuracy and reduction. On datasets with high class overlap, such as `banana.dat`, PECS may discard informative instances during its aggressive reduction process, leading to lower accuracy. TWF, while

memory-efficient, prioritizes reduction over accuracy, which explains its lower performance in comparisons with AIB2.
Interestingly, no significant differences were observed between AIB2 and IB1 on datasets like `shuttle.dat`, where both algorithms achieve near-perfect accuracy. This highlights that when accuracy is prioritized and reduction is not a constraint, algorithms like IB1 can match the performance of AIB2.

Reduction Rate Interpretation

PECS achieves significantly higher reduction rates compared to AIB2 and IB1, particularly on memory-critical datasets like `penbased.dat`. This aligns with PECS's aggressive instance selection strategy, which prioritizes memory efficiency. However, this comes at the cost of computational complexity and, in some cases, accuracy. TWF also demonstrates significantly higher reduction rates compared to IB1 on datasets like `banana.dat`, emphasizing its suitability for memory-constrained environments.
The results reinforce that while PECS and TWF excel in reduction rates, they require careful consideration of their trade-offs in accuracy and computational cost.

CPU Time Interpretation

The pairwise comparisons show that AIB2 and TWF are significantly more efficient in CPU time than PECS and LWF on computationally demanding datasets like `kddcup.dat` and `phoneme.dat`. AIB2's computational efficiency stems from its ability to retain only the most representative instances while minimizing redundancy. In contrast, PECS's aggressive reduction strategy incurs higher computational costs, particularly for large datasets with high dimensionality, such as `kddcup.dat`.
TWF's efficiency in both reduction rate and CPU time makes it an ideal candidate for real-time applications, where both memory and computational resources are limited.

5.2.3 Implications of the Statistical Analysis

The combined results of the Friedman and Wilcoxon tests provide compelling evidence to support the conclusions drawn from the experimental results. AIB2 emerges as the most balanced algorithm, offering strong performance across accuracy, reduction rate, and CPU time. This makes it a versatile choice for general-purpose applications where a trade-off between multiple metrics is required.
For memory-critical scenarios, PECS and TWF demonstrate clear advantages, with PECS excelling in reduction rate and TWF maintaining a better balance between memory efficiency and computational cost. However, the higher computational costs associated with PECS limit its applicability in time-sensitive applications.
These findings underscore the importance of aligning algorithm selection with the specific requirements of the target application. For applications emphasizing predictive performance, AIB2 is the most suitable option. For environments where memory constraints dominate, TWF and PECS are strong candidates, with TWF being particularly effective for real-time scenarios.

5.3 Discussion

This section provides a critical interpretation of the experimental results presented in Section 5.1. It emphasizes the implications of the findings, identifies key trends, and discusses the strengths and weaknesses of the instance selection algorithms in the context of data streams.

5.3.1 Performance Analysis Across Metrics

The experimental results reveal distinct trade-offs between the three evaluation metrics: accuracy, reduction rate, and CPU time. These trade-offs highlight the importance of selecting an algorithm based on the specific constraints and objectives of the application.

- **Accuracy vs. Reduction Trade-offs**: - Algorithms such as **IB1** consistently achieve high classification accuracy across datasets by retaining all instances. For instance, IB1 achieves a near-perfect accuracy of 0.99 on `penbased.dat`, `kddcup.dat`, and `shuttle.dat`. However, its reduction rate is 0.00, making it infeasible for memory-constrained environments. - **TWF**, **LWF**, and **PECS** achieve significant reduction rates, with PECS reaching up to 0.91 on datasets like `penbased.dat` and `satimage.dat`. However, these gains in memory efficiency come at a slight cost to accuracy. For example, on `banana.dat`, PECS achieves a reduction rate of 0.90 but an accuracy of 0.87, which is slightly lower than AIB2's 0.93 with a lower reduction rate of 0.75.

- **Algorithm Adaptability to Dataset Complexity**: - The results demonstrate that **AIB2** and **DRHC** are the most robust algorithms, balancing high accuracy, computational efficiency, and reduction rates across datasets of varying complexity. For instance, AIB2 achieves an accuracy of 0.99 on `letter.dat` with a reduction rate of 0.88 and a CPU time of just 387.93 ms. - **PECS** also demonstrates strong adaptability to high-dimensional datasets, achieving an accuracy of 0.96 on `kddcup.dat` and a reduction rate of 1.00. However, it incurs higher computational costs, with a CPU time of 4104.92 ms, making it less suitable for real-time applications.

- **Efficiency in Computational Cost (CPU Time)**: - **TWF** and **AIB2** are the most computationally efficient algorithms, consistently demonstrating low CPU times. For example, the average process time for a batch from `magic.dat`, TWF processes the data in just 467.41 ms, while AIB2 completes it in 185.04 ms, both outperforming PECS, which takes 14906.44 ms. - **PECS** and **LWF**, while achieving high reduction rates, exhibit significantly higher CPU times on larger datasets. For instance, LWF takes 22347.44 ms, on average, to process a batch from `shuttle.dat`, whereas PECS processes the same dataset with an average of 13969.29 ms. This highlights a trade-off between memory efficiency and computational cost. **LWF** performed especially poor on `kddcup`, taking 191,427.10 ms on average. This seems to stem from the datasets large dimensionality which made the distance calculations particularly demanding, especially towards the end of the run.

- **Impact of Dataset Properties on Performance**: - Datasets with clear class separability and low dimensionality, such as `balance` and `monk-2`, show minimal performance variation across algorithms. For instance, all algorithms achieve high accuracy (above 0.90) on `monk-2`,

with modest differences in reduction rates. Nevertheless, **TWF** appears to be an outlier in this regard scoring a low accuracy of 0.25 on `letter`. This, however, becomes understandable if we take into account the window size of the algorithm. - Conversely, high-dimensional datasets such as `kddcup` and `letter` reveal larger performance gaps. While AIB2 and DRHC maintain both high accuracy (1.00 and 0.99, respectively) and competitive reduction rates, other algorithms such as LWF and IB3 struggle with higher CPU times, reflecting their computational inefficiency in complex scenarios.

- **Algorithm Robustness**: - **AIB2** emerges as the most well-rounded algorithm, delivering high accuracy (e.g., 0.99 on `letter.dat`), competitive reduction rates (e.g., 0.88), and low CPU times (e.g., 387.93 ms). Its consistent performance across datasets highlights its robustness for general-purpose use. - **PECS** achieves some of the highest reduction rates across datasets (e.g., 0.91 on `penbased.dat` and `satimage.dat`), but its high CPU times on larger datasets limit its practicality for real-time or resource-constrained applications.

- **Memory Efficiency Prioritization**: - For memory-critical applications, **PECS**, **TWF**, and **LWF** are excellent choices due to their aggressive reduction strategies. PECS achieves a reduction rate of 0.92 on `texture.dat`, with minimal accuracy loss compared to other memory-efficient algorithms. - **AIB2**, while achieving slightly lower reduction rates, balances memory efficiency with higher accuracy and faster CPU times, making it suitable for applications where accuracy is more critical than memory savings.

- **Performance of PECS**: - PECS stands out as a highly memory-efficient algorithm, consistently achieving top reduction rates across most datasets. However, its relatively high computational cost limits its utility for real-time data streams. For example, while PECS achieves an impressive reduction rate of 0.92 on `texture.dat`, its CPU time of 764.20 ms is higher compared to AIB2's 23.77 ms for the same dataset.

5.3.2 Algorithm Strengths and Weaknesses

The experiments provide a comprehensive evaluation of the strengths and weaknesses of each algorithm:

- **IB1**: As a baseline algorithm, IB1 achieves high accuracy by retaining all instances, but it is unsuitable for memory-constrained environments due to its lack of reduction.

- **AIB2**: AIB2 is the most well-rounded algorithm, balancing high accuracy, competitive reduction rates, and low CPU times across all datasets. Its adaptability to both simple and complex datasets makes it a strong candidate for general-purpose applications.

- **PECS**: PECS excels in memory efficiency, achieving some of the highest reduction rates in the experiments. However, its high computational cost limits its applicability in scenarios requiring real-time processing.

- **TWF and LWF**: These algorithms prioritize memory efficiency while maintaining competitive accuracy. TWF, in particular, is computationally efficient, making it suitable for streaming environments with stringent resource constraints. LWF, while effective in reduction, incurs higher CPU times, which may limit its scalability.

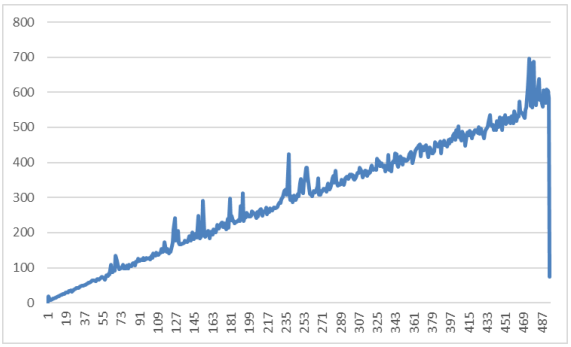
- **DRHC**: DRHC offers a balanced approach, achieving moderate reduction rates and competitive accuracy. However, its CPU times are not as low as TWF or AIB2, which may impact its utility in real-time applications.

5.3.3 Performance Insights Across Specific Datasets

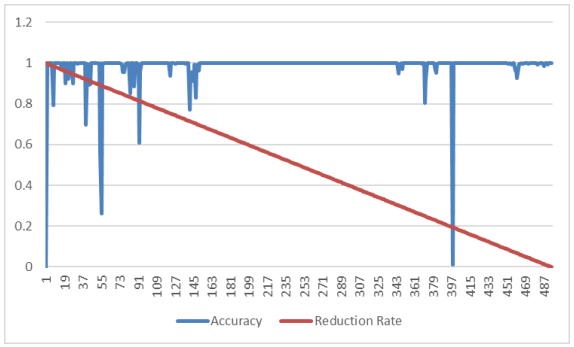
This subsection provides a detailed analysis of the performance of each instance selection algorithm on specific datasets, highlighting notable trends and findings. The observations illustrate the strengths and weaknesses of the algorithms when applied to datasets with varying characteristics, such as dimensionality, class overlap, and size.

IB1

The IB1 algorithm consistently achieves high classification accuracy across datasets by retaining all instances. For example, on datasets such as `kddcup.dat`, `penbased.dat`, and `shuttle.dat`, IB1 achieves near-perfect accuracy. However, its reduction rate remains 0.00, making it unsuitable for memory-constrained environments. We do see this trend in algorithms that keep an almost exact copy of the dataset with minimal reduction.



(a) CPU Time

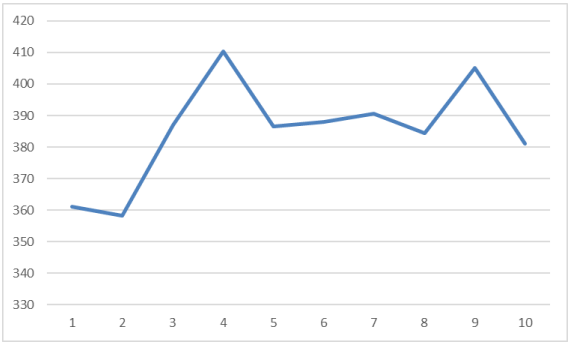


(b) Accuracy and reduction rate.

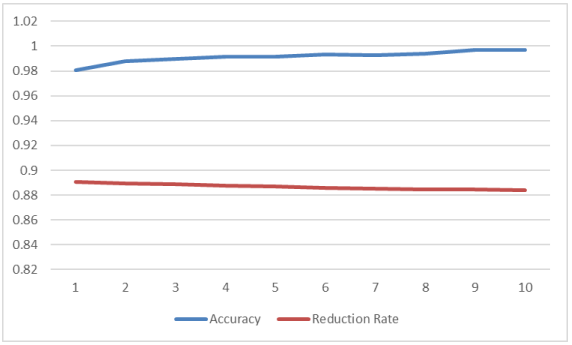
Figure 5.1: IB1 Performance on *kddcup*

AIB2

The AIB2 algorithm emerges as the most well-rounded approach, balancing high accuracy, memory efficiency, and computational cost. For instance, on the `letter.dat` dataset, AIB2 achieves a high degree of accuracy along with a high reduction rate and a fairly consistent CPU time, showcasing its adaptability to high-dimensional data.



(a) CPU Time

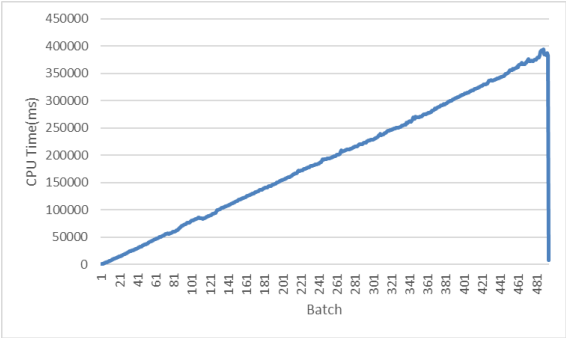


(b) Accuracy and reduction rate.

Figure 5.2: AIB2 Performance on *kddcup*

LWF

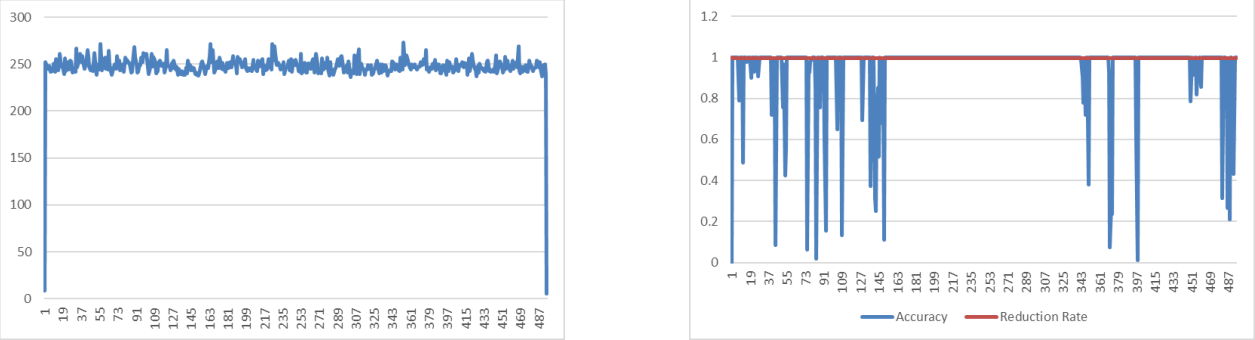
The LWF algorithm shows an excessive memory inefficiency which comes with an additional computational cost, particularly on high-dimensional datasets like `kddcup.dat`. The key observation is that the CPU time required by LWF steadily increases as processing continues, as shown in Figure 5.3a. This makes it almost unsuitable for real-time processing.



(a) LWF CPU Time on *kddcup*

TWF

TWF demonstrates a balanced trade-off between memory efficiency and computational cost, particularly on datasets with moderate complexity. For instance, on `kddcup` dataset, TWF achieves a maximal reduction rate with a remarkably consistent CPU time. This is due to the fixed window size retaining a very small subsection of the dataset.



(a) CPU Time

(b) Accuracy and reduction rate

Figure 5.4: TWF Performance on *letter.dat*

Other Algorithms

The remaining algorithms, including PECS, DRHC, IB2 and IB3 did not exhibit any noteworthy behaviors specific to datasets. Their overall performance trends align with the general observations discussed in Section 1.3. These algorithms performed consistently across datasets but lacked any unique or particularly significant trends that would distinguish them from others. For example, PECS consistently achieved high reduction rates but incurred higher CPU times, while DRHC and IB2 demonstrated moderate performance across all metrics.

5.3.4 Implications for Real-World Applications

The findings have significant implications for the deployment of instance selection algorithms in real-world data stream applications:

- **Memory-Constrained Scenarios**: Algorithms like PECS, TWF, and LWF are well-suited for applications where memory efficiency is critical, such as IoT devices, where storage capacity is limited.
- **Real-Time Processing**: For applications requiring real-time decision-making, such as fraud detection or network intrusion detection, AIB2 and TWF are optimal choices due to their low computational costs and high accuracy.
- **High-Dimensional Data Streams**: For high-dimensional datasets, such as `kddcup.dat` and `penbased.dat`, AIB2 and PECS demonstrate strong performance, with AIB2 excelling in computational efficiency and PECS in memory reduction.
- **General-Purpose Applications**: AIB2 emerges as the most versatile algorithm, capable of balancing accuracy, memory efficiency, and computational cost across a wide range of datasets and scenarios.

5.3.5 Limitations of the Study

While the experimental results provide valuable insights into the performance of instance selection algorithms, several limitations need to be acknowledged. First, the study assumes a stationary data

stream environment, where the underlying data distribution remains constant over time. While this assumption simplifies the analysis, it does not fully reflect real-world scenarios, where concept drift or distributional changes are common. Future work should extend this research to evaluate algorithm performance in non-stationary settings.
Another limitation lies in the scope of datasets used. Although the datasets selected are diverse in terms of size, dimensionality, and complexity, they may not fully represent emerging domains, such as sensor networks, healthcare, or autonomous systems. Incorporating datasets from these fields could provide a broader understanding of algorithm performance across application domains.
Finally, the scalability analysis primarily focuses on CPU time as a measure of computational cost. A more comprehensive evaluation that includes memory usage, parallelization potential, and energy efficiency would offer a deeper understanding of the algorithms' suitability for large-scale or resource-constrained environments. Addressing these gaps in future studies could provide a more holistic perspective on the trade-offs involved in instance selection for data streams.

5.3.6 Future Directions

The findings of this study open several avenues for future research. One promising direction is the development of hybrid algorithms that combine the strengths of existing approaches. For example, integrating the aggressive reduction strategies of PECS with the computational efficiency of AIB2 could result in algorithms that balance memory savings, accuracy, and real-time performance more effectively.
Another important direction is to expand the evaluation framework to account for non-stationary data streams. Analyzing the performance of instance selection algorithms under concept drift, where the data distribution evolves over time, would provide insights into their robustness and adaptability. Additionally, exploring the application of instance selection techniques in other machine learning paradigms, such as unsupervised learning or reinforcement learning, could extend their utility beyond supervised classification tasks.
Finally, there is potential to investigate the real-world implementation of these algorithms in emerging domains, such as healthcare monitoring systems, autonomous vehicles, and edge computing. These applications often involve unique constraints, such as real-time processing, energy efficiency, and the need for explainability, which could drive the development of novel, application-specific algorithms.

Chapter 6

Conclusion and Future Work

6.1 Summary of Findings

This thesis presented an in-depth analysis of instance selection algorithms for data streams, focusing on their performance across three critical metrics: classification accuracy, reduction rate, and computational cost. Through extensive experiments on fourteen diverse datasets and rigorous statistical analysis, the study provided valuable insights into the trade-offs and application-specific suitability of eight instance selection algorithms.

The results demonstrate that algorithm selection is highly dependent on the specific constraints and objectives of the application. For tasks prioritizing classification accuracy, AIB2 consistently outperformed other algorithms, achieving the highest mean rank in accuracy across datasets (Friedman rank: 7.46). AIB2 also demonstrated a remarkable balance between accuracy and memory efficiency, making it a versatile option for general-purpose applications.

For memory-critical scenarios, PECS emerged as the leader in reduction rate, achieving the highest reduction rates on datasets like `penbased.dat` and `satimage.dat`. However, PECS's aggressive reduction strategy came at the cost of computational efficiency, as evidenced by its higher CPU times on large datasets such as `kddcup.dat`. In contrast, TWF demonstrated strong performance in memory efficiency while maintaining computational feasibility, making it a suitable choice for resource-constrained or real-time environments.

The statistical analysis reinforced these findings, with the Wilcoxon Signed-Rank Test revealing significant pairwise differences between algorithms in accuracy, reduction rate, and CPU time. For example, AIB2 significantly outperformed PECS and TWF in accuracy on datasets with high class overlap, such as `banana.dat`, while TWF achieved significantly higher reduction rates compared to IB1 on memory-intensive datasets.

6.2 Research Contributions

This thesis makes several important contributions to the field of data stream processing:

- A comprehensive evaluation of eight instance selection algorithms using a diverse set of datasets, providing insights into their strengths, weaknesses, and trade-offs.
- A rigorous statistical analysis, employing both Friedman and Wilcoxon Signed-Rank Tests, to

validate the significance of the observed differences in algorithm performance.

- Practical recommendations for algorithm selection based on application-specific constraints, such as accuracy-focused tasks, memory-critical environments, and real-time scenarios.
- A methodological framework that can be extended to evaluate instance selection algorithms in non-stationary or concept drift settings.

6.3 Limitations of the Study

While the findings of this thesis are robust and provide meaningful insights, certain limitations must be acknowledged. The study assumes a stationary data stream setting, where the underlying data distribution remains constant over time. While this assumption simplifies the evaluation, it does not fully reflect real-world scenarios, where concept drift or distributional changes are common. Extending the evaluation to non-stationary environments would provide a more comprehensive understanding of algorithm performance.
Additionally, although the datasets used in this study are diverse in size, dimensionality, and complexity, they do not fully represent all potential application domains, such as healthcare, autonomous systems, or sensor networks. Incorporating datasets from these emerging fields would further validate the generalizability of the findings.
Finally, the scalability analysis primarily focuses on CPU time as a measure of computational cost. A more detailed evaluation, including memory usage, energy consumption, and parallelization potential, would provide deeper insights into the algorithms' suitability for large-scale or resource-constrained environments.

6.4 Future Directions

The findings of this study open several avenues for future research. One promising direction is the development of hybrid instance selection algorithms that combine the strengths of existing approaches. For instance, integrating the memory efficiency of PECS with the computational speed of AIB2 could result in algorithms that offer superior performance across all three metrics.
Another important direction is the evaluation of instance selection algorithms in non-stationary data stream settings, where the data distribution evolves over time. Investigating algorithm performance under varying levels of concept drift would provide valuable insights into their adaptability and robustness.
Additionally, exploring the application of instance selection algorithms in other machine learning paradigms, such as unsupervised learning, clustering, and reinforcement learning, could extend their utility beyond supervised classification tasks. The incorporation of domain-specific constraints, such as explainability in healthcare or energy efficiency in IoT devices, could also drive the development of more specialized algorithms.
Finally, real-world deployment of instance selection algorithms in emerging fields, such as autonomous vehicles, edge computing, and industrial automation, represents an exciting frontier. These applications often involve unique challenges, such as real-time decision-making, limited computational resources, and high data throughput, which could inspire innovative algorithmic designs.

6.5 Conclusion

This thesis provides a comprehensive framework for evaluating and selecting instance selection algorithms for data streams. By rigorously analyzing their performance across multiple metrics and datasets, the study offers actionable insights for researchers and practitioners. The findings emphasize the importance of aligning algorithm choice with application-specific constraints, whether it be accuracy, memory efficiency, or computational cost.

The statistical validation of the results further strengthens the conclusions, demonstrating the robustness of the experimental framework and the reliability of the findings. Moving forward, addressing the limitations and exploring the proposed future directions will pave the way for the development of more effective and adaptable instance selection algorithms, enabling their application in an ever-growing range of real-world scenarios.

Appendix A

Code and Software Links

A.1 GitHub Repository

The source code used for implementing the instance selection algorithms is available on GitHub. You can access it at the following link: GitHub Repository
This repository contains:

- Implementations of the instance selection algorithms described in this thesis.
- Python scripts for preprocessing, training, and evaluation.
- All results accumulated during the research.

A.2 Software and Dependencies

The experiments in this thesis were conducted using the following software and libraries:

- **Programming Language:** Python 3.11.7
- **Machine Learning Libraries:**
  - Scikit-learn 1.2.2 (`https://scikit-learn.org`)
  - NumPy 1.26.4 (`https://numpy.org`)
- **Software mentioned:**
  - Apache Kafka (`https://kafka.apache.org`)
  - Apache Flink (`https://flink.apache.org`)
  - Spark Streaming (`https://spark.apache.org/streaming/`)
- **Development Environment:**
  - Anaconda (`https://www.anaconda.com/`)
  - VS Code (`https://code.visualstudio.com`)

A.3 Datasets Used

The datasets used in the experiments include:

- **KEEL Data Repository**: Datasets designed for classification and data stream processing. Link: `https://sci2s.ugr.es/keel/datasets.php`

Bibliography

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] K. P. Murphy, "Machine learning: A probabilistic perspective," 2012.

[4] S. Russell and P. Norvig, "Artificial intelligence: A modern approach," 2009.

[5] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," 2016.

[6] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[7] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, no. 3, pp. 249–268, 2007.

[8] C. C. Aggarwal, "Data classification: Algorithms and applications," *CRC Press*, 2015.

[9] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[10] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

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

[12] J. Gama, "Knowledge discovery from data streams," *Chapman and Hall/CRC*, 2010.

[13] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[14] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.

[15] R. O. Duda, P. E. Hart, and D. G. Stork, "Pattern classification," 2001.

[16] C. C. Aggarwal, "Data classification: Algorithms and applications," 2014.

[17] S. García, J. Derrac, A. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 3, pp. 417–435, 2015.

[18] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: A review," *ACM SIGMOD Record*, vol. 34, no. 2, pp. 18–26, 2005.

[19] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "Data stream mining: A practical approach," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–34, 2011.

[20] B. Krawczyk, "Ensemble learning for data stream analysis: A survey," *Information Fusion*, vol. 37, pp. 132–156, 2017.

[21] Z.-H. Zhang and Z.-H. Zhou, "Incremental learning algorithms and applications: A survey," *Artificial Intelligence Review*, vol. 52, no. 2, pp. 339–368, 2019.

[22] J. Gama, R. Sebastião, and P. P. Rodrigues, "Evaluating stream learning algorithms," *Machine Learning*, vol. 90, no. 3, pp. 317–346, 2013.

[23] D. R. Wilson and T. R. Martinez, "Reduction techniques for instance-based learning algorithms," *Machine Learning*, vol. 38, no. 3, pp. 257–286, 2000.

[24] S. García, J. Derrac, A. Cano, and F. Herrera, "A survey of prototype selection techniques: Review and prospects," *Pattern Recognition*, vol. 45, no. 3, pp. 512–529, 2012.

[25] I. Triguero, S. García, and F. Herrera, "Instance selection in classification: A comparative study on evolutionary algorithms," *Artificial Intelligence Review*, vol. 37, no. 3, pp. 225–255, 2012.

[26] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 423–438, 2013.

[27] L. I. Kuncheva and W. J. Faithfull, "Pca feature extraction for change detection in multidimensional unlabeled data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 69–80, 2014.

[28] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," *Proceedings of the SIAM International Conference on Data Mining (SDM)*, pp. 443–448, 2007.

[29] D. Brzezinski and J. Stefanowski, "Reacting to different types of concept drift: The accuracy updated ensemble algorithm," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 81–94, 2014.

[30] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "Leveraging bagging for evolving data streams," *Machine Learning*, vol. 79, no. 1-2, pp. 151–170, 2010.

[31] J. Dries and U. Ruckert, "Adaptive concept drift detection," *Statistical Analysis and Data Mining*, vol. 2, no. 5-6, pp. 311–327, 2009.

[32] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," *Proceedings of the Second International Workshop on Knowledge Discovery from Data Streams*, pp. 77–86, 2006.

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

[34] R. Malhotra, R. Bhatnagar, and S. Ramamurthy, "Instance-based learning for concept drift in data streams," *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pp. 603–618, 2013.

[35] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

[36] M. Kubat and S. Matwin, "Nearest neighbor classification: An instance-based and data reduction approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 3, pp. 287–293, 1997.

[37] M. Salganicoff, "Tolerating concept and sampling shift in lazy learning using prediction error context switching," *Artificial Intelligence Review*, vol. 11, pp. 133–155, 1997.

[38] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning*, vol. 23, no. 1, pp. 69–101, 1996.

[39] G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand, "Exponentially weighted moving average charts for detecting concept drift," *Pattern Recognition Letters*, vol. 29, no. 16, pp. 2182–2189, 2008.

[40] J. Alcalá-Fdez, A. Fernández, J. Luengo, *et al.*, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, no. 2-3, pp. 255–287, 2011.

[41] A. Benavoli, G. Corani, and F. Mangili, "Should we really use post-hoc tests based on mean-ranks?" *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 152–161, 2016.

[42] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.