Karlsruher Institut für Technologie

Efficient Decision-Making in Data Streams under Limited Feedback

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Marco Heyden

aus Weinheim

Tag der mündlichen Prüfung: 13. Februar 2025
1. Referent: TT-Prof. Dr. Pascal Friederich
2. Referent: Assistant Prof. Dr. Heitor Murilo Gomes
Betreuer: Prof. Dr.-Ing. Klemens Böhm

Acknowledgements

When reading other people's acknowledgments, one might come to the conclusion that doing a PhD is a traumatic experience. Luckily, for me, this was not the case. My time at IPD (and FZI before that) was enjoyable 95 % of the time and, at worst, occasionally stressful. The reasons for these positive reflections are the many people who supported me along the way and to whom I owe the deepest gratitude.

First and foremost, my parents, Sabine and Dirk, as well as my sister, Monique, have provided unconditional support throughout my entire life. Likewise, my partner, Eva, has an incredible ability to read my mind, always believed in me, and made me "chill out a bit" whenever I was struggling during my PhD. I am deeply grateful for the strong bond we share and could not wish for a better family.

I would also like to thank Prof. Böhm for always having my back. He not only ensured the funding of our chair and enabled me to undertake a research visit on the other side of the world (literally!), but also shared his valuable experience, ideas, and commitment, which not only elevated my research but also fostered my personal development.

On a day-to-day basis, I collaborated closely with my PhD supervisor, Vadim Arzamazov. Over the three years we worked together, I have come to appreciate not only Vadim's outstanding qualities as a researcher and group leader but also his character. His kindness, reliability, dedication, sense of fairness, and eagerness to learn continue to amaze me.

I am also grateful to Edouard Fouché, who introduced me to academia in 2018 during my Master's thesis. Since then, we have worked closely together. Edouard is one of the most supportive people I know, and his impact on this dissertation has been significant—whether through sharing his knowledge, asking difficult questions, or providing honest feedback.

Of all the people at IPD, my office mate Federico Matteucci, was probably the one I spent the most time with. We not only had lively discussions on the various whiteboards in our office, but also established rituals, such as grabbing a *caffettóne* (a neologism meaning 'huge coffee' in Italian) each morning. I will surely miss this in the future.

I am also grateful to Heitor Gomes for welcoming me and putting up with me for three months at Victoria University of Wellington, and to Pascal Friederich for his support during the final stretch. Thank you as well to Daniel, Edouard, Federico, Florian, and Vadim for proofreading this dissertation.

Last but not least, I want to give a huge shout-out to all my colleagues at IPD, with whom I had the pleasure of working, and to all my students whom I supervised. Thank you all for making my PhD such a pleasant experience—one that I will be happy to look back on!

Abstract

Decision-making is fundamental to any intelligent system and crucial in countless situations, including everyday life and industrial applications, such as process control and machine operation, online marketing, and fraud detection. A decision-maker observes the present state of the environment and subsequently makes a decision. Then, the decision-maker receives feedback from the environment, which can depend on the decision taken. They can use the feedback to improve their decision-making strategy. However, due to the complexity and potential stochasticity of the environment, human decision-making can be biased and prone to errors. This calls for decision support systems (DSS) that provide guidance to the decision-maker and thus allow more informed decisions.

Developing decision support systems is challenging for several reasons. First, new information, such as environment states and feedback, becomes available over time. A DSS must therefore adapt as new information arrives. Second, decisions may be required very quickly. For instance, online fraud detection systems may need to classify thousands of financial transactions per hour as valid or fraudulent, requiring near-real-time processing. Third, real-world environments are often dynamic and evolve over time. Scammers, for example, continually devise new methods to bypass fraud detection systems. To remain effective, a DSS must adapt to such changes.

Machine learning algorithms for data streams—potentially infinite sequences of data points that arrive over time—meet these demands. They can incorporate new information incrementally, process data quickly, and adapt to dynamic environments, making them well-suited for the development of DSS. However, many real-world applications challenge the assumption that sufficient feedback is available to train these algorithms. In some applications, e.g., online advertising, one can observe feedback only for the decision taken, but not for alternative decisions. In other applications, e.g., industrial process monitoring, it is impossible to label each data point, due to the amount of data generated and the complexity of obtaining it. In addition, receiving feedback can be costly. For instance, an online advertiser has to pay a cost when displaying ads to potential customers, and inspecting failures in industrial machines may require temporary shutdowns that cause a decrease in productivity. This dissertation addresses these challenges by developing data stream algorithms that learn effectively under limited or no feedback, approaching the problem from three perspectives.

The dissertation first focuses on applications where feedback is costly and limited to the chosen decision, known as the budgeted multi-armed bandit (BMAB) setting. It occurs in, for instance, online advertising, where a decision-maker with limited budget must find out efficiently which advertisement works best for a group of potential customers. We

illustrate the challenges that cause existing algorithms to struggle in this setting and then address these challenges with a novel UCB-sampling algorithm, ω-UCB, that relies on asymmetric confidence intervals. Our approach is theoretically justified and outperforms existing algorithms empirically.

Second, the dissertation addresses a scenario where feedback may be limited to only some observations. This is common in applications like fraud detection, which generate data at a fast pace but require decisions in real time. Decision trees are popular algorithms for this scenario due to their interpretability, ability to handle numerical and categorical data, and robustness to outliers. However, the most widely used decision trees for data streams are feedback inefficient, as we illustrate in this work. We thus present a new feedback-efficient algorithm, PLASTIC, that uses decision tree restructuring to achieve feedback efficiency.

Third, it considers situations where feedback is entirely unavailable. This is common in industrial applications, due to missing feedback about malfunctions or failures of complex machines. Here, decision-makers may resort to unsupervised algorithms that generate feedback based on the state of the environment. We propose the Adaptive Bernstein Change Detector (ABCD), a new algorithm for detecting and characterizing changes in possibly high-dimensional data streams. In comparison to existing algorithms, it provides rich feedback about when a change occurs, which dimensions are affected, and its magnitude.

To summarize, this dissertation improves decision-making under limited feedback by contributing

(I) ω-UCB, a UCB-sampling algorithm for decision making under budget constraints and costly feedback, presented at the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining,

(II) PLASTIC, a novel feedback-efficient incremental decision tree algorithm, presented at the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2024), and

(III) ABCD, an unsupervised concept drift detector and descriptor for high-dimensional data streams, published in 'Data Mining and Knowledge Discovery' (Springer).

Zusammenfassung

Entscheidungsfindung ist grundlegend für jedes intelligente System und allgegenwärtig in Alltagssituationen und industriellen Anwendungen wie Prozesssteuerung, Maschinenbetrieb, Online-Marketing und Betrugserkennung. Ein Entscheidungsträger beobachtet den aktuellen Zustand der Umgebung und trifft daraufhin eine Entscheidung. Anschließend erhält er Rückmeldung ("Feedback") aus der Umgebung, welche von seiner Wahl abhängen kann. Das Feedback kann schließlich dazu genutzt werden, die Entscheidungsstrategie zu verbessern. Aufgrund der Komplexität und möglichen Unsicherheit der Umgebung ist die menschliche Entscheidungsfindung jedoch potenziell verzerrt und fehleranfällig. In solchen Situationen können computergestützte Hilfssysteme zur Entscheidungsfindung (decision support system (DSS)) dabei helfen, fundierte Entscheidungen zu treffen.

Die Entwicklung von DSS ist aus mehreren Gründen herausfordernd. Erstens werden neue Informationen, wie Umgebungszustände und Feedback, erst sukzessive verfügbar. Ein DSS sollte sich daher auf Basis neuer Informationen anpassen können. Zweitens können Entscheidungen sehr schnell erforderlich sein. So müssen Betrugserkennungssysteme tausende Finanztransaktionen pro Stunde nahezu in Echtzeit als gültig oder betrügerisch klassifizieren. Drittens sind reale Umgebungen oft dynamisch und verändern sich im Zeitverlauf. Betrüger entwickeln beispielsweise kontinuierlich neue Methoden, um Betrugserkennungssysteme zu umgehen. Ein DSS muss sich solchen Änderungen anpassen, um wirksam zu bleiben.

Maschinelle Lernalgorithmen für Datenströme—Sequenzen von fortlaufend eintreffenden Datenpunkten—erfüllen diese Anforderungen. Sie können von neuen Informationen schrittweise lernen, Daten nahezu in Echtzeit verarbeiten und sich an dynamische Umgebungen anpassen. Damit eignen sie sich ideal für die Entwicklung von DSS. Bestehende Algorithmen benötigen jedoch ausreichend Feedback, um effektiv zu lernen. Dieses aber ist in vielen realen Anwendungen nicht in erforderlichem Maße verfügbar. In manchen Anwendungen, wie z. B. Online-Werbung, ist Feedback nur für die getroffene Entscheidung verfügbar, nicht jedoch für deren Alternativen. In anderen Anwendungen, wie z. B. der Überwachung von Industrieprozessen, ist es aufgrund der Datenmenge und Komplexität nicht möglich, Rückmeldung für jeden Datenpunkt zu erhalten. Zudem kann der Erhalt von Feedback mit Kosten verbunden sein. Die Anzeige von Werbung in Suchmaschinen ist beispielsweise kostspielig und die Analyse von Maschinenausfällen kann zeitweisen Prozessstillstand und damit Produktivitätsverluste verursachen. Die vorliegende Dissertation widmet sich diesen Herausforderungen aus drei Perspektiven und stellt Datenstromalgorithmen vor, welche effektiv von eingeschränktem oder gar fehlendem Feedback lernen.

Als Erstes konzentriert sich die Dissertation auf Anwendungen, in denen Feedback mit Kosten verbunden und auf die gewählte Entscheidung beschränkt ist. Dieses Szenario ist als budgeted multi-armed bandit (BMAB) bekannt. Es tritt beispielsweise in Online-Werbekampagnen auf, in denen ein Entscheidungsträger mit begrenztem Budget effizient herausfinden muss, welche Anzeige für eine Gruppe potenzieller Kunden am besten funktioniert. Zu Beginn werden Herausforderungen aufgezeigt, welche bestehende BMAB-Algorithmen überwinden müssen. Im Anschluss wird ein neuer Algorithmus basierend auf upper confidence bound (UCB)-Sampling, namens ω-UCB, vorgestellt, welcher diese Herausforderungen mittels asymmetrischer Konfidenzintervalle löst. Der Ansatz ist theoretisch fundiert und bestehenden Algorithmen empirisch überlegen.

Als Zweites behandelt die Dissertation Szenarien, in denen Feedback nur für manche Beobachtungen verfügbar ist. Dies ist üblich in Anwendungen wie der Betrugserkennung, in denen neue Daten schnell generiert werden und gleichzeitig Entscheidungen in Echtzeit erfordern. Entscheidungsbäume sind aufgrund ihrer Interpretierbarkeit, ihrer Fähigkeit, numerische und kategorische Daten zu verarbeiten, sowie ihrer Robustheit gegenüber Ausreißern beliebte Algorithmen für solche Szenarien. Jedoch sind die am weitesten verbreiteten Entscheidungsbäume für Datenströme ineffizient im Umgang mit Feedback. Die Dissertation stellt daher einen neuen Algorithmus, PLASTIC, vor, der durch Umstrukturierung Feedback-Effizienz erreicht.

Als Drittes werden Situationen betrachtet, in denen kein Feedback verfügbar ist. Üblich ist dies beispielsweise in industriellen Anwendungen, da hier oftmals Feedback über Fehlfunktionen oder Ausfälle komplexer Maschinen fehlt. Entscheidungsträger können dann auf unüberwachte Algorithmen zurückgreifen, welche Feedback selbst, basierend auf dem Zustand der Umgebung, generieren. Die Dissertation präsentiert einen neuen Algorithmus, den Adaptive Bernstein Change Detector (ABCD), der Änderungen in potenziell hochdimensionalen Datenströmen erkennt und charakterisiert. Im Vergleich zu bestehenden Algorithmen liefert er differenziertes Feedback darüber, ob eine Änderung aufgetreten ist, welche Dimensionen betroffen waren und wie stark die Änderung war.

Zusammenfassend leistet diese wissenschaftliche Arbeit durch die Entwicklung von

(I) ω-UCB, eines UCB-Sampling-Algorithmus für Entscheidungen unter Budgetbeschränkungen und kostspieligem Feedback, vorgestellt auf der '30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining',

(II) PLASTIC, eines Feedback-effizienten inkrementellen Entscheidungsbaumalgorithmus, präsentiert auf der 'European Conference on Machine and Principles and Practice of Knowledge Discovery in Databases' (ECML PKDD 2024) und

(III) ABCD, eines unüberwachten Algorithmus zur Erkennung und Charakterisierung von Änderungen in hochdimensionalen Datenströmen, veröffentlicht in der Zeitschrift 'Data Mining and Knowledge Discovery' (Springer)

einen signifikanten Beitrag zur Verbesserung und Weiterentwicklung der Entscheidungsfindung unter eingeschränktem Feedback.

Table of Contents

Part I.

Introduction

1. Overview

1.1. Decision-Making in Data Streams

1.1.1. The Decision-Making Process

A decision is a choice made by a decision-maker between competing beliefs about the environment or among alternative actions, with the intent of achieving the decision-maker's goals [Fox01]. Decisions are part of our daily lives. Some decisions are simple, e.g., what to wear in the morning, while others are more complex:

Example 1.1 (Fraud detection)**.** A payment provider has to evaluate in real time whether payments are valid transactions or fraud based on information such as the amount of money, time of day, location, recipient, or user behavior. Failure to detect fraud is expensive, but false flagging of valid transactions as fraud has a negative impact on customer satisfaction [Car+18].

Example 1.2 (Online advertising)**.** A market researcher has to make several decisions when advertising a new product, e.g., what customer group to target, how to design the ads, or which advertisement channels to choose. Ineffective marketing campaigns lead to high advertising costs and low profits [Yua+12].

Example 1.3 (Predictive maintenance)**.** An operator of a chemical production plant has to continuously judge whether all the subsystems and processes are working correctly or whether some part of the plant shows irregularities that require intervention. Failing to identify these irregularities can lead to extended downtimes, while early detection allows the operator to address issues before they escalate. However, frequent early interventions can also become costly due to replacement parts and the short but frequent downtimes involved [FCM22].

Simple decisions are based on our experience and a clear understanding of the nature of events and their causal effects, such as 'my clothes get wet if it rains, so I should wear a raincoat'. But complex decisions, e.g., the ones faced by a payment provider, market researcher, or plant operator, require repeatedly gathering relevant experience by trial and error. Figure 1.1 illustrates this sequential decision-making process. At time t, the decision-maker observes the environment state $\mathbf{x}_t$ and chooses a decision k_t. At time $t+1$,

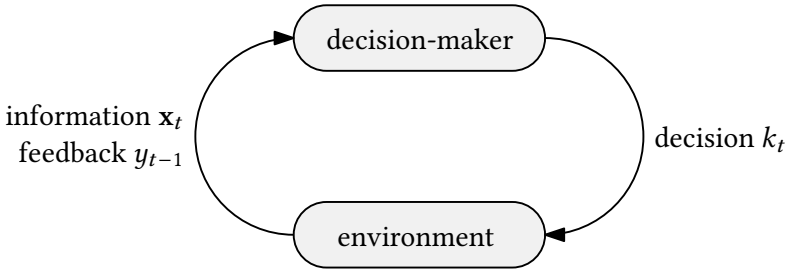


Figure 1.1.: Sequential decision-making

the environment provides the decision-maker with feedback y_t about the decision taken at time t. The decision-maker can use y_t to improve their decision-making strategy.

It is well-known, however, that human decision-making is prone to systematic errors and biases. In particular, when decision-making is complex or feedback is subject to uncertainty [TK74]. The following two challenges capture this.

C1. **Complexity:** Information about the environment can be arbitrarily complex. For instance, the state $\mathbf{x}_t$ might be represented by high-dimensional data, e.g., images or readings from many sensors within a production plant. Additionally, $\mathbf{x}_t$ can involve a combination of data types from various domains, including continuous, discrete, ordinal, and categorical variables. These sources of complexity challenge the decision-maker to make informed decisions.

C2. **Stochasticity:** In many real-world situations, outcomes are influenced by randomness. For instance, consider a decision-maker in charge of selecting online advertisements to display to users. The effectiveness of these ads depends on uncertain user behaviors, such as click-through rates or purchase decisions, which vary across users and are further influenced by external factors. Even with historical data, the decision-maker cannot predict exactly how any individual user will respond to a given ad. This inherent uncertainty forces the decision-maker to account for randomness when optimizing decisions over time, balancing exploration (trying out different ads) with exploitation (showing ads that worked well in the past) [LS20].

C1 and C2 motivate the use of intelligent decision support systems (DSS), which guide the decision-maker by simplifying the decision-making process or by taking uncertainty into account [Fox01; Sàn22]. Figure 1.2 illustrates a DSS-guided decision-making process. The DSS provides the decision-maker with surrogate feedback $\hat{y}_t$ for observation $\mathbf{x}_t$. In contrast to the original decision-making process (Figure 1.1) in which feedback about $\mathbf{x}_t$ is only revealed at time $t+1$, the surrogate feedback $\hat{y}_t$ can guide the decision-maker already at time step t.

1.1.2. Designing Decision Support Systems

The usefulness of a DSS—simply called *model* from now on—depends heavily on the reliability of the feedback it provides to the decision-maker. Given this requirement, we

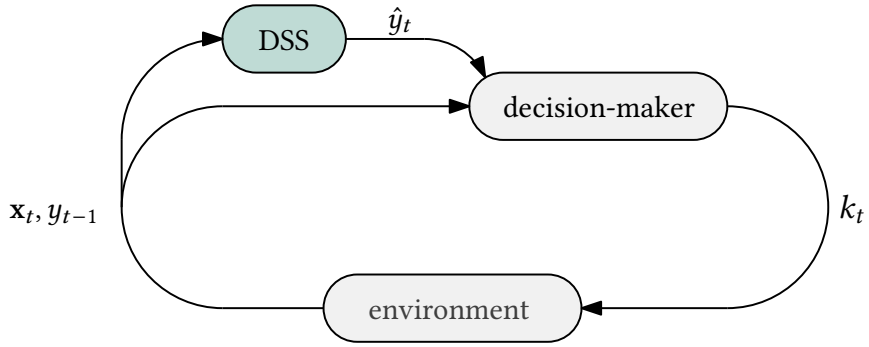


Figure 1.2.: DSS-guided decision-making

aim to develop a model that minimizes some notion of error, or *loss*, between predicted and actual feedback as new data becomes available over time. To accomplish this, one could consider all observation-feedback pairs available up to time t as a dataset S that represents the underlying data distribution. One could then use machine learning (ML) techniques to train a model at each time step, denoted m_t, by minimizing the average loss observed on S.

Figure 1.3 illustrates the basic steps one might follow to create a ML model from data [MRT18]. First, one would randomly partition S into a training set, validation set, and test set. Then, one would train suitable ML algorithms on the training set while varying their hyperparameters to generate multiple candidate models. The validation set would then be used to evaluate these models and select the one that minimizes the validation loss as the best-performing model m_t. Finally, one would evaluate m_t on the test set to assess its performance on unseen data. If the test loss is sufficiently low (indicating good generalization), one could deploy the model.

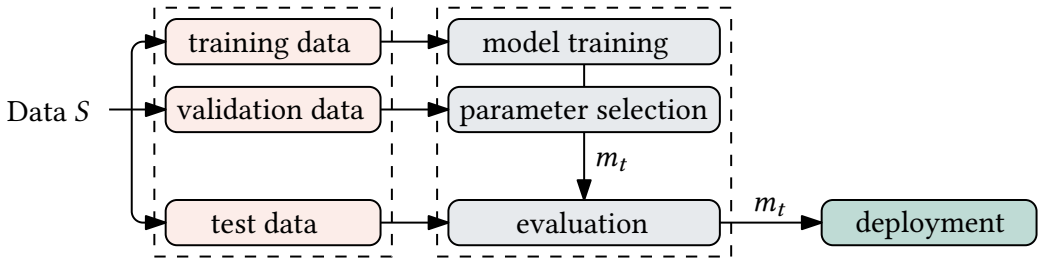


Figure 1.3.: The traditional machine learning pipeline (adapted from [MRT18])

We argue, however, that such a static process of obtaining an ML model from batch data is not suitable for decision-making, due to two additional challenges.

C3. **New information:** In sequential decision-making processes, new data becomes available over time. Ideally, a ML algorithm should thus be able to incorporate new data *incrementally*, i.e., as single observations or in batches. Also, note that ongoing processes accumulate large amounts of data over time. Think, for example, of a stream of sensor data from a production plant that operates day and night. Storing and processing the data repeatedly as a batch quickly becomes impractical or even

infeasible. This calls for models that do not require storing the data and can learn incrementally. [Bif+18a].

C4. **Outdated knowledge**: Real-world environments change over time. Some changes may occur gradually, such as wear and tear in mechanical systems, while others may be abrupt and caused by unforeseen events, like financial crises or natural disasters. This phenomenon is well-known in the literature; some research fields call it *non-stationarity* [LS20], others refer to it as *concept drift* or *change* [Bif+18c]. Concept drift causes ML models to become outdated and perform poorly on newly arriving data. ML models for decision-making should hence be able to adapt to concept drift.

Challenges C3 and C4 have been the main drivers of research on *data streams*, an algorithmic abstraction of sequential data generation processes that facilitates real-time analytics [Bif+18d]. As a result, algorithms for ML for data streams (MLDS) are more suitable for decision-making than algorithms that can only learn from batch data. The next section briefly introduces data stream fundamentals. For a more in-depth discussion, we refer the reader to [Bif+18e; Gam10].

1.1.3. The Data Stream Abstraction

We define a data stream as follows.

Definition 1.1 (Data stream)**.** A *data stream* S is a possibly never-ending sequence of observations $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_t, y_t), \ldots\}$ drawn from an ordered set of distributions $\{\mathcal{S}_{s_1,s_2}, \mathcal{S}_{s_2,s_3}, \mathcal{S}_{s_3,s_4}, \ldots\}$, called *concepts*, such that

$$\forall t \in [s_i, s_{i+1}) : (\mathbf{x}_t, y_t) \overset{\text{i.i.d.}}{\sim} \mathcal{S}_{s_i,s_{i+1}}.$$

That is, a data stream is a concatenation of sequences of observations, where the data points within each sequence are i.i.d. samples from the same concept. The length of sequence i is given by the number of data points that arrived within the time interval $[s_i, s_{i+1})$. Since the time points s_i divide a data stream into i.i.d. samples of data, we call them *change points*. We refer to the transition between concepts as *concept drift* or simply as *change*.

MLDS has mainly focused on developing incremental algorithms and techniques to make these algorithms adaptive to change. Figure 1.4 illustrates an incremental algorithm. Updates to the model occur after the arrival of each new observation. In the figure, the changing colors indicate the updated model. The most up-to-date model (m_3 in the figure) is the one used for predicting newly arriving observations, for which feedback has not yet been revealed. Since the model changes over time due to incremental updates, the deployed model remains *trainable*. This is fundamentally different from the traditional ML pipeline for batch data (Figure 1.3) that separates model training and deployment.

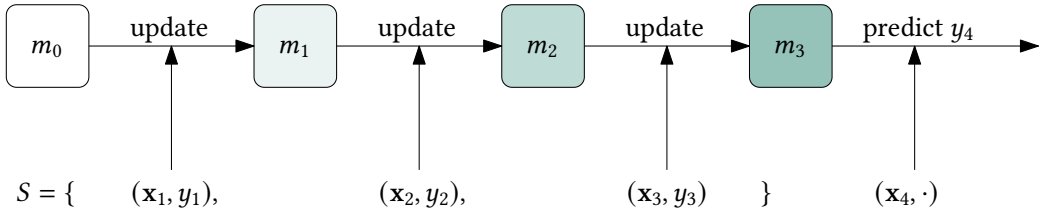


Figure 1.4.: Incremental learning in data streams

Existing algorithms for data streams include ones for classification and regression, such as incremental decision trees (Section 2.3) or ensembles [GRB19; Gom+17b], change detection algorithms (Section 2.4), clustering algorithms, sketches, outlier detection, and adaptive multi-armed bandit algorithms [Gom+17b; GRB19; Bif+18e; Gam10; FKB19]. These typically assume that feedback is cheap, plenty, and readily available. Those assumptions, however, are rarely met in practice, posing an additional challenge (C5) for the applicability of data stream algorithms [Gom+23].

1.1.4. Feedback in Decision-Making

Feedback plays a crucial role in decision-making as it determines the extent to which a decision-maker can learn from the environment and adjust future decisions. However, feedback varies along many dimensions, including availability, type, and cost. Figure 1.5 illustrates these key dimensions that we elaborate on in the following.

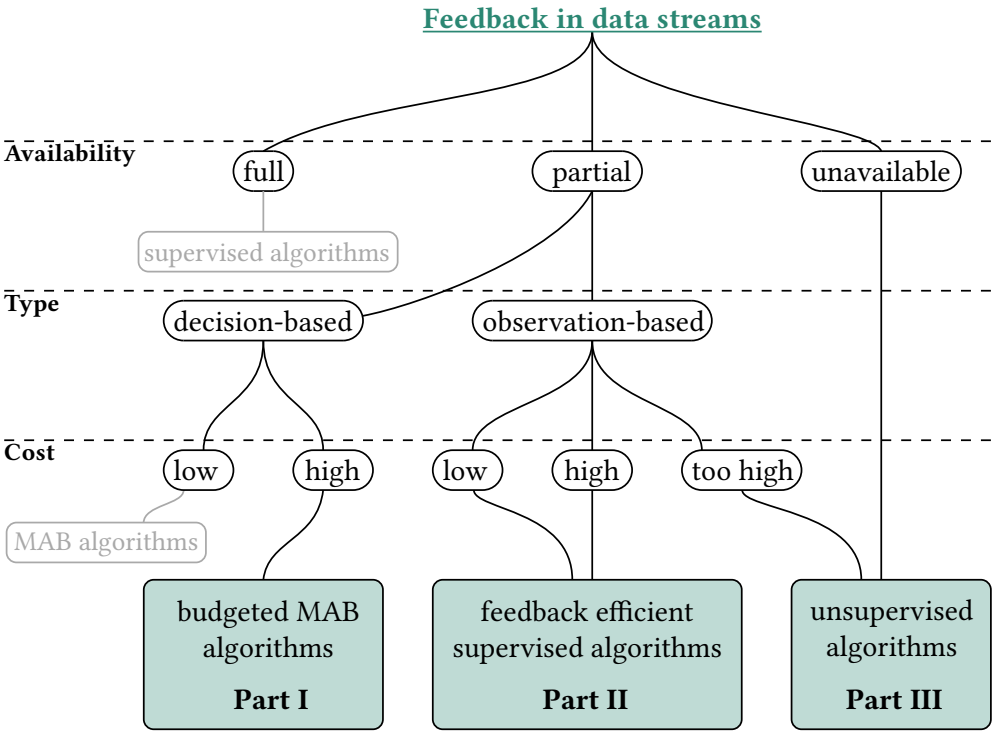


Figure 1.5.: Types of feedback in data streams

Availability. Supervised algorithms, e.g., for classification, assume *full feedback*—meaning that feedback becomes available for all observations and potential decisions. Consider, for instance, the fraud detection system discussed in Example 1.1: it predicts whether a transaction is valid or fraudulent. Ideally, the environment then provides feedback on whether that prediction was correct for each transaction. Since "valid" and "fraudulent" are mutually exclusive, the feedback about the true class implicitly confirms or rules out both possible choices. The full feedback setting is the most common and well-researched setting found in the data stream literature [Gom+23]. In certain applications, however, feedback may be entirely *unavailable*. For example, in industrial applications, including the chemical production plant in Example 1.3, the absence of historical data about faults or malfunctions is a common challenge, rendering supervised algorithms inapplicable [FCM22]. Hence, decision-makers often resort to unsupervised algorithms for process monitoring. The *partial feedback* setting represents a middle ground between the two previous settings. Here, we distinguish between observation-based and decision-based feedback.

Type. Figure 1.6 illustrates partial feedback types. *Observation-based* partial feedback occurs when feedback is only available for some observations. The previously mentioned payment provider, for instance, may receive thousands of transactions every second. In a non-ideal but more realistic scenario, feedback thus only becomes available for a small subset of transactions and with a delay. This motivates the use of feedback-efficient supervised algorithms and techniques such as semi-supervised and active learning [Gom+23]. *Decision-based* partial feedback is limited to the chosen action. An example of this type is found in online advertising (Example 1.2): the advertiser receives feedback only for the displayed ad, but not for the alternatives.

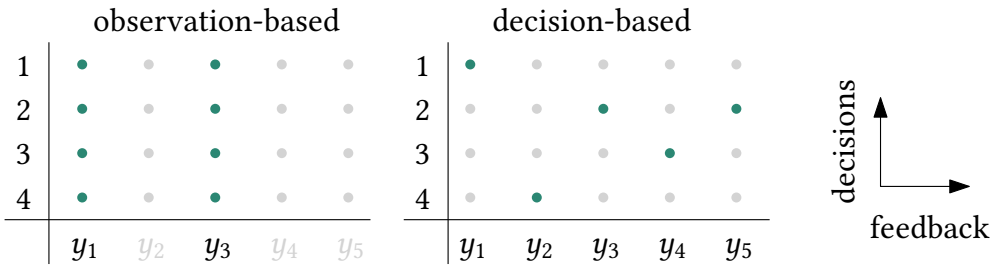


Figure 1.6.: Partial feedback types for a time horizon of 5 and four possible decisions

Cost. Feedback can also vary in cost. It may be inexpensive to obtain, e.g., when comparing weather forecasts with actual weather measurements, or it can be costly. In industrial manufacturing, for instance, obtaining feedback about faulty parts may require a temporary shutdown of the production plant. This motivates the development of feedback-efficient or unsupervised algorithms for process monitoring. Similarly, advertisers bid for favorable ad placements in online search results. The price they pay depends on their own bid and those of their competitors. Ads with better placement lead to more purchases, but are also more expensive. Hence, a decision-making algorithm should account for the costs associated with ad placements [Yua+12].

1.2. Research Gaps

While MLDS and online decision-making are active research fields, most existing algorithms are designed for full feedback or overlook the cost of obtaining it [Bif+18a; Gam10; Gom+23]. This renders them impractical in many real-world scenarios, including those mentioned in Section 1.1. To address this, we identify three research gaps related to limited and costly feedback, illustrated in Figure 1.5, and describe them in detail below.

1.2.1. Decision-Based Feedback

Motivation. Decision-based partial feedback occurs when a decision-maker receives feedback only for the chosen decision, but not for alternative decisions. Stochastic multi-armed bandits (MAB) are a common model of this scenario. Here, a player repeatedly chooses one of K arms (the decisions) and receives an arm-dependent reward (the feedback) drawn randomly from some unknown arm-dependent distribution. The goal is to maximize the cumulative reward by playing the arm with the highest expected reward as often as possible. The expected rewards are initially unknown, so the player must balance trying arms to learn their expected rewards (exploration) versus using the current information to play arms with known high expected rewards (exploitation).

Algorithms for the stochastic MAB setting do not account for the costs associated with different decisions. However, this is crucial in several real-world situations, including the selection of a cloud service provider [APP11], energy-efficient task scheduling for battery-powered embedded devices [Tra+12], bid optimization [Bor+07], or optimizing advertising on social media (Example 1.2). Figure 1.7 illustrates this so-called budgeted multi-armed bandit (BMAB) setting [Tra+10]. The player not only earns a reward for choosing an arm, but also has to pay a random cost and is restricted by a finite budget.

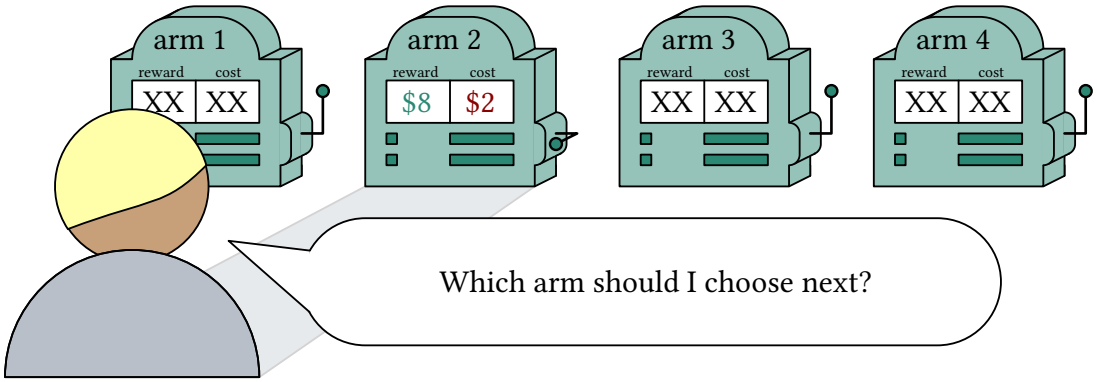


Figure 1.7.: A 4-armed budgeted multi-armed bandit

Challenges. Many existing budgeted multi-armed bandit (BMAB) algorithms rely on upper confidence bound (UCB) sampling [Xia+15a; Xia+16; Xia+17; Wat+17; Wat+18]. These algorithms balance exploration and exploitation by keeping track of a UCB of the

ratio between expected rewards and costs for each arm. At each time step, they play the arm with the highest UCB. However, we illustrate in Section 2.2 that existing UCB-algorithms fail to accurately model the ratio between expected rewards and costs. This inhibits their ability to balance exploration and exploitation effectively.

1.2.2. Full and Observation-Based Feedback

Motivation. Observation-based partial feedback occurs when the environment provides feedback for only a subset of observations. Even in the full feedback setting, feedback arrives incrementally and is particularly valuable during the early stages of execution. In both scenarios, algorithms must make efficient use of the available feedback to learn effectively. Classification algorithms for data streams are particularly well-suited in this case as they allow incremental model updates as soon as new feedback arrives. We summarize classification algorithms for data streams in Section 2.3. One algorithm class, decision trees, come with many benefits for decision-making: they are interpretable, can deal with numerical and categorical data, and are robust to outliers [Bre+84; Qui93]. They are also popular base models in ensembles [Gom+20]. A long-standing line of research has thus focused on developing tree-based algorithms that can learn incrementally from data streams. Hoeffding Trees (HT) and Extremly Fast Decision Trees (EFDT) are arguably the most well-known incremental decision trees [DH00; MWS18]. However, neither HT nor EFDT use feedback in a particularly efficient way, as we describe next.

Challenges. HT and EFDT keep track of the data and label distribution at the individual nodes. Both algorithms split a node once they have identified a good split with high probability. EFDT applies a more relaxed splitting mechanism than HT to improve feedback efficiency. The relaxed splitting mechanism, however, also leads to frequent suboptimal splits. EFDT thus revises its split decisions over time, which consists of pruning sub-optimal trees. This not only leads to sudden and significant accuracy drops (see Figure 1.8 for an illustration), but also discards a large portion of the already obtained feedback.

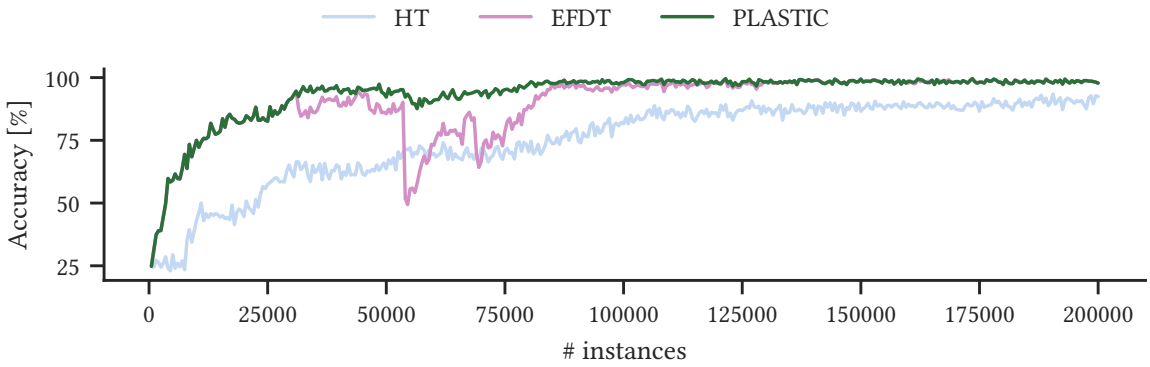


Figure 1.8.: Comparison of HT, EFDT, and PLASTIC (our contribution) using data generated by MOA's Random Tree Generator (10 binary attributes, 5 classes, default random seed).

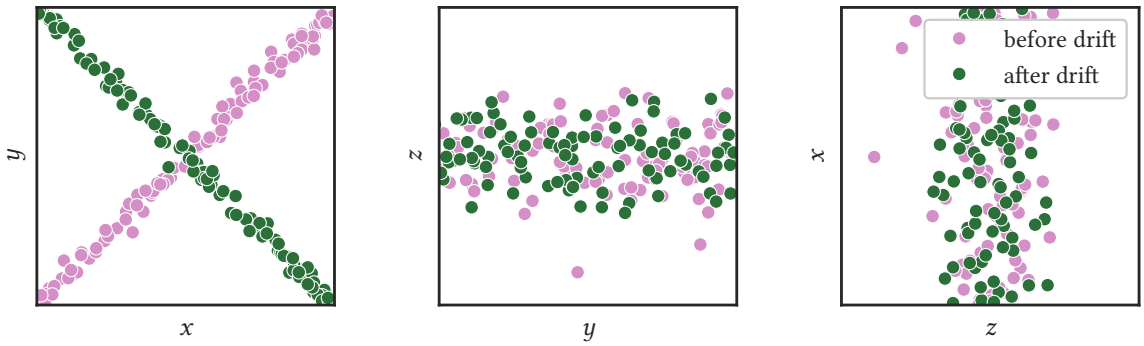
1.2.3. Generating Feedback

Motivation. When feedback is unavailable or too costly to obtain, a decision-maker may resort to unsupervised algorithms. These can provide the decision-maker with feedback about, say, clusters or outliers in the data, or whether the environment has changed. Unsupervised change detectors monitor the environment state $\mathbf{x}_t$ and detect changes in the underlying distribution (Section 1.1.3). These changes can indicate a change in the underlying process. However, simply knowing *that a change occurred* (R1) is insufficient when dealing with multivariate or even high-dimensional data. Instead, an algorithm should also provide feedback about the *dimensions that were affected* (R2) and *how strong* (R3) the change was in those dimensions. This allows for a targeted response in complex systems by generating feedback about which subsystem was affected and how severely.

Challenges. While prior works acknowledge the relevance of R1–R3 [Lu+19; Web+18], fulfilling them in combination remains challenging due to three interconnected reasons:

1. Changes may affect only a few dimensions or their correlation, as illustrated in Figure 1.9a. Here, the change is only visible in the combination of x and y, but not in their marginal distributions. Hence, monitoring dimensions separately is insufficient.
2. The z-dimension is independent from x and y (Figures 1.9b and 1.9c) and does not belong to the change subspace. Unaffected dimensions "dilute" a change, i.e., a change occurring in a subspace seems less severe in the full space. This might make changes harder to detect in all dimensions. Detecting the change subspace, however, should occur *after* detecting a change, as monitoring all possible subspaces is intractable.
3. Consequently, one should restrict computation of change severity to the change subspace to eliminate dilution.

We see that fulfilling R1–R3 should occur sequentially: first, an algorithm should detect the change, then identify the affected dimensions, and finally compute the change's severity in those dimensions. However, as highlighted in Section 2.4, none of the existing approaches fully achieves this yet.



(a) Projection 1: y over x **(b)** Projection 2: z over y **(c)** Projection 3: x over z

Figure 1.9.: Illustration of change subspace on 3-dimensional toy data.

1.3. Contributions

This dissertation addresses the previously motivated research gaps by presenting feedback-efficient algorithms designed for decision-making in data streams under limited feedback. Our contributions tackle three main key research questions:

- **Q1 – Decision-Based Partial Feedback:** How to optimize sequential decisions under budget constraints when feedback is costly and resources are limited?
- **Q2 – Observation-Based Partial Feedback:** How to improve efficiency in learning when feedback is available only for a subset of observations?
- **Q3 – Feedback Generation:** How to guide decision-making when no direct feedback from the environment is available?

To deal with **Q1**, we propose ω-UCB, a novel algorithm for the BMAB setting. Existing BMAB algorithms often favor the selection of arms with low average costs. However, these arms may also yield low rewards. ω-UCB employs asymmetric confidence intervals to improve the estimation of the ratio between expected rewards and costs over existing algorithms. We prove that the loss of our algorithm grows only logarithmically with the budget. Our experiments confirm ω-UCB's feedback efficiency under budget constraints that exceeds the previous state of the art. We published our results in [Hey+24b], created a summary video[1], and published an article showcasing the benefits of our algorithm[2].

To tackle **Q2**, we propose PLASTIC, an incremental, feedback-efficient decision tree. PLASTIC addresses EFDT's issues of sudden and unpredictable accuracy drops by restructuring the otherwise pruned subtree. The approach leverages a property we call *decision tree plasticity*—it allows restructuring a decision tree without affecting its predictions. Our experiments show that this adaptation enhances both average and worst-case accuracy of the previous state of the art. We published our results in [Hey+24c].

Finally, we address **Q3** by presenting the Adaptive Bernstein Change Detector (ABCD), an unsupervised change detector for high-dimensional data streams that relies on the principle of dimensionality reduction. This third part augments our two previous contributions in various aspects. First, ABCD is unsupervised and hence requires no feedback from the environment. The algorithm is thus applicable whenever the first two contributions are not. Second, ABCD identifies the change subspace and quantifies change severity, thereby providing the decision-maker with richer feedback than existing techniques. Last, one can couple ω-UCB and PLASTIC with ABCD to adapt them after a change was detected—a common approach for developing adaptive incremental algorithms [Bif+18c]. We published our results in [Hey+24a].

[1] youtu.be/tRQ1-DiXIME

[2] towardsdatascience.com/optimizing-marketing-campaigns-with-budgeted-multi-armed-bandits-a65fccd61878 (Published on Medium via Towards Data Science)

1.4. Prior Works

This dissertation builds upon the following publications:

- Marco Heyden et al. "Budgeted multi-armed bandits with asymmetric confidence intervals". In: *KDD '24: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Barcelona, Spain: ACM, 2024, pp. 1073–1084. ISBN: 9798400704901. DOI: 10.1145/3637528.3671833
- Marco Heyden et al. "Leveraging plasticity in incremental decision trees". In: *Machine Learning and Knowledge Discovery in Databases. Research Track. ECML PKDD 2024*. Vilnius, Lithuania: Springer Nature, 2024, pp. 38–54. ISBN: 978-3-031-70362-1. DOI: 10.1007/978-3-031-70362-1_3
- Marco Heyden et al. "Adaptive Bernstein change detector for high-dimensional data streams". In: *Data Mining and Knowledge Discovery* 38.3 (2024), pp. 1334–1363. DOI: 10.1007/S10618-023-00999-5

We have incorporated their content but revised it to create a cohesive monograph. While these prior works include the key contributions of this dissertation, the text has been adapted—extended, rephrased, restructured, or condensed—as needed. Algorithms, definitions, equations, figures, notation, theorems, lemmas, and tables may differ slightly from the original works due to adjustments for consistency. Chapter 2 builds upon the related work sections of the three publications. The main chapters of this dissertation (Chapters 3, 4, and 5) explicitly indicate the prior works they are based on. Other chapters may also incorporate content from these prior works.

1.5. Outline

The remainder of this dissertation is structured as follows.

Chapter 2 of Part I discusses related work.

Part II contains our contributions: Chapter 3 addresses **Q1**—we present our multi-armed bandit algorithm for decision-making under budget constraints. Chapter 4 addresses **Q2**—we describe our incremental decision tree algorithm that tackles the inefficiencies of existing approaches related to observation-based feedback. Chapter 5 addresses **Q3**—we introduce our approach for change detection in high-dimensional data streams that characterizes change according to when and in which subspace it occurred, and how strong it was.

Part III wraps up this dissertation: Chapter 6 presents our conclusions, and Chapter 7 outlines directions for future research.

Appendices A.1, A.2, and A.3 collect additional proofs, experiments, details, and illustrations of Chapters 3, 4, and 5, respectively.

2. Related Work

This chapter places the contributions of this dissertation in the context of related research and introduces key concepts and relevant algorithms. It starts with an overview of the related research fields in Section 2.1. Sections 2.2, 2.3, and 2.4 then describe foundational work and algorithms related to each specific contribution.

2.1. Related Research Areas

2.1.1. Continual Learning

The field of ML for data streams (MLDS) is closely related to continual learning (CL), as both deal with training models on continuously arriving data [KK21; Wan+24; Bif+18d]. Despite this similarity, CL and MLDS make distinct assumptions about the relevance of past data: CL assumes that past data remains relevant over time. Thus, the main goal is to train models that perform well across multiple sequential tasks, learning new tasks without catastrophically forgetting how to handle previous ones [Kir+17]. In contrast, MLDS assumes the latest data to be the most relevant. Consequently, the primary goal in MLDS is to adapt the ML model so that it performs well on the current data rather than retaining performance across all data.

Another notable difference is the focus on algorithm classes. CL research has largely concentrated on neural network (NN) models [Wan+24], whereas they remain under-explored in current MLDS literature [Bah+21]. One reason is that the iterative training process of NN conflicts with data stream requirements to process each instance only once and in near real-time [Bah+21; Bif+18a]. Additional challenges include the adaptability of NN models to concept drift and the balance between runtime efficiency, convergence speed, and predictive performance [Zho+23].

Due to the above differences, CL and MLDS have evolved primarily as separate research areas [Bif+18a; Wan+24]. Nevertheless, recent studies are beginning to address them both in combination [AP19; KK21; Gun+23]. Our contributions may prove valuable at the intersection of CL and MLDS if these fields continue to converge.

2.1.2. Time Series Analysis

MLDS is closely related to the field of time series analysis (TSA) as both areas deal with sequential data. However, MLDS focuses on real-time processing and analytics in a continuous, never-ending stream of data. In contrast, TSA aims at analyzing and understanding patterns in completed data sequences, such as financial or electrocardiogram data. Consequently, the data stream literature focuses on developing incremental and change-adaptive algorithms for tasks like classification, regression, clustering, outlier detection, change detection, and association rule mining [Bif+18a]. TSA literature, in contrast, focuses on tasks related to time-dependent patterns, such as forecasting, modeling, segmentation, as well as anomaly detection and trend analysis [BD16; SS17; Wei23].

Both areas also differ in the underlying assumption about neighboring data points: MLDS assumes data to be i.i.d. while TSA assumes that neighboring observations are correlated. A data stream is said to be stationary as long as the observations are i.i.d. samples from the same distribution. A time series is said to be stationary when its mean and autocorrelation remain constant over time [BD16].

While both research areas overlap—e.g., the fields of change detection in data streams and in time series—TSA algorithms often struggle with the continuous arrival of new data. We further elaborate on this in Section 2.4.

2.1.3. Semi-Supervised Learning

Semi-supervised learning (SSL) summarizes algorithms and techniques to improve model performance by learning from the combination of labeled and unlabeled data. That makes it promising for learning from observation-based partial feedback. SSL is well-studied in the static setting [EH20]. Existing techniques broadly belong to

1. wrapper methods, such as self-training and co-training, which are compatible with supervised algorithms,
2. unsupervised preprocessing, such as feature extraction and clustering, and
3. intrinsically semi-supervised algorithms which modify the learning objective or optimization procedure to account for unlabeled data.

However, their application in data streams introduces additional challenges, such as computational efficiency and concept drift. For instance, many wrapper methods and clustering algorithms require multiple passes over the data [Gom+23]. This is impractical for data streams which often involve high data arrival rates and memory constraints. Even in the batch setting, applying SSL can degrade performance when the data violates the assumptions of the SSL technique [SNZ08; LZ15; Oli+18]. Concept drift, which causes the data to change over time, complicates this even further.

Despite these challenges, SSL for data streams is an active field of research [Gom+23]. However, this dissertation does not directly explore SSL for data streams, but focuses

instead on resolving feedback inefficiencies in two widely-used algorithms for the full-feedback setting (Section 2.3.2). By resolving these inefficiencies, we aim to lay a foundation for future SSL research, as discussed in Chapter 7.

2.1.4. Reinforcement Learning

Reinforcement learning (RL) is one of the most general (and arguably most well-known) frameworks for sequential decision-making. In RL, a decision-maker observes the state of the environment, makes a decision, and receives a reward. This decision may also affect the environment, potentially leading to a new state. The goal is to learn sequences of decisions that maximize (some function of) long-term cumulative rewards. A common assumption in RL is the Markov property: the next state $\mathbf{x}_{t+1}$ depends only on the current state $\mathbf{x}_t$ and the decision k_t. For example, in chess, the new positions on the board depend solely on the current configuration and the next move, while prior states and moves are irrelevant. For a comprehensive overview of RL, we refer the reader to [SB18].

While RL is well-suited to scenarios where decisions influence the environment, many real-world applications do not have this property. For example, in fraud detection, labeling a transaction as fraudulent does not affect subsequent transactions. In such cases, observations at successive time steps can be assumed to be independent. This dissertation focuses on these scenarios, with the goal of developing decision support systems that generate reliable surrogate feedback at any given time. Yet, RL is closely related to one of our main contributions: a budgeted multi-armed bandit algorithm. In the next section, we introduce the multi-armed bandit (MAB) framework.

2.2. Decision-Based Feedback

2.2.1. Foundation

Figure 2.1 illustrates three popular frameworks for learning from decision-based partial feedback, RL, contextual multi-armed bandits (CB) and traditional MAB.

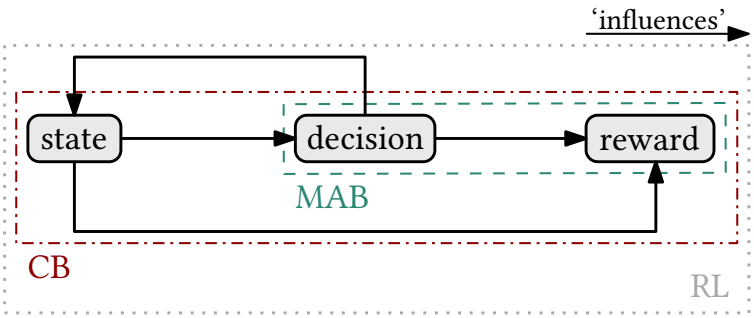


Figure 2.1.: The reinforcement learning cycle

In RL, which we described previously, decisions can influence the state of the environment, as indicated by the backward arrow in Figure 2.1.

In CB, the state and decision at time t have no effect on the subsequent state at time $t + 1$. That is, the state represents *external* factors that contain information about the connection between the decisions and the rewards. For example, when generating movie recommendations, a streaming platform can leverage context information about a user (the state), such as age or gender, to improve the recommendations (the decision). However, the recommended movies have obviously no effect on a user's age or gender. The goal is to learn which decisions to take in which context. Traditional MAB simplify the decision-making process even further by not considering the environment's state. Instead, the goal is to balance exploring different decisions to learn about their reward and exploiting the collected knowledge to choose the decisions that maximize the reward.

Many different MAB variants exist. For instance, one can distinguish between MAB with a finite or infinite number of arms. One can also differentiate between *stochastic* and *adversarial* MAB. In the former, the rewards of each arm are drawn i.i.d. from some unknown, arm-dependent distribution. Adversarial MAB relax this assumption and allow arbitrary reward sequences, even those chosen by an adversary. We refer to [LS20] for a summary of the common settings and algorithms. Last, one can distinguish between MAB restricted by a time horizon and ones restricted by a *budget*. The latter are suitable whenever decision-making is costly, and the costs are unknown in advance. Our first contribution, ω-UCB (Chapter 3), belongs to this category. In particular, we consider the stochastic budgeted MAB (BMAB) setting, in which rewards and costs are i.i.d. samples from unknown distributions. The next section discusses related work in this field.

2.2.2. Algorithms

Tran-Thanh et al. [Tra+10] introduced the BMAB problem and proposed ε-first algorithms in which exploration and exploitation phases are separate from each other. Subsequent algorithms KUBE [Tra+12] and PD-BwK [BKS13] propose to formulate the problem as bandits with knapsacks (BwK), where the size of the knapsack represents the available budget. Both algorithms require knowledge of the budget B. This restricts their applicability when B is an unknown quantity. However, we argue that such a restriction is unnecessary since the advantage of exploiting knowledge about B becomes negligible for sufficiently large budgets (cf. Section 3.2). Another approach, UCB-BV1 [Din+13], addresses the special case of discrete random costs. Later solutions [Xia+15a; Xia+15b; Xia+17; Wat+17; Wat+18] adapted concepts from traditional MAB, such as upper confidence bound (UCB) or Thompson sampling [ACF02; Tho33; Tho35]. These algorithms, illustrated on a high level in Algorithm 2.1, can deal with continuous random costs and unknown budgets.

UCB sampling balances exploration and exploitation by keeping track of a UCB for the ratio between expected rewards and costs and playing the arm with the highest UCB. If an arm was played only a few times, uncertainty about its expected rewards and costs is high. This leads to a high UCB and increased exploration. With more plays, uncertainty

decreases, and the UCB approaches the true value of the arm. This reduces exploration and promotes the exploitation of arms with consistently high UCBs.

Budgeted Thompson sampling (BTS) balances exploration and exploitation by maintaining distributions over the ratio of expected rewards to costs for each arm. BTS models these ratios using Beta distributions, where the parameters correspond to the sums of observed rewards and costs. At each step, BTS samples from these Beta distributions and selects the arm with the highest ratio of the samples. As an arm gets played more often, the probability mass of the respective Beta distributions concentrates around the expected rewards and costs. This gradually leads to the exploitation of arms with a high reward-to-cost ratio.

Algorithm 2.1 UCB and Thompson sampling for BMAB

Require: The available budget B and the number of arms K

procedure UCB SAMPLING
  UCB $\leftarrow [\infty]^K$ ▷ A vector with reward-cost UCB for each arm
  $t \leftarrow 0$
  while $B > 0$ **do**
    $t \leftarrow t + 1$
    $k_t \leftarrow \arg\max_{k \in [K]}$ UCB
    Play arm k_t
    Observe reward r_t and cost c_t
    Update UCB based on k_t, r_t, c_t, and t
    $B \leftarrow B - c_t$
procedure BTS
  $R \leftarrow [1]^K$ ▷ A vector with sum of rewards for each arm
  $C \leftarrow [1]^K$ ▷ A vector with sum of costs for each arm
  $t \leftarrow 1$
  while $B > 0$ **do**
    Sample $\tilde{r}_k \sim \text{Beta}(R_k, t - R_k)\ \forall\ k \in [K]$
    Sample $\tilde{c}_k \sim \text{Beta}(C_k, t - C_k)\ \forall\ k \in [K]$
    $k_t \leftarrow \arg\max_{k \in [K]} \tilde{r}_k / \tilde{c}_k$
    Play arm k_t
    Observe reward r_t and cost c_t
    $B \leftarrow B - c_t$
    $t \leftarrow t + 1$
    if r_t or c_t are continuous **then** ▷ Transform r_t, c_t to Bernoulli samples
      $r_t \sim \text{Bernoulli}(r_t)$
      $c_t \sim \text{Bernoulli}(c_t)$
    $R_{k_t} \leftarrow R_{k_t} + r_t$
    $C_{k_t} \leftarrow C_{k_t} + c_t$

Some UCB algorithms [Wat+17; Wat+18; Xia+17; CES20] compute the bound from the ratio of the average reward and average cost plus some uncertainty-related term, cf. Eq. (2.1). We call this type *united* (u). Other algorithms [Xia+17; BKS13; Xia+16] divide the reward's UCB by the cost's lower confidence bound (LCB), cf. Eq. (2.2). We refer to this type as

composite (c). There also are *hybrid* (h) algorithms [Xia+17; Xia+15a] that combine both types.

$$UCB_u = \frac{average\ reward}{average\ cost} + uncertainty \tag{2.1}$$

$$UCB_c = \frac{average\ reward + uncertainty}{average\ cost - uncertainty} \tag{2.2}$$

However, most current algorithms have at least one of the following issues:

(i1) **Over-optimism:** The algorithm often computes UCB that are too tight.

(i2) **Over-pessimism:** The algorithm often computes UCB that are too loose.

(i3) **Invalid values:** Negative or undefined UCB occur if the cost's lower confidence bound in Eq. (2.2) becomes negative or zero.

The latter can happen, for instance, when computing the lower confidence bound of an arm's expected cost with Hoeffding's inequality [Xia+17; Xia+15a].

Figure 2.2 illustrates issues (i1) and (i2). We randomly parameterized 10 000 Bernoulli reward and cost distributions and sampled from them. We used these samples to compute 99 % confidence intervals of the reward-cost ratio using several state-of-the-art UCB sampling algorithms. The left plot of Figure 2.2 shows the share of cases in which the expected reward-cost ratio exceeds (i.e., violates) its UCB. Values above 1 % indicate overly tight bounds. The right plot shows the UCB of the reward-cost ratio divided by its expectation, with higher values indicating looser bounds. Existing united algorithms (u) tend to suffer from issue (i1), hybrid approaches suffer from either of both issues, and issue (i2) mainly affects composite approaches. An exception is UCB-B2 [CES20], who's UCB is both tight and reliable (although not as tight as that of ω-UCB).

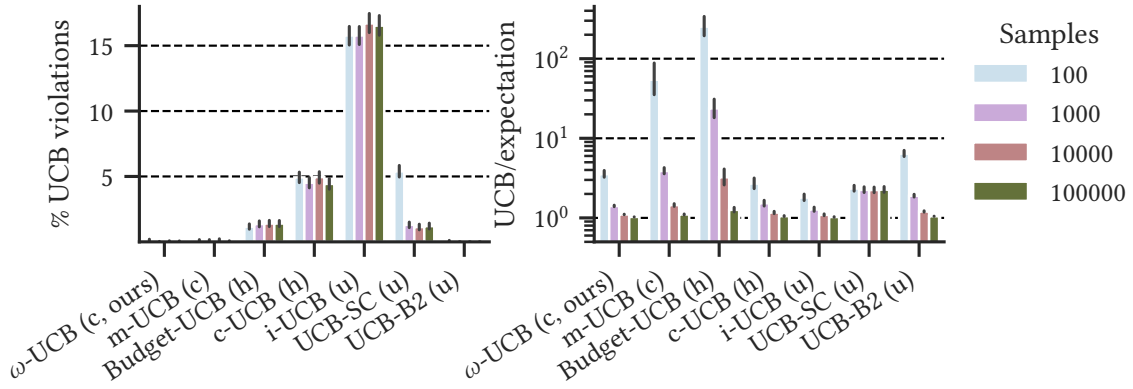


Figure 2.2.: Illustration of over-optimism (i1) and over-pessimism (i2)

To address (i1) and (i2), some approaches provide a hyperparameter for adjusting the confidence interval manually [Xia+17]. However, setting such a hyperparameter is difficult

as it depends on the unknown mean and variance of the rewards and costs. To address (i3), current approaches set the UCB of the ratio to infinity [Xia+17] or the cost LCB to a small positive value [Xia+15a]. These heuristic solutions largely ignore the information already acquired about the cost distribution and tend to cause either (i1) or (i2).

The one algorithm based on Thompson sampling, BTS [Xia+15b], requires transforming continuous rewards and costs into Bernoulli samples. As a result, the algorithm disregards information about the variance of rewards and costs, causing over-pessimism (i2) when the variance of the reward or cost distribution is small. MRCB [Xia+15a] deals with the challenge of playing multiple arms in each time step; when playing only one arm at a time, the algorithm becomes m-UCB [Xia+17] that is similar to our algorithm. However, m-UCB relies on Hoeffding's inequality, which does not take into account the distance between a random variable's sample mean and boundaries. To see why this is problematic, consider the following example[1]:

Example 2.1 (Arm-selection bias in m-UCB)**.** Assume two arms with unknown expected rewards and costs of $\mu_1^r = 0.8, \mu_1^c = 0.2, \mu_2^r = 0.1, \mu_2^c = 0.1$. Clearly, arm 1 offers a higher ratio between rewards and costs and should thus be preferred. However, m-UCB is biased towards pulling arm 2, due to the arm's lower average costs. For instance, if $t = 10\,000$, and both arms were played $n_1 = n_2 = 1000$ times, using m-UCB with $\alpha = 1$ would yield reward-cost UCB values of ≈ 2.95 for arm 1 and ≈ 48.6 for arm 2 due to the high influence of the denominator in Eq. (2.2). m-UCB would hence pull arm 2. In comparison, ω-UCB would compute values of ≈ 5.5 and ≈ 2.1, and play arm 1.

More recent algorithms are adaptations of existing ones to specific scenarios. For example, [Ava+21] develop a multi-platform algorithm for Bandits with Knapsacks, and [DJG22] extend BwK to the combinatorial setting in which the algorithm can pull one or more arms in each round. However, the authors assume that an arm's cost is a known, fixed quantity while we address the challenge of dealing with cost distributions. [CES19] proposes a novel 'bandits with interruptions' framework in which a player can interrupt a taken action to reduce the cost. [CES20] extends BMAB to handle unbounded cost and reward distributions and presents algorithms for various settings. Algorithm UCB-B2 is tailored for our setting of bounded rewards and costs, and relies on an empirical version of Bernstein's inequality. However, using this bound does not resolve the bias illustrated in Example 2.1.

The above algorithms either have issues (i1)–(i3) [Xia+17; Xia+15a; Wat+17; Wat+18; Xia+15b; CES20], are not designed for continuous random costs [Tra+12; Din+13; Xia+15b], require knowledge of B [Tra+12; BKS13], or have been shown to perform inferior to others [Tra+10; Din+13; Tra+12]. Figure 2.3 provides a compilation of existing head-to-head empirical comparisons between various algorithms. Upwards-pointing triangles indicate that the algorithm in the corresponding row outperformed the algorithm in the corresponding column in the respective paper, while downward-pointing triangles indicate the opposite. Circles represent cases where both algorithms performed similarly,

[1] One can construct analogous examples for other symmetric bounds, such as Bernstein's inequality.

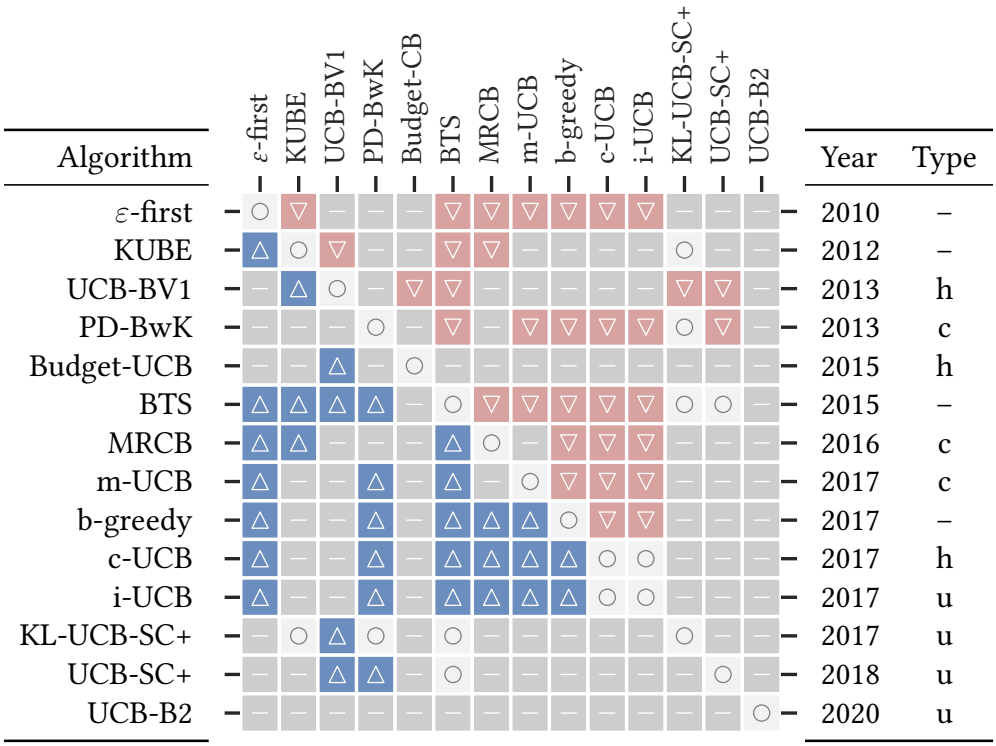


Figure 2.3.: Empirical performance of BMAB algorithms according to related work

while horizontal lines indicate that the algorithms have not been compared. One can see that KUBE [Tra+12] outperforms ε-first [Tra+10], while UCB-BV1 [Din+13] and BTS [Xia+15a] outperform KUBE. UCB-BV1 is inferior to more recent algorithms [Xia+15b; Xia+15a; Wat+17; Wat+18]. BTS [Xia+15b], b-greedy [Xia+17], and {i, c, m}-UCB [Xia+17] outperform PD-BwK [BKS13].

2.3. Full and Observation-Based Feedback

2.3.1. Foundation

Learning from full or observation-based feedback includes algorithms for classification and regression. We focus on the former, since PLASTIC [Hey+24c], our second contribution, belongs to this category. For the latter, we refer the reader to [Bah+21; Gom+20; Bif+18f].

Bahri et al. [Bah+21] divide data stream classification algorithms into five categories: frequency-based, neighborhood-based, neural-based algorithms, ensembles, and tree-based algorithms (Figure 2.4). Examples of frequency-based techniques include the Naive Bayes (NB) classifier [FGG97] and more resource-efficient adaptations for data streams [BMB18]. NB allows for incremental updates as it only requires updating counters of attribute values and labels. Neighborhood-based algorithms include versions of k-nearest neighbor (kNN) that address computational efficiency [Bah+20] or adaptability to concept drift [LHW16].

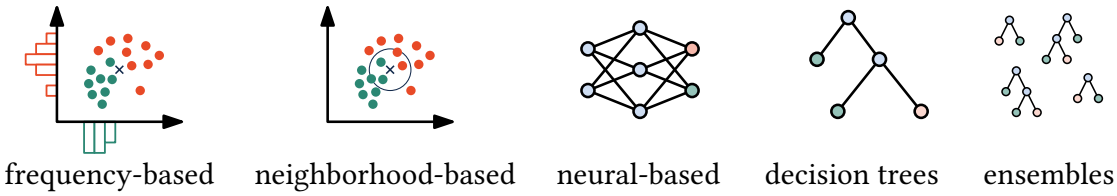


Figure 2.4.: Data stream classification algorithms

NN-based algorithms have so far received limited research attention [Bah+21; Gun+22]. [AP19] introduced autonomous deep learning (ADL), a self-constructing NN that dynamically adjusts its width and depth in response to under-fitting, over-fitting, and concept drift. Building on this concept, later works extended self-constructing mechanisms to autoencoders and recurrent NN [Das+19; Ash+20]. However, a recent study indicates that ADL is less competitive than ensemble methods, which often rely on decision trees [Gun+22]. To address this, the authors proposed continuously adaptive neural networks for data streams (CAND), which maintain a pool of models with varying hyperparameter configurations and dynamically select the best-performing model for the current data. Most recently, [SGW24] proposed elastic online deep learning (EODL), a NN with adjustable depth that adapts to concept drift using intermediate classifiers attached to its hidden layers. Experimental results suggest that EODL outperforms CAND.

Ensemble methods, which combine the predictions of multiple base models, are particularly well suited for data streams and have thus received considerable research attention [Gom+17a; Kra+17; Bah+21]. In comparison to single models, ensembles can adapt to concept drift by resetting, removing, or reweighting the poorly performing base models. This further allows the use of base models that do not implement concept drift adaptation mechanisms by default. Last, training the base learners can often occur in parallel, reducing the computational overhead compared to a single model. Several ensemble techniques for data streams exist. These often adopt concepts from traditional ensembles, such as bagging and boosting [Bre96; FS97; Oza05; Gun+24], random forests [ASM07; Gom+17b], or random patches [LG12; GRB19].

Tree-based algorithms for data streams are the most popular base models for ensembles due to (1) their instability (which increases ensemble diversity), (2) their ability to learn incrementally from one instance at a time, and (3) their ability to handle both numerical and categorical data [Gom+17a]. But they are also useful as single models due to their interpretability. Our second contribution, PLASTIC, is an incremental decision tree. Section 2.3.2 thus summarizes tree-based algorithms in more detail.

2.3.2. Algorithms

A widely adopted decision tree algorithm for data streams is the Hoeffding Tree (HT) algorithm [DH00]. HT, outlined in Algorithm 2.2, builds a decision tree incrementally: At each leaf, HT keeps track of counters n_{ijk}, i.e., one counter for each attribute (a_i), value

(v_j), class (y_k) combination.[2] Based on the counters, HT can make predictions and compute the information gain IG for each possible split. The algorithm splits a leaf once the IG difference between the best and second-best possible split exceeds a threshold

$$\epsilon_{\text{split}} = \sqrt{\frac{R^2 \ln(1/\alpha)}{2n}}.$$

We observe that ϵ_{split} increases as α decreases and decreases as n increases. This implies that smaller values of α require the algorithm to collect more observations before splitting a leaf, as it seeks splits of higher confidence. Conversely, choosing a larger α promotes tree growth by reducing the number of observations required, though at the expense of making less confident splits.

Extremly Fast Decision Tree (EFDT) [MWS18], which is particularly related to our approach, improves HT's statistical efficiency by relaxing the condition under which they split a leaf. Compare Line 6 in Algorithm 2.2 to Line 5 in Algorithm 2.3: In comparison to HT, EFDT already splits a leaf if the IG of any attribute is significantly larger than 0. Thus, EFDT requires less data to build a tree, leading to better predictions after fewer data points. The downside is that such eager splitting leads to many suboptimal splits. Thus, EFDT revises its former split decisions and re-splits a node if the split turns out to be suboptimal. In the process, the subtree below that node is lost, causing sudden and unexpected accuracy drops, illustrated in Figure 1.8.

Algorithm 2.2 Hoeffding Tree (HT)

```
1: procedure Leaf Node
2:     for each new instance x_i arriving at leaf l do
3:         Use x_i to update counters in l
4:         a_1 ← attribute with the highest IG
5:         a_2 ← attribute with the 2nd-highest IG
6:         if IG(a_1) > IG(a_2) with probability > 1 − α then
7:             Split l at a_1
```

Over time, HT has been subject to further developments focusing on vague data [HY09; DMP21], stability [PHK07], its statistical foundation [Rut+13; Rut+14a; Rut+14b; Rut+15], and prediction quality. Other research has addressed resource consumption [Bif+10a], and implemented adaptivity to concept drift [HSD01; GFR06; BG09; WLH12; MSW22]. Regarding prediction quality, [GRM03] proposes to use classifiers, e.g., Naive Bayes, at the leaves of a tree. This approach is widely applicable, for example in HT, EFDT, and our approach. To achieve change adaptivity, the semi-supervised SUN algorithm creates concept clusters at the leaves of an incremental decision tree and detects deviations between older clusters and new ones [WLH12]. Extending our algorithm in a similar fashion could be interesting future work. Other algorithms monitor accuracy at the nodes

[2] In the case of numerical attributes one typically relies on discretization, e.g., histograms, to maintain memory efficiency. Refer to [Bif+18b] for additional information.

Algorithm 2.3 Extremly Fast Decision Tree (EFDT)

procedure LEAF NODE
  for each new instance $\mathbf{x}_i$ arriving at leaf node l **do**
    Use $\mathbf{x}_i$ to update counters in l
    $a_1 \leftarrow$ attribute with the highest IG
    if $IG(a_1) > 0$ with probability $> 1 - \alpha$ **then** ▷ Relaxed splitting criterion
      Split l at a_1
procedure INTERNAL NODE
  for each new instance $\mathbf{x}_i$ arriving at internal node s **do**
    Use $\mathbf{x}_i$ to update counters in s
    $a_1 \leftarrow$ attribute with the highest IG
    $a_c \leftarrow$ current split attribute
    if $IG(a_1) > IG(a_c)$ with probability $> 1 - \alpha$ **then** ▷ Split revision
      Remove subtree below s
      Split s at a_1

in the tree [BG09] and start building a new subtree in the background when accuracy decreases. This 'background tree' replaces the current subtree once it is more accurate. The most recent Extremely Fast Hoeffding Adaptive Tree (EFHAT) [MSW22] does this and combines it with EFDT's relaxed initial splitting mechanism. EFHAT is the state of the art in adaptive incremental decision trees.

Some early ML research, ID5R [Utg89] and ITI [UBC97], has used subtree restructuring to build decision trees incrementally. However, these algorithms are unsuitable for data streams: (1) The algorithms are intended to build a tree from some given database. Hence, the approaches assume data to follow the same distribution and be finite in volume. (2) Both approaches are slow due to greedy tree revision (ID5R) and recursive restructuring of subtrees (ITI). (3) They also consume large amounts of memory, as every leaf has to store all its data points to facilitate future restructuring. PLASTIC, in contrast, relies on statistical testing to decide whether restructuring is beneficial and only stores the counters required for future splits.

2.4. Generating Feedback

2.4.1. Foundation

When feedback from the environment is unavailable, unsupervised algorithms can generate surrogate feedback to guide decision-making. These algorithms fall into several categories, including clustering [ZA21], frequent pattern mining [LJA14], outlier detection [SOB22], and change detection (Figure 2.5). Our third contribution, ABCD (Chapter 5), focuses on the latter.

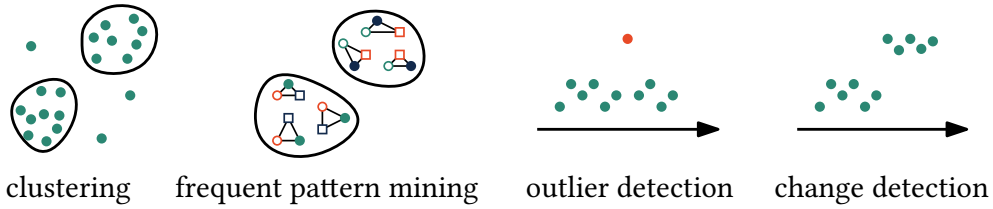


Figure 2.5.: Unsupervised algorithms that generate feedback

While outlier detection identifies individual or brief bursts of observations that deviate from the current data concept, change detection evaluates whether the underlying concept of a data stream has shifted persistently. Change detection algorithms differ in several key aspects, including:

- the types of time windows they analyze (e.g., fixed, sliding, or adaptive),
- their suitability for univariate, multivariate, or high-dimensional data, and
- whether they operate online (in data streams) or offline (in static data sequences).

We now briefly discuss these aspects.

Window types. Change detectors identify changes whenever a measure of discrepancy between newer observations (the current window) and older observations (the reference window) exceeds a threshold. Some approaches, e.g., D3 [Göz+19] or PCA-CD [Qah+15], implement the reference and current window as two contiguous sliding windows (Figure 2.6, center). Other approaches, such as IBDD [SCM20], IKS [Rei+16], and WATCH [Fab+21] use a fixed reference window (Figure 2.6, left). A major problem is to choose the appropriate size for the window; thus [BG07] propose windows of adaptive size, which grow while the stream remains unchanged and shrink otherwise (Figure 2.6, right). Several works leverage this principle, e.g., [Sun+16; Kha+15; FKB19; Sur+22; KFH24] and our approach.

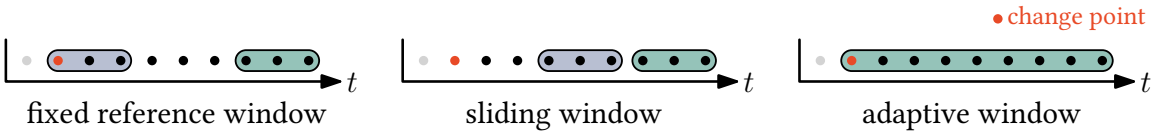


Figure 2.6.: Window types

Change detection in time series. Change detection in time series, also known as offline change point detection or signal segmentation, divides a time series of a given length into K homogeneous segments [TOV20]. Many of the respective algorithms are not suitable for data streams: Some require specifying K a priori [BP03; HC07; LLC15], others scale superlinearly with time [KFE12; LBA14; MJ14; Cha+17; GA18]. WATCH [Fab+21], discussed in more detail in Section 2.4.2, is the state-of-the-art extension of offline change point detection to data streams.

Pattern-based change detection. A related research field, pattern-based change detection, deals with identifying changes in temporal graphs [Log+18; ICC19; Imp+20a; Imp+20b]. In particular, [Log+18] detects changes in the graph, identifies the affected subgraphs, and quantifies the amount of change for these subgraphs. This is similar to our methodology. However, these methods work well with graph data, while we consider vector data. To apply these methods in our context, one would need to create a graph, e.g., by representing each dimension as a node and indicating pairwise correlations with edges. However, constructing such a graph scales poorly to high-dimensional observations and leaves changes in the correlation between more than two dimensions unnoticed.

Univariate and multivariate change detection. Univariate change detectors aim to find changes in windows of univariate data points. They are typically used in conjunction with other models, such as classifiers, to detect whether their performance decreases [Lu+19]. Hence, they are best suited for scenarios in which feedback is abundant. Two univariate change detectors, Adaptive Windowing (ADWIN) [BG07] and SeqDrift2 [PSK14], share some similarities with our approach, ABCD. Like ADWIN, our approach relies on an adaptive window. Like SeqDrift2, it uses Bernstein's inequality [Ber24]. But unlike ADWIN and SeqDrift2, ABCD can handle multivariate data. Change detectors for *multivariate* and *high-dimensional* data can be used to detect changes in the environment state $\mathbf{x}_t$. ABCD belongs to this category, so we focus our review on these types of change detection algorithms.

2.4.2. Algorithms

To detect changes in multivariate (MV) data, some approaches apply univariate algorithms in each dimension of the stream. [FDK19] propose to use one ADWIN detector per dimension. They declare a change whenever a certain fraction of the detectors agree. We call this approach AdwinK later on. Similarly, IKS [Rei+16] uses an incremental variant of the Kolmogorov-Smirnov test deployed in each dimension. Unlike AdwinK, IKS issues an alarm if at least one dimension changes.

There also exist approaches specifically designed for multivariate [JRA20; Cec+20; Qah+15; Göz+19; Das+06], or even high-dimensional (HD) data [Fab+21; SCM20]. Similar to ABCD, [JRA20] and [Cec+20] use dimensionality-reduction methods to capture the relationships between dimensions. However, our approach is computationally more efficient, limits the probability of false alarms, identifies the change subspace, and estimates change severity. D3 [Göz+19] uses the AUC-ROC score of a discriminative classifier that tries to distinguish the data in two sliding windows. It reports a change if the AUC-ROC score exceeds a pre-defined threshold. PCA-CD [Qah+15] first maps observations in two windows to fewer dimensions using PCA. Then, the approach estimates the Kullback–Leibler (KL) divergence between both windows for each principal component. PCA-CD detects a change if the maximum observed KL divergence exceeds a threshold. However, principal component analysis (PCA) can only represent linear relationships and [GW19] points out that relying

Table 2.1.: Change detection algorithms

Approach	Reference	Type	R1	R2	R3
ADWIN	[BG07]	UV	✓	–	–
SeqDrift2	[PSK14]	UV	✓	–	–
kdq-Tree	[Das+06]	MV	✓	–	✓
PCA-CD	[Qah+15]	MV	✓	–	✓
IKS	[Rei+16]	MV	✓	✓	–
LDD-DSDA	[Liu+17]	MV	✓	–	–
AdwinK	[FDK19]	MV	✓	✓	–
D3	[Göz+19]	MV	✓	–	✓
ECHAD	[Cec+20]	MV	✓	–	✓
IBDD	[SCM20]	HD	✓	–	✓
WATCH	[Fab+21]	HD	✓	–	✓
ABCD (ours)	[Hey+24a]	HD	✓	✓	✓

on the maximum divergence ignores combined drift in multiple dimensions. LDD-DSDA [Liu+17] measures the degree of local drift that describes regional density changes in the input data. The approach proposed by [Das+06] structures observations from two windows (sliding or fixed) in a kdq-tree. For each node, they measure the KL divergence between observations from both windows. However, [Qah+15] shows experimentally that this approach is not suitable for high-dimensional data.

IBDD [SCM20] and WATCH [Fab+21] specifically address challenges arising from high-dimensional data. The former monitors the mean squared deviation between two equally sized windows. The latter monitors the Wasserstein distance between a reference and a sliding window. However, both can neither detect change subspaces nor measure change severity.

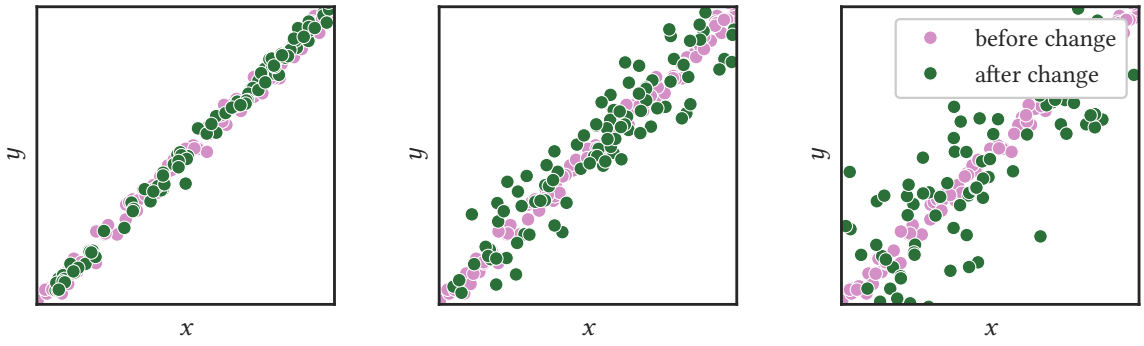
Change subspace. The notion of a *change subspace* is different from the existing notion of a *change region* [Lu+19]. The former describes a subset of dimensions that changed, and the latter identifies density changes in some local region, e.g., a hyper-rectangle or cluster [Liu+17]. Our definition of change subspaces is related to *marginal change magnitude* [Web+18], but is more general as it also accounts for changes in a subspace's joint distribution. Because high-dimensional spaces are typically sparse, identifying density changes in them is not effective. On the other hand, knowing that a change affected a specific set of dimensions can help identify the cause of the change, e.g., the part of the chemical plant from Example 1.3 that shows irregularities. Thus, we focus on detecting change subspaces in ABCD.

In the domain of statistical process control, some approaches extend well-known methods, such as Cusum [Pag54] or Shewhart charts [She31], to multiple dimensions. They address the problem of identifying change subspaces to some extent. However, they often make

unrealistic assumptions: they focus on Gaussian or sub-Gaussian data [CFT21; XXM20], require that dimensions are initially independent [CFT21], require subspace changes to be of low rank [XXM20], and assume that the change subspace size is known a priori [JCG18].

From the approaches for multivariate data, only AdwinK and IKS identify the corresponding change subspace. However, both approaches fail to find changes that hide in subspaces, e.g., correlation changes, because they monitor each dimension in isolation. In contrast, our approach aims to learn the relationships between different dimensions so that it can detect such changes. Next, since AdwinK only detects changes that affect more than a certain number of dimensions, it is incapable of identifying changes in smaller subspaces.

Change severity. According to [Lu+19], change severity is a positive measure of the discrepancy between the data observed before and after the change. Figure 2.7 illustrates this concept. To quantify change severity, one can either measure the divergence between distributions directly, as done by kdq-Tree [Das+06], LDD-DSDA [Liu+17], and WATCH [Fab+21], or indirectly with a score that correlates with change severity, as done by D3 [Göz+19]. Following this reasoning, an approach that satisfies R3 (cf. Section 1.2.3) should compute a score that depends on the change severity [Göz+19; Das+06; SCM20; Qah+15; Fab+21], i.e., the higher the score, the higher the severity. Finally, hypothesis-testing-based approaches, such as ADWIN [BG07], SeqDrift2 [PSK14], AdwinK [FDK19], or IKS [Rei+16], do not quantify change severity: a slight change observed over a longer time can lead to the same p-value as a severe change observed over a shorter time, hence p-values are not informative about change severity.



(a) No change **(b)** Change of medium severity **(c)** Change of high severity

Figure 2.7.: Illustration of change severity on 2-dimensional toy data.

Part II.

Contributions

3. Sequential Decision-Making under Budget Constraints

The content of this chapter is based on the following publication.

- Marco Heyden et al. "Budgeted multi-armed bandits with asymmetric confidence intervals". In: *KDD '24: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Barcelona, Spain: ACM, 2024, pp. 1073–1084. ISBN: 9798400704901. DOI: 10.1145/3637528.3671833

3.1. Chapter Overview

This chapter presents ω-UCB, an upper confidence bound (UCB) sampling algorithm for budgeted multi-armed bandits (BMAB). A decision maker ('player') faces the challenge of repeatedly choosing one of K possible decisions ('arms') with unknown expected rewards and costs. The goal is to maximize the total reward for a given budget. Hence, the player seeks to choose the arm with the highest reward-cost ratio as often as possible.

We derive asymmetric confidence intervals for bounded random variables. These intervals have the same range as the random variable and scale with the distance between the sample mean and the boundaries, yielding a more accurate and tight estimation of the reward-cost ratio than our competitors. Our formula generalizes Wilson's score interval for binomial proportions [Wil27] to arbitrary bounded random variables.

We propose a new UCB sampling algorithm, ω-UCB, that uses asymmetric confidence intervals to overcome the issues of existing algorithms, which we illustrated in Section 2.2.2 of our related work. We also propose an extension of ω-UCB, called ω^*-UCB, which uses the observed variances of the arms' rewards and costs to further tighten the UCB.

We analyze our approach theoretically. Our analysis shows that ω-UCB enjoys logarithmic regret for parameter $\rho \geq 1$ and sublinear regret in general.

We conduct experiments on typical settings found in the literature and real-world social network advertising data. Our results demonstrate that our algorithms have substantially lower regret than their competitors for both small and large budgets.

We share the code of our implementation and experiments.[1]

[1] github.com/heymarco/OmegaUCB

3.2. Problem Definition

We address the stochastic setting with K arms. Each arm $k \in [K]$ has continuous or discrete reward and cost distributions with unknown expected values $\mu_k^r \in [0, 1)$ and $\mu_k^c \in (0, 1]$, respectively. Assume without loss of generality that arm $k = 1$ has the highest ratio μ_k^r/μ_k^c among all arms. At time t a player chooses an arm k_t and subsequently observes reward feedback $r_t \in [0, 1]$ and cost feedback $c_t \in [0, 1]$. We assume that the arms are independent and that rewards and costs observed at different time steps are i.i.d. This is consistent with previous work [Xia+15a; Xia+15b; Xia+17; Wat+17; Wat+18]. We do not make any assumptions about the correlation between rewards and costs of the same arm. The game ends after T_B plays that exhaust the available budget B.

Let $\mathbb{1}_k(k_t)$ be the indicator function: $\mathbb{1}_k(k_t) = 1$ iff $k_t = k$, else $\mathbb{1}_k(k_t) = 0$. The number of plays and the sample average of rewards and costs of arm k at time T are:

$$n_k(T) = \sum_{t=1}^{T} \mathbb{1}_k(k_t)$$

$$\hat{\mu}_k^r(T) = \frac{1}{n_k(T)} \sum_{t=1}^{T} \mathbb{1}_k(k_t) r_t$$

$$\hat{\mu}_k^c(T) = \frac{1}{n_k(T)} \sum_{t=1}^{T} \mathbb{1}_k(k_t) c_t$$

The player's goal is to minimize the feedback obtained from suboptimal arms. One can measure this as 'pseudo-regret', describing the difference to the cumulative reward R^* of an optimal algorithm, given by $R^* - \mathbb{E} \sum_{t=1}^{T_B} r_t$.

Finding the optimal algorithm in BMAB is known to be NP-hard, due to the 'knapsack problem' [Tra+10]. However, always choosing arm 1 leads to a suboptimality of at most $2\mu_1^r/\mu_1^c$, negligible for not-too-small budgets [Xia+15b]. Thus, previous work [Xia+15a; Xia+15b; Xia+17; Wat+17; Wat+18], as well as our own approach, aim to minimize regret relative to an algorithm that always selects arm 1:

$$\text{Regret} = \sum_{i=1}^{K} \mu_k^c \Delta_k \mathbb{E}[n_k(T_B)], \quad \text{where } \Delta_k = \frac{\mu_1^r}{\mu_1^c} - \frac{\mu_k^r}{\mu_k^c}$$

3.3. ω-UCB

We now detail our algorithm ω-UCB, and analyze it theoretically.

3.3.1. Algorithm

Our algorithm ω-UCB, summarized in Algorithm 3.1, starts by playing each arm once. At each subsequent time step t, the algorithm chooses the arm k_t with the highest upper confidence bound of the ratio of the expected reward μ_k^r to the expected cost μ_k^c. Let $\omega_{k+}^r(\alpha, t)$ denote the upper confidence bound of μ_k^r for a confidence level $1-\alpha$. Similarly, $\omega_{k-}^c(\alpha, t)$ is the lower confidence bound of μ_k^c. ω-UCB chooses k_t according to:

$$k_t = \arg\max_{k \in [K]} \Omega_k(\alpha, t), \text{ where } \Omega_k(\alpha, t) = \frac{\omega_{k+}^r(\alpha, t)}{\omega_{k-}^c(\alpha, t)} \tag{3.1}$$

Unlike other policies that rely on the same principle [Xia+17; Xia+15a; Xia+16; BKS13; CES20], ω-UCB calculates asymmetric confidence bounds that are shifted towards the center of the range of the random variable. This leads to tighter UCB for the reward-cost ratio, especially when an arm's expected cost or the number of plays is low.

Theorem 3.1 (Asymmetric confidence interval)**.** Let X be a random variable bounded in the interval $[m, M]$, with unknown expected value $\mu \in [m, M]$ and variance σ^2. Let z denote the number of standard deviations that lead to a $1-\alpha$ coverage of the standard normal distribution. Let $\hat{\mu}$ be the sample mean of n i.i.d. samples of X. Then,

$$\Pr[\mu \notin [\omega_-(\alpha), \omega_+(\alpha)]] \leq \alpha,$$

with

$$\omega_\pm(\alpha) = \frac{\mathrm{B}}{2\mathrm{A}} \pm \sqrt{\frac{\mathrm{B}^2}{4\mathrm{A}^2} - \frac{\mathrm{C}}{\mathrm{A}}}, \tag{3.2}$$

where

$$\mathrm{A} = n + z^2\eta,\ \mathrm{B} = 2n\hat{\mu} + z^2\eta(M+m),\ \mathrm{C} = n\hat{\mu}^2 + z^2\eta Mm$$

and

$$\eta = \begin{cases} \frac{\sigma^2}{(M-\mu)(\mu-m)} & \text{if } \mu \in (m, M) \\ 1 & \text{if } \mu \in \{m, M\}. \end{cases}$$

Proof. Using the central limit theorem and Bhatia-Davis inequality [BD00], we follow similar steps as [Wil27]. We first handle the case where $\mu \in (m, M)$. We then address the cases $\mu = m$ and $\mu = M$.

Case $\mu \in (m, M)$. The central limit theorem states that for a large enough sample size, $\hat{\mu}$ approximately follows a normal distribution with mean μ and variance σ^2/n. I.e.,

$$\hat{\mu} \sim \mathcal{N}\left(\mu, \sqrt{\frac{\sigma^2}{n}}\right) \iff \frac{\hat{\mu} - \mu}{\sqrt{\frac{\sigma^2}{n}}} \sim \mathcal{N}(0, 1).$$

Therefore, $\hat{\mu}$ likely falls into an interval that is centered around μ and scaled by σ. The value z is the number of standard deviations such that $\hat{\mu}$ falls out of the corresponding confidence interval with a probability of α.

$$\Pr\left[\hat{\mu} \notin \left[\mu - \frac{\sigma}{\sqrt{n}}z, \mu + \frac{\sigma}{\sqrt{n}}z\right]\right] = \alpha \tag{3.3}$$

Next, we apply the Bhatia-Davis inequality [BD00] to express σ as a function of μ. It states that $\sigma^2 \leq (M - \mu)(\mu - m)$. Hence, there exists a factor $\eta \in [0, 1]$ such that

$$\sigma^2 = \eta(M - \mu)(\mu - m). \tag{3.4}$$

This gives us an expression for the interval bounds in Eq. (3.3) that is quadratic w.r.t. μ:

$$(\hat{\mu} - \mu)^2 = \frac{\sigma^2}{n}z^2 = \frac{\eta(M - \mu)(\mu - m)}{n}z^2 \tag{3.5}$$

Solving Eq. (3.5) for μ yields the endpoints $\omega_-(\alpha)$ and $\omega_+(\alpha)$ of our confidence interval:

$$\Pr[\mu \notin [\omega_-(\alpha), \omega_+(\alpha)]] = \alpha$$

with

$$\omega_-(\alpha), \omega_+(\alpha) = \frac{B}{2\mathrm{A}} \pm \sqrt{\frac{B^2}{4\mathrm{A}^2} - \frac{C}{\mathrm{A}}}$$

and

$$\mathrm{A} = n + z^2\eta, \quad B = 2n\hat{\mu} + z^2\eta(M + m), \quad C = n\hat{\mu}^2 + z^2\eta Mm.$$

Cases $\mu = m$ and $\mu = M$. For $\mu = m$ (the case for $\mu = M$ is analogous), the probability that μ is not in the confidence interval $[\omega_-(\alpha), \omega_+(\alpha)]$ is zero, which is less than α. Additionally, we have $\sigma^2 = (M - \mu)(\mu - m) = 0$, which implies that Eq. (3.4) holds for any choice of η. However, since μ is an unknown quantity, we can never be certain that $\mu = m$ based on some sample from X. In the worst-case scenario, X is a random variable with support $\{m, M\}$, with μ greater than and approximately equal to m. In this case, $\eta = 1$ by definition. Hence, we define $\eta = 1$ for $\mu \in \{m, M\}$. Combining the special case that $\mu \in \{m, M\}$ with the result from the previous paragraph gives Theorem 3.1. □

Illustration of asymmetry. The center of the confidence interval is a weighted average of the sample mean and the center of the range of the random variable. This leads to confidence intervals that are shifted towards the center of the range of the random variable.

Algorithm 3.1 ω-UCB

Require: K ▷ Number of arms
Require: B ▷ Available budget
Require: ρ ▷ Confidence interval scaling parameter, defaults to $1/4$
Require: $\boldsymbol{\eta}^r$ ▷ Reward variance parameters, defaults to $[1]^K$
Require: $\boldsymbol{\eta}^c$ ▷ Cost variance parameters, defaults to $[1]^K$

```
t ← 0
n ← [0]^K
while B > 0 do
    if t < k then                                   ▷ Play each arm once
        Play arm t
        Observe r_t, c_t and update μ̂^r_k(t), μ̂^c_k(t)
        n[t] = 1
    else
        z_ρ(t) = √(2ρ log t)          ▷ Corresponds to α(t) < 1 − √(1 − t^{−ρ})
        for all arms k ∈ 1 ... K do
            Compute Ω_k(α, t) from Eq. (3.1) and Eq. (3.2) with z_ρ(t),
            η^r[k], η^c[k], n[k], μ̂^r_k(t), and μ̂^c_k(t)
        Find k_t = arg max_k Ω_k(α, t)
        Play arm k_t
        Observe r_t, c_t and update μ̂^r_k(t), μ̂^c_k(t)
        n[k_t] = n[k_t] + 1
    B = B − c_t
    t = t + 1
```

To see this, we can compare the distance between $\hat{\mu}$ and the interval center $B/2A$ to half the width of the confidence interval:

$$\text{Asymmetry} = \frac{\hat{\mu} - \frac{B}{2A}}{\sqrt{\frac{B^2}{4A^2} - \frac{C}{A}}} \in [0, 1]$$

For Bernoulli random variables, after some derivations and inserting the definitions of A, B, C as specified in Theorem 3.1, we obtain

$$\text{Asymmetry} = \frac{(2\hat{\mu} - 1)^2 z^2}{4n\hat{\mu}(1 - \hat{\mu}) + z^2}$$

Figure 3.1 plots the asymmetry measure for different values of n over $\hat{\mu}$ and $z = 3$. (1) For $\hat{\mu} = 1$ and $\hat{\mu} = 0$, it takes on a maximum value of 1, while for $\hat{\mu} = 0.5$, asymmetry is 0. (2) For a given value of $\hat{\mu} \in (0, 1)$, asymmetry decreases with increasing sample size. (3) Related to this, we see that asymmetry is maximal for a given $\hat{\mu}$ for $n = 1$.

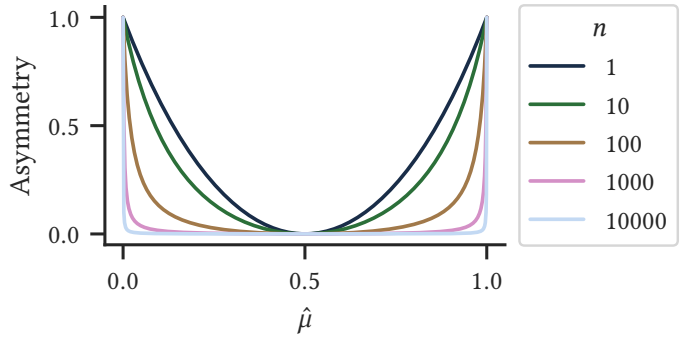


Figure 3.1.: Confidence interval asymmetry for Bernoulli rewards and costs for $z = 3$ and different $n, \hat{\mu}$

Discussion. According to the Bhatia-Davis inequality [BD00], $0 \leq \sigma^2 \leq (M-\mu)(\mu-m)$, and hence $\eta \in [0, 1]$. For the special case of Bernoulli random variables, $\eta = 1$, $m = 0$, $M = 1$, and Theorem 3.1 recovers Wilson's original confidence interval for Binomial proportions [Wil27]. However, this theorem is more flexible than Wilson's original method. It enables tighter confidence intervals for non-Bernoulli costs or rewards by setting $\eta < 1$ when an estimate or an upper bound of the variance is available. Our experiments demonstrate that this flexibility leads to a significant performance improvement.

The following theorem defines an upper confidence bound $\Omega(\alpha)$ for the ratio of the expected values of two random variables. Combined with Theorem 3.1, it facilitates the computation of an arm's index according to Eq. (3.1).

Theorem 3.2 (UCB for ratio of expected values)**.** Let R and C be two bounded random variables with expected values $\mu^r \geq 0$ and $\mu^c > 0$. Let $\omega^r_+(\alpha) \geq 0$ denote the upper confidence bound of R and $\omega^c_-(\alpha) > 0$ the lower confidence bound of C as given in Theorem 3.1. Let $\Omega(\alpha) = \omega^r_+(\alpha)/\omega^c_-(\alpha)$. Then,

$$\Pr\left[\frac{\mu^r}{\mu^c} > \Omega(\alpha)\right] \leq \alpha$$

Proof. Define events $E_1 = \mu^r > \omega^r_+(\alpha)$ and $E_2 = \mu^c < \omega^c_-(\alpha)$. A violation of the UCB of the reward-cost ratio requires that either E_1 or E_2 occurs, or that both events happen simultaneously. Therefore, by the union bound, we have that

$$\Pr\left[\frac{\mu^r}{\mu^c} > \Omega_k\right] = \Pr\left[\frac{\mu^r}{\mu^c} > \frac{\omega^r_+(\alpha)}{\omega^c_-(\alpha)}\right] \leq \Pr[E_1] + \Pr[E_2]$$

E_1 and E_2 both occur with probability $\leq \alpha/2$, hence $\Pr\left[\mu^r/\mu^c > \Omega_k\right] \leq \alpha$. □

A UCB-sampling algorithm that keeps parameter α constant leads to linear regret in the worst case. This is because such an algorithm will eventually stop exploring arms that may have high costs and low rewards in the beginning. To avoid this problem, ω-UCB decreases the value of α as the time t progresses, similarly to the UCB1-algorithm [ACF02].

This adaptive approach helps to ensure continued exploration of arms and guarantees sub-linear regret. Theorem 3.3 introduces the scaling law and relates it to the confidence level.

Theorem 3.3 (Time-adaptive confidence interval)**.** For an arm k, let μ_k^r be its expected reward, μ_k^c its expected cost, and $\Omega_k(\alpha, t)$ the upper confidence bound for μ_k^r/μ_k^c, as in Eq. (3.1). For $\rho, t > 0$, and $\alpha(t) < 1 - \sqrt{1 - t^{-\rho}}$ it holds that

$$\Pr\left[\Omega_k(\alpha, t) \geq \frac{\mu_k^r}{\mu_k^c}\right] \geq 1 - \alpha(t),$$

that is, the upper confidence bound holds asymptotically almost surely.

Proof. We start from Theorem 3.1 and equate the confidence level $1 - \alpha$ of the individual reward and cost distributions to the number of standard deviations z. This involves the cumulative density function (cdf) of the standard normal distribution. We then replace the cdf with an approximation that has a closed-form solution for z. Our choice of z cancels out the exponential term in this approximation, similar to the UCB1-algorithm [ACF02]. Last, we apply Theorem 3.2 to obtain the final result.

Step 1. We relate the confidence level $1-\alpha(t)$ at time t to the cumulative density function of the standard normal distribution (erf abbreviates the error function).

$$1 - \frac{\alpha(t)}{2} = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{z}{\sqrt{2}}\right)\right)$$

Solving for $\alpha(t)$ yields:

$$\alpha(t) = 1 - \operatorname{erf}\left(\frac{z}{\sqrt{2}}\right)$$

Step 2. We now replace the error function $\operatorname{erf}\left(\frac{z}{\sqrt{2}}\right)$ in the equation above with a series expansion based on Bürmann's theorem [SS14; WW21]; we summarize all but the first addend in a remainder term $\gamma\left(\frac{z}{\sqrt{2}}\right) > 0$. This term has a maximum of $\gamma(0.71) \approx 0.0554$ and approaches 0 for larger z:

$$\alpha(t) = 1 - \left(\sqrt{1 - \exp\left(-\frac{z^2}{2}\right)} + \gamma\left(\frac{z}{\sqrt{2}}\right)\right)$$

Omitting the γ-term gives an upper bound for $\alpha(t)$:

$$\alpha(t) < 1 - \sqrt{1 - \exp\left(-\frac{z^2}{2}\right)} \tag{3.6}$$

Step 3. Next, we choose z as a function of $\log t$ and $\rho > 0$, $z_\rho(t) = \sqrt{2\rho \log t}$. This results in a time-increasing confidence level $1 - \alpha(t)$:

$$\Pr[\mu \notin [\omega_-(\alpha(t)), \omega_+(\alpha(t))]] \leq \alpha(t) \text{ with } \alpha(t) < 1 - \sqrt{1 - t^{-\rho}}$$

Step 4. Applying Theorem 3.2 gives

$$\Pr\left[\frac{\mu_k^r}{\mu_k^c} > \Omega_k(\alpha, t)\right] \leq \alpha(t) \text{ with } \alpha(t) < 1 - \sqrt{1 - t^{-\rho}}.$$

The complementary event, $\Omega_k(\alpha, t) \geq \mu_k^r/\mu_k^c$, holds with a probability of at least $1 - \alpha(t)$,

$$\Pr\left[\Omega_k(\alpha, t) \geq \frac{\mu_k^r}{\mu_k^c}\right] \geq 1 - \alpha(t) \text{ with } \alpha(t) < 1 - \sqrt{1 - t^{-\rho}},$$

which is the result given in Theorem 3.3.

□

With $\alpha(t) < 1 - \sqrt{1 - t^{-\rho}}$, the confidence level $1 - \alpha$ approaches 1 as time t goes to infinity. This encourages the exploration of arms that are played less frequently. Moreover, it establishes a logarithmic dependence between z in Theorem 3.1 and t, i.e., $z_\rho(t) = \sqrt{2\rho \log t}$, which will be useful in our regret analysis. The next section analyzes the worst-case regret of ω-UCB. To simplify notation, we abbreviate $\Omega_k(\alpha, t)$ as $\Omega_k(t)$, $\omega_{k-}^c(\alpha, t)$ as $\omega_{k-}^c(t)$, and $\omega_{k+}^r(\alpha, t)$ as $\omega_{k+}^r(t)$ in our regret analysis.

3.3.2. Regret Analysis

We first bound the expected number of suboptimal plays $\mathbb{E}[n_k(\tau)]$ before some time step τ. We use the result to derive the regret bound of ω-UCB (cf. Theorem 3.5). For improved clarity, we omit longer proofs here but provide them in Appendix A.1.

Theorem 3.4 (Number of suboptimal plays)**.** For ω-UCB, the expected number of plays of a suboptimal arm $k > 1$ before time step τ, $\mathbb{E}[n_k(\tau)]$, is upper-bounded by

$$\mathbb{E}[n_k(\tau)] \leq 1 + n_k^*(\tau) + \xi(\tau, \rho), \tag{3.7}$$

where

$$\xi(\tau, \rho) = (\tau - K)\left(2 - \sqrt{1 - \tau^{-\rho}}\right) - \sum_{\mathrm{t=K+1}}^{\tau} \sqrt{1 - t^{-\rho}}, \tag{3.8}$$

$$n_k^*(\tau) = \frac{8\rho \log \tau}{\delta_k^2} \max\left\{\frac{\eta_k^r \mu_k^r}{1 - \mu_k^r}, \frac{\eta_k^c (1 - \mu_k^c)}{\mu_k^c}\right\}, \tag{3.9}$$

$\delta_k = \Delta_k/(\Delta_k + 1/\mu_k^c)$, and K and Δ_k are defined as before, cf. Section 3.2.

Proof. Appendix A.1.1 contains the proof of the theorem. The derivation of δ_k is in Appendix A.1.2. □

The 1 in Eq. (3.7) represents the very first pull of arm k. We interpret the term $n_k^*(\tau)$ as the number of plays that leads to a "sufficiently small" deviation between μ_k^r/μ_k^c and $\hat{\mu}_k^r(t)/\hat{\mu}_k^c(t)$. This number is logarithmic w.r.t. τ. The term $\xi(\tau, \rho)$ represents those plays that occur despite the arms' sufficiently small deviation between the true reward-cost ratio and the ratio of the sample means.

For each suboptimal play, the player suffers a greater-than-zero regret in expectation. Hence, the number of suboptimal plays is closely linked to the regret. Lemma 4 in [Xia+17] establishes this connection for BMAB. In combination with Theorem 3.4, this lemma thus yields the worst-case regret for ω-UCB.

Theorem 3.5 (Finite-budget instance-dependent regret)**.** Define Δ_k, $n_k^*(\tau_B)$, and $\xi(\tau, \rho)$ as before, and let $\tau_B = \lfloor 2B/\min_{k\in[K]} \mu_k^c \rfloor$. For any $\rho > 0$, ω-UCB suffers instance-dependent regret of

$$\text{Regret} \leq \sum_{k=2}^{K} \Delta_k \left(1 + n_k^*(\tau_B) + \xi(\tau_B, \rho)\right) + \mathcal{X}(B) \sum_{k=2}^{K} \Delta_k + \frac{2\mu_1^r}{\mu_1^c}, \tag{3.10}$$

where $\mathcal{X}(B)$ is in $\mathcal{O}\left((B/\mu_{min}^c)e^{-0.5B\mu_{min}^c}\right)$.

Proof. Lemma 4 of [Xia+17] provides an algorithm-independent regret expression for BMAB policies:

$$\text{Regret} \leq \sum_{k=2}^{K} \Delta_k \mathbb{E}[n_k(\tau_B)] + \mathcal{X}(B) \sum_{k=2}^{K} \Delta_k + \frac{2\mu_1^r}{\mu_1^c} \tag{3.11}$$

where

$$\tau_B = \left\lfloor \frac{2B}{\min_{k\in[K]} \mu_k^c} \right\rfloor \text{ and } \mathcal{X}(B) \text{ is in } \mathcal{O}\left((B/\mu_{min}^c)e^{-0.5B\mu_{min}^c}\right).$$

Substituting $\mathbb{E}[n_k(\tau_B)]$ in Eq. (3.11) with the result from Theorem 3.4 completes the proof. □

Theorem 3.5 and our definition of $n_k^*(\tau)$ show that the regret of ω-UCB decreases for small η_k^r and η_c^k. I.e., when the reward and cost distributions have a small variance compared to a Bernoulli variable with the same expected value.

The term $\xi(\tau_B, \rho)$ decreases, while $n_k^*(\tau)$ increases with ρ. Further derivations show that for increasingly large budgets, $\xi(\tau_B, \rho)$ converges if $\rho > 1$, grows logarithmic if $\rho = 1$, and superlogarithmic (in the order of $\mathcal{O}(B^{1-\rho})$) if $\rho < 1$; see Appendix A.1.4 for the details. This results in the following asymptotic behavior:

Theorem 3.6 (Asymptotic regret)**.** The regret of ω-UCB is in

$$\text{Regret} \in \begin{cases} \mathcal{O}\left(B^{1-\rho}\right), & \text{if } 0 < \rho < 1, \\ \mathcal{O}(\log B), & \text{if } \rho \geq 1. \end{cases}$$

Proof. See Appendix A.1.4. □

3.3.3. Evaluation

This section presents the experimental setup used to evaluate the policies. We introduce the MAB settings, followed by the configurations of ω-UCB and its competitors. We conducted the experiments on an Ubuntu 20.04 server with x86-architecture, using 32 cores, each running at 2.0 GHz, and 128 GB of RAM.

3.3.3.1. BMAB settings

We use MAB settings based on synthetic and real data, which we describe separately. Each setting comprises a specific combination of reward and cost distributions, as well as the number of arms K. See Table 3.1 for a summary.

Table 3.1.: BMAB evaluation settings

Type	Distribution	Parameters	K	Used in	ID
Synthetic	Bernoulli	$\mathcal{U}(0,1)$	10	[Xia+15b; Xia+17]	S-Br-10
			50	[Xia+17]	S-Br-50
			100	[Xia+15a; Xia+15b]	S-Br-100
	General. Bernoulli	$\mathcal{U}(0,1)$	10	[Xia+15b; Xia+16]	S-GBr-10
			50	[Xia+16]	S-GBr-50
			100	[Xia+15b]	S-GBr-100
	Beta	$\mathcal{U}(0,5)$	10	[Xia+17; Xia+16]	S-Bt-10
			50	[Xia+17; Xia+16]	S-Bt-50
			100	[Xia+15a]	S-Bt-100
Facebook	Bernoulli	given	$[2, 97]$	–	FB-Br
	Beta	random	$[2, 97]$	–	FB-Bt

Synthetic data. Previous studies use synthetic settings with rewards and costs drawn from discrete (Bernoulli or Generalized Bernoulli with outcomes $\{0.0, 0.25, 0.5, 0.75, 1.0\}$) or continuous (Beta) distributions [Xia+15a; Xia+17; Wat+17; Wat+18]. These studies typically generate parameters randomly within a given range [Xia+17; Xia+15a; Xia+15b;

Xia+16] and use 10 to 100 arms [Xia+15a; Xia+15b; Xia+17; Xia+16; Wat+18]. We adopt this and set the parameter ranges to those used in related work.

Social-media advertisement data. We also evaluate our algorithm in a social media advertisement scenario described in Example 1.2. We use real-world data from [Lem17]. It contains information about different ads based on their target gender (female or male) and age category (30–34, 34–39, 40–44, 45–49), along with the number of displays and clicks, the total cost, and the number of purchases. We group the ads by target gender and age category, resulting in 19 "advertisement campaigns" (BMAB settings). Each campaign has between 2 and 93 ads (arms). We compute each ad's expected rewards μ_k^r and costs μ_k^c as the average revenue per click and the average cost per click, respectively. We model both discrete and continuous rewards and costs. For the discrete case, we sample rewards and costs from two Bernoulli distributions with expected values of μ_k^r and μ_k^c, respectively. In the continuous case, we use a Beta distribution and sample the distribution parameters from a uniform distribution with a range of (0, 5). We then adjust one of the parameters to ensure that the expected values of rewards and costs match μ_k^r and μ_k^c.

3.3.3.2. Policies

We test the performance of two variants of our algorithm: ω-UCB and ω^*-UCB. The ω-UCB variant uses a fixed value of $\eta = 1$ for rewards and costs. The ω^*-UCB variant uses $\eta_k^r = \eta_k^c = 1$ as default but estimates their values using sample variance and mean once arm k has been played sufficiently many times ($n_k(t) \geq 30$), $\hat{\eta}_k = \hat{\sigma}_k^2/((M - \hat{\mu}_k)(\hat{\mu}_k - m))$, where symbols with a hat ($\hat{}$) refer to sample estimates. We experiment with two values of the hyperparameter ρ: $\rho = 1$ and $\rho = 1/4$. The former is the minimum value for which we have proven logarithmic regret. The latter has performed well in our sensitivity study, as we will demonstrate in Section 3.3.3.4.

We compare the performance of our algorithm to several other state-of-the-art BMAB policies, including BTS, Budget-UCB, i-UCB, c-UCB, m-UCB, b-greedy, UCB-SC+, and UCB-B2. We set the hyperparameters for each competitor to the values recommended in their respective papers. Appendix A.1.5 details the policies and their hyperparameters.

3.3.3.3. Results

To observe the asymptotic behavior of the policies, we set the budget B to $1.5 \cdot 10^5$ times the minimum expected cost. We run each algorithm until the available budget is depleted and report the average results over 100 independent repetitions with the repetition index as the seed for the random number generator. Since we draw the expected cost μ_k^c and expected reward μ_k^r randomly in each repetition of the experiment, we normalize the budget in our graphs. Note that, in some instances, an approach may not appear in a graph if its regret exceeds the graph's upper y-axis limit.

Synthetic Bernoulli. Figure 3.2a shows the regret of the policies for Bernoulli-distributed rewards and costs with 95 % confidence intervals. We do not present ω^*-UCB in this experiment since its results are almost identical to ω-UCB. ω-UCB and BTS achieve logarithmic regret and demonstrate better asymptotic behavior than other methods. Although {i,c,m}-UCB may outperform ω-UCB with $\rho = 1$ for small budgets, their regret grows rapidly as the budget increases, indicating poor asymptotic behavior. One surprising observation is that UCB-B2 is not competitive in our experiments despite its tight and accurate UCB (cf. Figure 2.2). We assume that this approach decreases the confidence level too fast, which we found can result in over-exploration. For $K = 50$ and $K = 100$, our algorithm has lower regret than BTS. BTS outperforms ω-UCB with $\rho = 1$ only on the 10-armed bandit. ω-UCB with $\rho = 1/4$ outperforms all other policies on small and large budgets and regardless of K. Comparing $\rho = 1/4$ with $\rho = 1$, one can see that the curve for $\rho = 1$ is linear (the x-axis is logarithmic), while for $\rho = 1/4$ it is convex. We conclude that $\rho = 1/4$ leads to smaller regret than $\rho = 1$ for not-too-large budgets but that $\rho = 1$ performs better asymptotically.

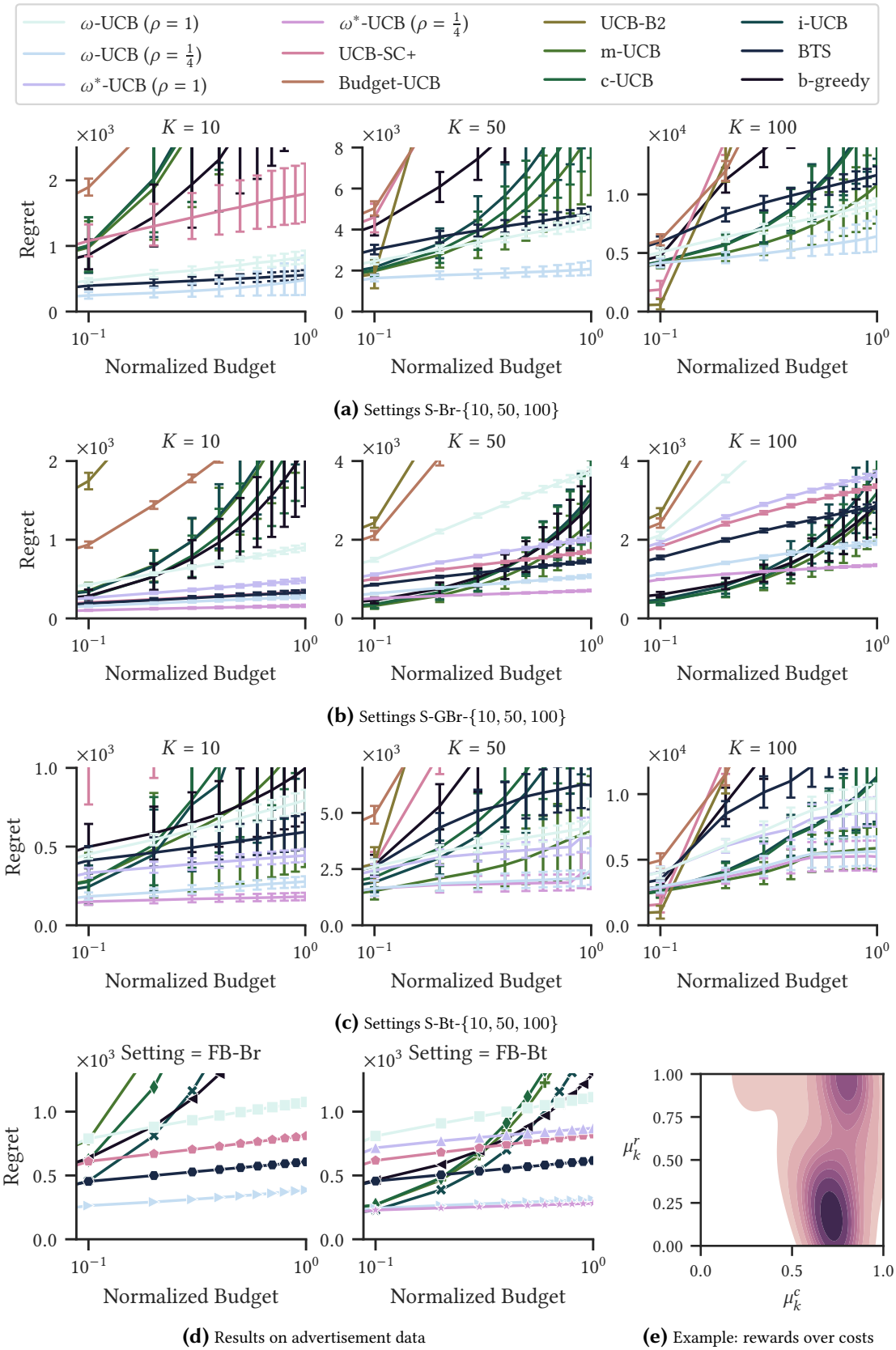
Synthetic Generalized Bernoulli and synthetic Beta. Figure 3.2b shows the policies' regret for rewards and costs drawn from Generalized Bernoulli distributions, and Figure 3.2c for Beta distributed rewards and costs. Besides ω-UCB, we also present the results for ω^*-UCB, which estimates η_k^r and η_k^c from the sample variance of rewards and costs. ω-UCB and ω^*-UCB with $\rho = 1/4$ outperform their competitors, except for $K = 100$ in the Beta bandit where m-UCB performs comparable. We also notice that BTS is not competitive in this evaluation setting, likely because the algorithm cannot account for the (often very small) variance of the sampled Beta distributions. Our ω^*-UCB algorithm is advantageous in such cases.

Social-media advertisement data. Figure 3.2d shows the results of our study on social-media advertisement data[2] [Lem17]. Here, the default choice $\rho = 1/4$ outperforms all other competitors. $\rho = 1/4$ also outperforms $\rho = 1$ significantly, although both choices show good asymptotic behavior. The advertisement data settings seem to be easier for some competitors (BTS, KL-UCB-SC+) and harder for others (i,c,m-UCB). The likely cause is that the distribution of expected rewards and costs between arms is non-uniform: costs are biased towards 1, and rewards are biased towards the boundaries of $[0, 1]$, as the kernel density estimation (KDE) plot for a BMAB with $K = 33$ in Figure 3.2e illustrates exemplary. Last, we observe that ω^*-UCB has lower regret than ω-UCB, although the effect is not as strong as in the synthetic settings.

3.3.3.4. Sensitivity study

We investigate the performance difference between ω-UCB and ω^*-UCB, as well as the sensitivity of our algorithm with respect to the hyperparameter ρ. The results, based on

[2] Available at `kaggle.com/datasets/madislemsalu/facebook-ad-campaign` under a PDDL license.



(a) Settings S-Br-{10, 50, 100}

(b) Settings S-GBr-{10, 50, 100}

(c) Settings S-Bt-{10, 50, 100}

(d) Results on advertisement data

(e) Example: rewards over costs

Figure 3.2.: Evaluation results of BMAB algorithms

our synthetic settings (cf. Table 3.1) for $K = 50$, are shown in Figure 3.3. We omit the results for $K = 10$ and $K = 100$ as they look similar to $K = 50$. ω-UCB and ω^*-UCB perform best with $\rho = 1/4$ when rewards and costs follow Bernoulli distributions. Both policies achieve comparable performance in this case. It appears that estimating η (which is 1 in Bernoulli bandits) does not result in a performance decrease of ω^*-UCB compared to ω-UCB. Also, even though $\rho = 1/8$ works well for ω-UCB when rewards and costs follow a generalized Bernoulli or Beta distribution, $\rho = 1/4$ remains a near-optimal choice for ω^*-UCB. Based on these results, we recommend using ω^*-UCB with $\rho = 1/4$ as a default.

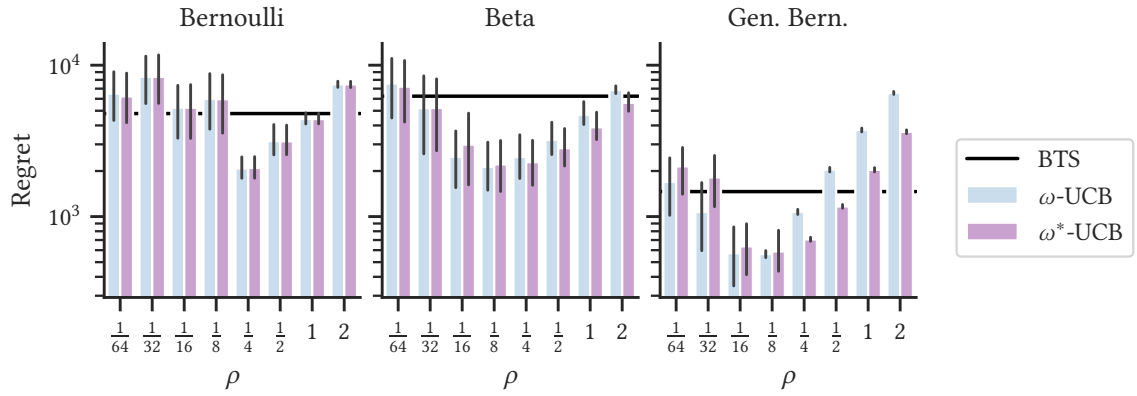


Figure 3.3.: Sensitivity study showing the regret for different choices of ρ

3.4. Summary

This chapter presented ω-UCB, a BMAB algorithm which leverages asymmetric confidence intervals to accurately bound the ratio between rewards and costs. It also discussed ω^*-UCB, which takes into account the observed variances of the reward and costs. While ω-UCB has strong theoretical guarantees, ω^*-UCB excels empirically by tightening the confidence intervals based on the observed variances.

Zooming out, this chapter addressed the challenge of learning from decision-based partial feedback under a finite budget. The next chapter takes a different perspective by presenting algorithms for full and observation-based partial feedback settings.

4. Feedback-Efficient Incremental Decision Tree Mining

The content of this chapter is based on the following publication.

- Marco Heyden et al. "Leveraging plasticity in incremental decision trees". In: *Machine Learning and Knowledge Discovery in Databases. Research Track. ECML PKDD 2024*. Vilnius, Lithuania: Springer Nature, 2024, pp. 38–54. ISBN: 978-3-031-70362-1. DOI: 10.1007/978-3-031-70362-1_3

4.1. Chapter Overview

Commonly used incremental decision trees for mining data streams include the Hoeffding Tree (HT) and Extremly Fast Decision Tree (EFDT) algorithms. EFDT exhibits faster learning than HT. However, due to its split revision procedure, EFDT suffers from sudden and unpredictable accuracy decreases caused by subtree pruning (Figure 1.8).

We propose PLASTIC, an incremental decision tree that restructures the otherwise pruned subtree to overcome the sudden and unpredictable accuracy decreases in EFDT. We also propose a simple yet effective change-adaptive variant, PLASTIC-A.

We introduce the concept of decision tree plasticity: restructuring a decision tree without affecting its predictions. PLASTIC leverages this concept to alleviate the negative effects of sudden and unexpected subtree pruning during split revision.

We compare our algorithms to state-of-the-art incremental decision tree algorithms on 9 synthetic and 14 real-world data streams. Our results on synthetic data show that PLASTIC is able to avoid EFDT's sudden accuracy drops, leading to benefits of up to 50 % accuracy. Real-world data validates our findings: nonadaptive PLASTIC remains highly competitive against even adaptive decision trees. However, our adaptive variant, PLASTIC-A, surpasses both, achieving accuracy gains of up to 10 % on data streams with strong temporal dependencies.

We share the code of our implementation and experiments.[1][2]

[1] github.com/heymarco/PLASTIC (java)

[2] github.com/heymarco/CapyMOA-PLASTIC (python)

4.2. Problem Definition

We consider the setting of data stream classification. A data stream S is an infinite sequence of observations $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_t, y_t), \ldots\}$ that arrive over time, where t denotes the current time step. We assume that all $\mathbf{x}_i \in S$ are from the same d-dimensional domain $\mathcal{S}$, which is usually numerical, categorical, or mixed. We call the j-th attribute a_j. For an instance $\mathbf{x}_i$, its j-th attribute value is $\mathbf{x}_{ij}$. If a_j is categorical, the number of its distinct categories is ν_j. Each $\mathbf{x}_i$ has a label y_i that belongs to a finite set of categories $[K]$. A learner's goal is to learn a model m such that $y_i = m(x_i)$. We make the common assumption that $\mathbf{x}_t$ and y_t are revealed simultaneously, i.e., in each time step the learner can access the tuple $(\mathbf{x}_t, y_t)$ to construct m [Bif+18b].

4.3. PLASTIC

Our algorithm adopts EFDT's split-and-revise strategy: it splits a node early and revises the split when more data becomes available. Thus, both approaches share the same structure shown in Algorithm 2.3. However, while split-revision may trigger subtree pruning in EFDT, it leads to subtree restructuring in PLASTIC. To give an intuition, Section 4.3.1 describes subtree restructuring for categorical attributes on a conceptual level. Section 4.3.2 introduces the algorithm on a more technical level and discusses adaptations for numerical and binary categorical splits.

4.3.1. Concept

PLASTIC exploits the fact that the order in which an instance traverses the nodes of a decision tree from root to leaf does not affect prediction.

Example 4.1 (Decision tree plasticity)**.** Consider the left-most branch ■—●—○ from root (■) to leaf (○) in Figure 4.1. Any instance with attribute values ■ $= 0$ and ● $= 0$ will arrive at ○. Hence, from the viewpoint of the leaf, ■—●—○ $\equiv$ ●—■—○.

For a given branch, this means that one can freely rearrange the order of nodes without affecting any predicted label. Similarly, by rearranging the branches of a decision tree systematically, one can change its structure without affecting its predictions. We refer to this concept as *decision tree plasticity*.

Our idea is to leverage the concept of plasticity during split revision to facilitate the desired split without discarding the affected subtree. In the example, assume that split revision has identified ● as the currently best split for the root node in Figure 4.1, instead of splitting at the current attribute ■.

In response, EFDT would prune the subtree and resplit the root. PLASTIC, in contrast, restructures the subtree. It first creates a representation of the subtree in which its

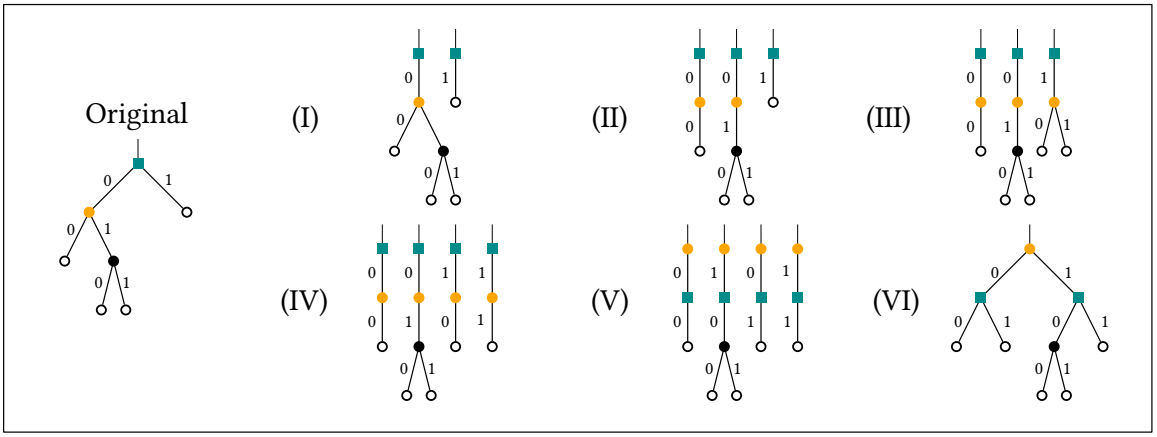


Figure 4.1.: Illustration of PLASTIC

branches are disconnected from each other. The algorithm then restructures each branch and reconnects them. Our algorithm has the following steps:

Steps (I) and (II). Create a disconnected representation of the subtree. If a node already contains the desired split attribute (e.g., the ●-node in Figure 4.1) the branches below remain unchanged.

Steps (III) and (IV). Before restructuring can occur, PLASTIC requires each branch to contain a node with the desired split (●). Thus, our approach enforces the desired split at the leaf of a branch if necessary. Then it disconnects those new splits, similarly to the transition between Steps (I) and (II). As a result, all branches now contain the desired split at their second-to-last node.

Step (V). Move the second-to-last node in each branch to the root position.

Step (VI). Rebuild the subtree. The first branch serves as the initial decision tree prototype for attaching all other branches iteratively: For each branch, except for the first one, the algorithm traverses the prototype until there is a divergence. For example, the second branch in Step (V) diverges already at the root, so PLASTIC attaches it as a new child to the root. The third branch follows the left path of the subtree and diverges at ■. So the approach creates another branch under the left ■-node. The last branch follows the right path and diverges at ■ and thus becomes another child under the right ■-node.

4.3.2. Algorithm

Algorithm 4.1 describes decision tree restructuring. RESTRUCTURE requires the root r of the subtree to be restructured, and the new desired split attribute a^*. In case of a numerical

or binary categorical split, the procedure also requires the split threshold or split category v^*. In the following, we simply call v^* *split value*.

Our algorithm starts by creating a copy of the subtree's root node (line 2) and initializing two queues Q and F (line 3 and 4): Q contains the disconnected branches. Each such branch is represented as a list of tuples. The i-th tuple contains the i-th node of a branch together with the split value that identifies the $i+1$-th node of the branch. After initialization, each branch only has one tuple. It contains the root and a split value that identifies the respective child. F will contain the finished branches, i.e., the ones that have the structure as shown in step (V), which our algorithm can incorporate in the restructured tree.

In the while-loop, our algorithm extends the branches in Q until they have the structure shown in step (V) (line 6). The algorithm then moves these branches to F. As soon as F contains branches, PLASTIC incorporates them into the restructured decision tree (lines 7–9). After PLASTIC has restructured all branches, it resets the counters at the internal nodes which have become invalid. This does not affect prediction, which is done by the leaf nodes. At last, our approach replaces the original subtree with its restructured counterpart (line 12).

Numeric and binary categorical splits. When dealing with numerical and binary categorical splits, not only the split attribute must be the desired one, but also the split value must match with the desired split value. Since the split values of numerical splits typically differ, our algorithm adjusts the node's split values to the desired value and resets the invalid counters within the subtree. By adjusting the split value, some part of the tree might have become unreachable. This occurs when the split value of a successor node, which splits at the same attribute, falls outside the adjusted range. Our algorithm eliminates such unreachable branches.

PLASTIC supports binary categorical splits through a set of transformations. In Figure 4.2, v is the current split value of a binary split, v^* is the desired split value, and shaded branches stand for *multiway* splits with one child per category. Our algorithm applies these transformations to nodes that have already split at the desired attribute but whose split type (binary/multiway) or split value does not match v^*. This allows our algorithm to preserve most of the subtree without requiring pruning to accommodate the desired split.

For the ease of discussion, we refer to 'root' as the ●-node that is visible in Figure 4.2, not to the root node of the restructured subtree.

Case 1: Multiway–Binary. PLASTIC enforces the desired split at the root. It reattaches all previous children (except the v^*-child) to the newly created $\neg v^*$-child. This operation increases the subtree depth by 1.

Case 2: Binary–Multiway. The root currently splits at the desired split attribute but uses split value v instead of v^*. PLASTIC enforces the desired multiway split at the root and reattaches the children of the original v-child.

Case 3: Binary–Binary. The root currently splits at the desired split attribute but uses split value v instead of v^*. PLASTIC enforces the desired binary split at the $\neg v$-child. This increases the subtree depth by 1. Next, the approach attaches the subtree under the original $\neg v$-child to the $\neg v^*$-child. Last, the approach swaps the newly created v^*-child with the original v-child that was still attached to the root.

Algorithm 4.1 PLASTIC

```
1:  procedure RESTRUCTURE(subtree root r, desired attribute a*, desired split value v*)
2:      r' ← a copy of r
3:      Q ← {(r, v) : v that identify the children of r}        ▷ Unfinished branches
4:      F ← {}                                                  ▷ Branches with structure as in step (V)
5:      while Q or F not empty do
6:          DECOUPLEBRANCHES(Q, F)                              ▷ Steps (I) – (V)
7:          while F not empty do
8:              b ← F.popleft()
9:              ADDBRANCHTOTREE(r', b)                          ▷ Step (VI)
10:     Reset counters in restructured nodes (excl. r')
11:     Replace r with restructured subtree r'
```

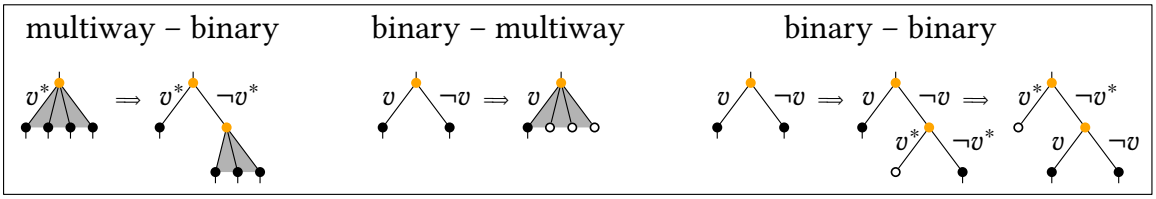


Figure 4.2.: Transformations to handle different types of categorical splits

Runtime discussion. During inference, each new data point traverses the tree until it reaches a leaf. Hence, the time for inference depends on the depth of the respective branch. This is the same as in HT and EFDT. The same holds when updating the counters. The complexity of restructuring a given subtree is linear with regard to its size (number of nodes).

Complexity of decision tree restructuring. We make two worst-case assumptions: (1) The subtree is balanced with a depth of n. (2) The nodes in the subtree split at the n highest-cardinality attributes. Let $\mathcal{C}$ be the set of all ν_j for $j = 1, 2, \ldots, d$. For numerical attributes, let $\nu_j = 2$. Now define

$$C = \{\nu_j \in \mathcal{C} : \nu_j \text{ is among the } n \text{ largest values in } \mathcal{C}\}\,.$$

Then, $\prod_{\nu_j \in C} \nu_j$ is an upper bound on the number of subtree leaves. Last, let ν^* be the number of distinct categories of the desired split attribute. Since PLASTIC enforces a split of the leaves at the desired split attribute, the worst-case maximum number of restructured

branches is $\nu^* \prod_{\nu_j \in C} \nu_j \leq \nu^* \nu_{max}^n$ where $\nu_{max} = \max(C)$. The quantity $\nu^* \nu_{max}^n$ is also the maximum size of any restructured subtree. Hence, restructuring grows linearly with the subtree size. □

To limit PLASTIC's worst-case runtime for large decision trees, we configure it to traverse the restructured subtree only until a maximum depth λ (the full pseudocode in Appendix A.2.1 shows this optional hyperparameter). When fixing λ, the worst-case complexity becomes constant. If runtime is critical, one can also restructure the subtree on a separate thread and use the old subtree in the meantime. In our experiments with $\lambda = 5$, runtime has not been an issue.

4.3.3. Evaluation

4.3.3.1. Algorithms

We compare PLASTIC to its closest competitors HT [DH00] and EFDT [MWS18] and against EFHAT [MSW22], which is the current state-of-the-art adaptative incremental tree for data streams with concept drift. We also include a simple adaptive baseline version of our approach: PLASTIC-A detects changes in accuracy and starts training a background tree upon change detection. The background tree replaces the current tree once it is more accurate. This is conceptually similar to EFHAT, but much simpler: EFHAT relies on fine-grained monitoring at each tree node. In contrast, PLASTIC-A only detects changes at the root of the tree.

All approaches use a significance level of $\alpha = 0.001$ when evaluating splits. In the case of EFDT and PLASTIC(-A), we reevaluate splits with the same frequency as HT evaluates whether a leaf should be split (the *grace period* parameter). We leave the grace period at its default choice of 200. We use this interval in EFHAT and PLASTIC-A to determine the number of instances these approaches should have seen before considering the background tree as replacement. As in [MSW22], we instantiate EFHAT with an ADWIN change detector in its default configuration. For PLASTIC and PLASTIC-A, we use a default value of $\lambda = 5$ that seemed to provide a good runtime-accuracy-tradeoff in general. Majority voting is used for prediction.

4.3.3.2. Data streams

We evaluate the approaches on 9 synthetic and 14 real-world data streams using a test-then-train methodology, in which instance is first used for prediction and then for training. The synthetic data streams are free of concept drift and instances are i.i.d. The real-world data streams may contain concept drift and violate the i.i.d.-assumption. This allows to examine the algorithms' behavior in controlled environments and in real-world scenarios. For synthetic data, we generate 2×10^5 instances, repeat each experiment 30 times and set the generators' random seeds to the experiment repetition index. We do not require

multiple experiment repetitions on real-world data as all approaches are deterministic and the order of data points is given.

Synthetic generators. We use MOA's [Bif+10b] synthetic data stream generators for classification. Unless stated otherwise, we use their default configuration.

- **Agrawal** creates a binary classification problem with nine attributes, out of which six are numerical and three categorical.
- **Hyperplane** creates a binary classification problem in which the class label specifies whether a data point lies above or below a randomly parameterized hyperplane.
- In **LED**, the task is to predict the digits displayed on a seven-segment display.
- **RandomRBF** produces a radial basis function stream. Samples cluster normally distributed around a set of randomly chosen centroids. An instance's class label is the label of the closest centroid.
- **RandomTree-(c, m, n)** are variants of MOA's random tree generator that create a 5-class classification problem by sampling instances from a randomly created decision tree with ten binary categorical attributes (c), five binary categorical and numerical attributes (m), and ten numerical attributes (n), respectively.
- **SEA** generates a binary classification problem from three numerical attributes, of which only two are relevant for classification. By default, the generator adds 10 % noise to the data.
- **Waveform** generates a 3-class classification problem with 21 numerical attributes based on a random differentiation of waveforms.

Real-world data streams. We evaluate our approach on the same real-world data streams used by our closest competitor EFDT. Exceptions are the datasets Skin and Font: we refrain from using them as they do not represent real-world streaming data and are (almost perfectly) sorted by label. Instead, we add a more recent activity recognition data stream, HARTH, as well as RIALTO, INSECTS, and SENSORS, the largest data streams from the USP change point detection benchmark [Sou+20]. Refer to Table 4.1 for a summary.

4.3.3.3. PLASTIC vs. EFDT

We first compare PLASTIC to its predecessor EFDT. This lets us showcase the effect of PLASTIC's restructuring procedure vs. EFDT's subtree pruning. Figure 4.3 shows the accuracy difference between both approaches on synthetic data. Figure 4.4 shows the same for real-world data. For example, a value of 10 in the plots means that PLASTIC outperformed EFDT by 10 % at that point in the stream. Respectively, a negative value means that EFDT outperformed PLASTIC. Note that we will compare the average accuracy between all approaches in Section 4.3.3.4. We also provide the maximum and minimum

Table 4.1.: Real-world data streams

Dataset	Classification task	# Inst.	# Num.	# Cat.	# Class
RIALTO [LHW16]	Environment monitoring	82,250	27	–	10
SENSORS [Zhu10]		2,219,803	5	–	54
COVTYPE [BD99]		581,012	41	11	7
HARTH [Log+21]	Human activity recognition	6,461,328	6	–	12
PAMAP2 [Rei12]		3,850,505	52	–	25
WISDM [Wei19]		15,630,426	1	3	6
HEPMASS [Whi16]	Physics particle detection	10,500,000	28	–	2
HIGGS [BSW14]		11,000,000	28	–	2
SUSY [Whi14]		5,000,000	18	–	2
AIRLINES [IGD11]	Flight delay (yes, no)	539,383	4	3	2
GAS [Fon15]	Gas with highest concentration	4,178,504	16	–	3
INSECTS [Sou+20]	Flying insects classification	905,145	33	–	22
KDD [Sto+98]	Network intrusion detection	494,021	34	7	23
POKER [RO06]	Prediction of Poker hands	1,025,010	–	10	10

values across the experiment repetitions for synthetic data, indicated by the shaded area. For real-world data, we omitted the results for POKER and SUSY from the figure. In POKER, no tree-revision took place, resulting in identical results for PLASTIC and EFDT; the results for SUSY looked similar to the ones for HEPMASS.

We observe in Figure 4.3 that EFDT's performance decreases frequently and suddenly compared to PLASTIC. This is due to EFDT's subtree pruning, which can lead to the removal of large parts of the decision tree. PLASTIC, on the other hand, restructures the branches and thus remains able to predict with comparably high accuracy. EFDT's subtree pruning occurs even on supposedly easy problems such as LED. Though the difference in terms of average accuracy between both approaches may only be small on some data streams (e.g., a mere 0.5 % on LED), large temporary drops in predictive performance are nonetheless problematic, for instance, in safety- or security-critical applications.

Our results on real-world data (Figure 4.4) are similar: our approach exhibits fewer and less extreme accuracy drops compared to EFDT on 10 out of 14 data streams, except on AIRLINES, HEPMASS, HIGGS and SUSY. This is surprising, as data streams with concept drift should favor algorithms that implement some forgetting mechanism—for instance, the subtree pruning in EFDT. It seems that, in reality, many concept drifts do not or only partially affect the decision boundary. In such cases, keeping information through restructuring remains advantageous, explaining PLASTIC's superior performance over EFDT. On AIRLINES, EFDT outperforms our approach, in particular at the beginning. Both approaches perform similarly on HEPMASS, SUSY, and HIGGS, although our approach seems to have a slight edge over EFDT.

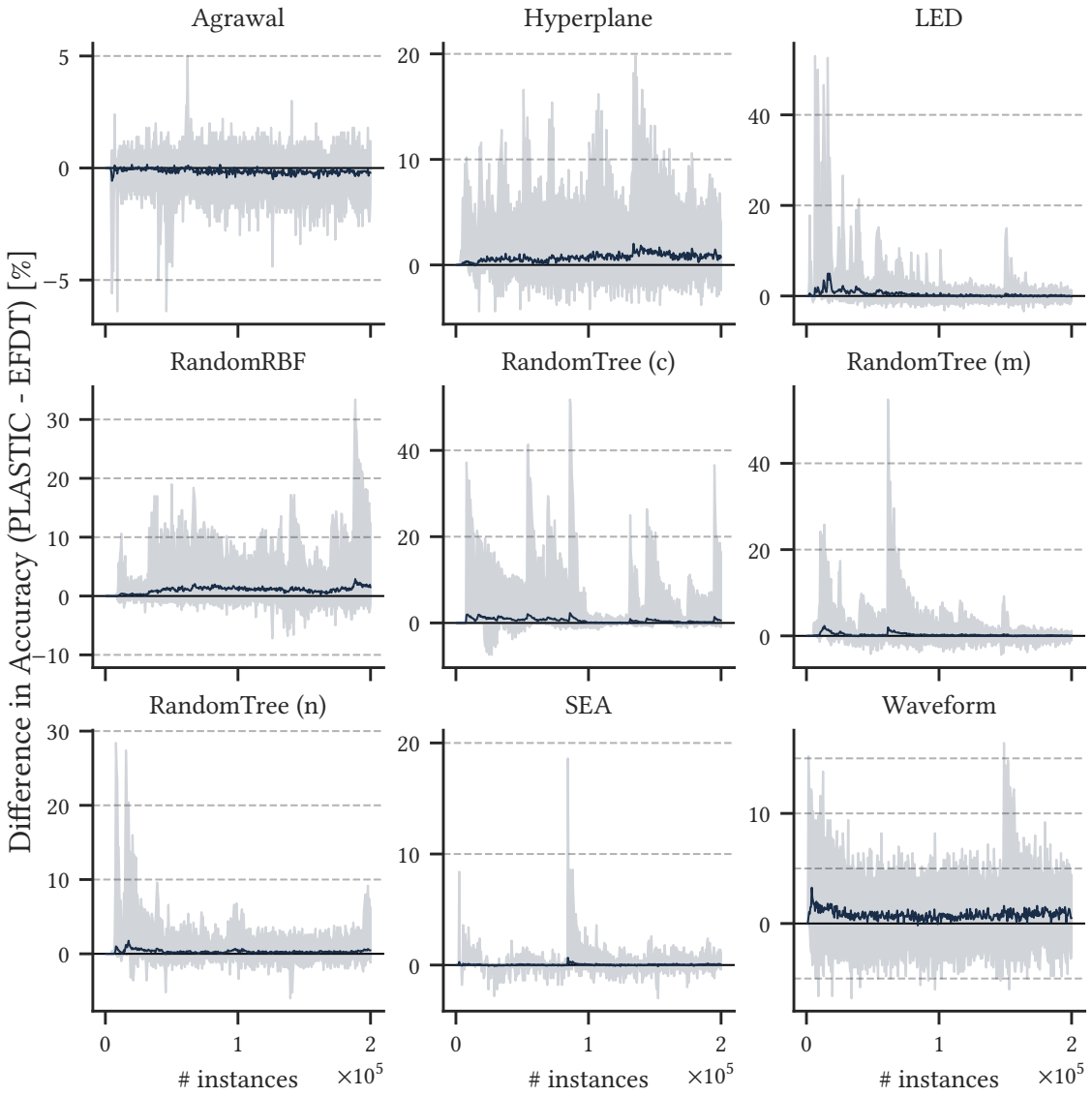


Figure 4.3.: Difference in accuracy between PLASTIC and EFDT on synthetic data; shaded area represents maximum and minimum across experiment repetitions

4.3.3.4. Average accuracy and ranks

Tables 4.2 and 4.3 show the average accuracy and ranks of all approaches on synthetic and real-world data, respectively. The last column, shows the accuracy of a no-change classifier (NC), which always predicts the last seen label. In terms of average accuracy, EFHAT and PLASTIC perform equally well on synthetic data (86.6 %). PLASTIC-A achieves 86.5 %, EFDT 86.1 %, and HT 79.7 %. EFHAT and PLASTIC also have similar ranks (2.1 and 2.2), followed by PLASTIC-A (2.6). The good performance of EFHAT on these data streams is in line with the findings in the original paper [MSW22], which already suggest that EFHAT performs well even on stationary data. HT's splitting mechanism seems too conservative, leading to slower learning and lower accuracy on average. EFDT performs resplitting too

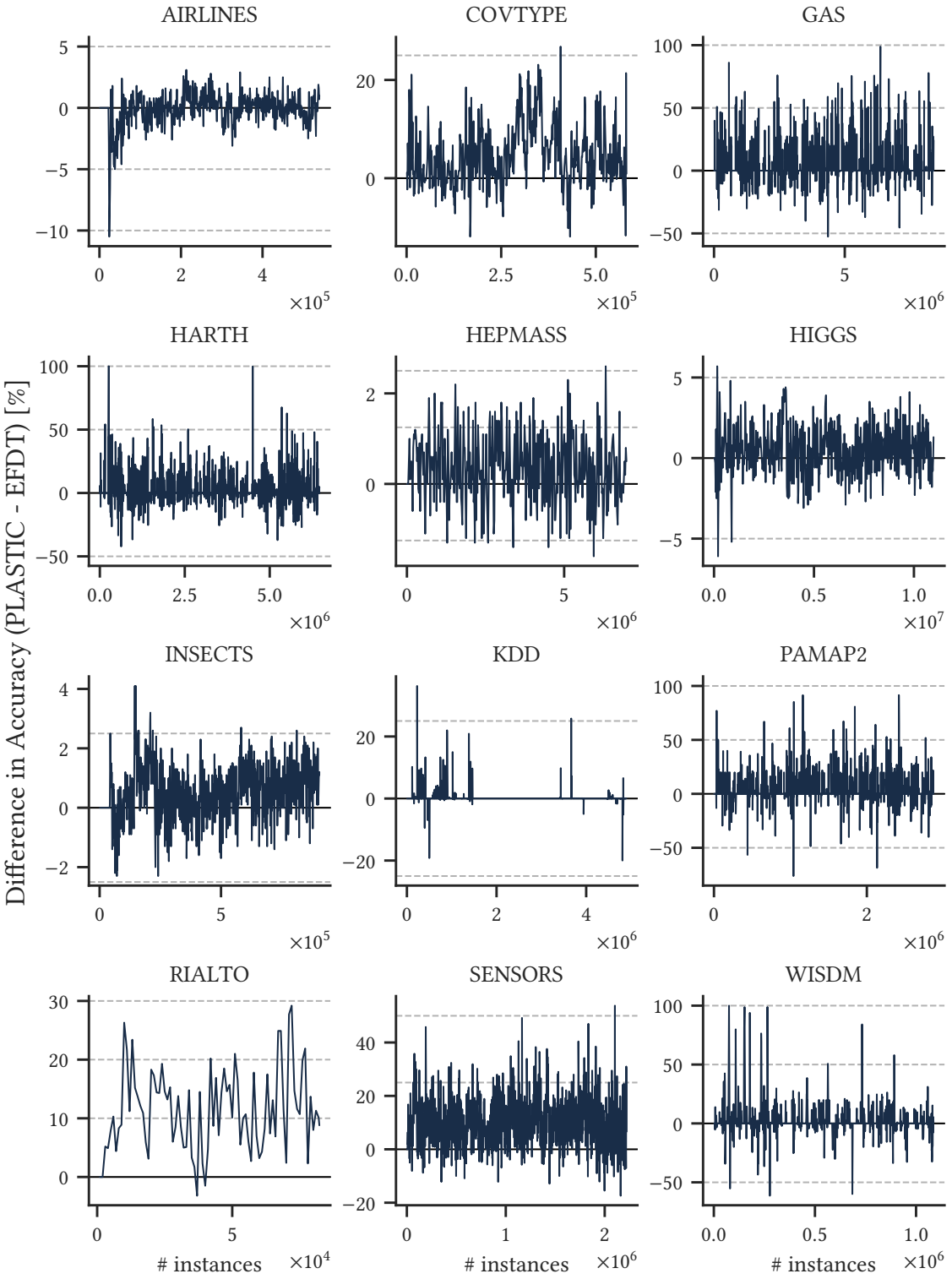


Figure 4.4.: Difference in accuracy between PLASTIC and EFDT on real-world data

frequently, resulting in sudden accuracy drops (cf. Figure 4.3). Last, we explain the slight difference between PLASTIC and PLASTIC-A with false alarms of the change detection mechanism. This can happen because of slight decreases in accuracy after restructuring (remember that we prune the restructured subtree at a depth of $\lambda = 5$).

Table 4.2.: Average accuracy on synthetic data

Approach	HT	EFDT	EFHAT	PLASTIC	PLASTIC-A
Agrawal	**94.7**	94.6	93.9	94.4	94.5
Hyperplane	76.4	76.9	77.7	77.6	**77.8**
LED	48.0	72.6	73.0	**73.1**	**73.1**
RandomRBF	83.5	84.9	**86.1**	85.9	85.0
RandomTree (c)	82.5	96.0	**96.5**	**96.5**	**96.5**
RandomTree (m)	87.4	94.8	**95.2**	95.1	95.1
RandomTree (n)	84.3	91.8	**92.3**	92.1	91.7
SEA	87.4	**88.0**	**88.0**	**88.0**	87.7
Waveform	72.7	75.8	76.3	76.6	**76.8**
Avg. accuracy	79.7	86.1	**86.6**	**86.6**	86.5
Standard dev.	9.89	5.55	5.40	5.35	**5.29**
Rank	4.6	3.6	**2.1**	2.2	2.6
Runtime [s]	**0.57**	1.63	1.51	1.85	2.60

On real-world data, PLASTIC-A performs best, beating its nonadaptive counterpart by 1.1 % and EFHAT by 1.4 % on average. PLASTIC outperforms EFDT by 2.3 %. Our approaches also achieve the best rank (2.29), followed by EFHAT (2.5), EFDT (3.71), and HT (4.21). The competitive performance of PLASTIC is surprising given that approaches with drift adaptation mechanisms should have a clear advantage. In particular, PLASTIC performs well on the environment monitoring data streams, where it beats EFHAT by up to 6.9 %. We assume that these streams contain recurring concept drift (e.g., due to day and night) and therefore favor algorithms that retain information.

Overall, we notice that change adaptation is particularly effective when subsequent observations mostly have the same label (see the data streams where the accuracy of a no-change classifier is 99.9 %). On such streams, a newly created background tree will perform similarly to the no-change classifier and hence quickly replace the current tree. Note, however, that this replacement also causes forgetting of the old concepts and leads to poor performance if the concepts reoccur.

4.3.3.5. Runtime

The last rows in Table 4.2 and Table 4.3 show the geometric average wall-clock time of all approaches. We use the geometric mean to ensure that the dataset with the highest

Table 4.3.: Average accuracy on real-world data

Approach	HT	EFDT	EFHAT	PLASTIC	PLASTIC-A	NC
RIALTO	24.2	37.8	42.3	**49.2**	47.4	0.0
SENSORS	15.8	38.2	42.7	**48.1**	47.1	0.1
COVTYPE	68.3	77.4	79.6	**82.1**	81.3	95.1
HARTH	79.5	86.5	89.2	88.3	**90.9**	99.9
PAMAP2	58.4	94.5	98.3	96.6	**98.6**	99.9
WISDM	65.6	80.6	89.0	82.6	**93.1**	99.9
HEPMASS	**85.3**	84.7	85.2	85.0	85.0	50.0
HIGGS	69.4	68.4	**70.0**	68.9	68.3	50.2
SUSY	**78.9**	78.7	**78.9**	78.8	78.8	50.4
AIRLINES	62.2	63.7	62.7	63.6	**64.2**	58.0
GAS	91.5	96.7	99.1	98.2	**99.2**	99.9
INSECTS	55.6	64.9	65.3	**65.4**	64.8	13.1
KDD	99.5	**99.8**	99.3	**99.8**	99.7	99.9
POKER	53.6	67.4	67.4	67.4	**70.7**	43.2
Avg. accuracy	64.8	74.2	76.4	76.7	**77.8**	61.4
Rank	4.21	3.71	2.50	**2.29**	**2.29**	–
Runtime [s]	**61.8**	110.6	198.6	141.6	175.0	27.1

runtime does not dominate the reported number. As expected, HT is the fastest approach as it does not perform split revision.

On synthetic data, EFHAT is faster than EFDT and PLASTIC, although the difference is relatively small. The additional runtime is due to EFDT's split revision and PLASTIC's restructuring. PLASTIC-A is the slowest approach on synthetic data. The approach not only performs restructuring, but also grows a background tree if it detected a change. Since the synthetic data streams are i.i.d. and stationary, it is harder for the background tree to outperform the current tree. Therefore, PLASTIC-A might maintain the current and the background tree simultaneously for an extended time period.

On real-world data, EFDT is about 25 % faster than PLASTIC. EFHAT, on the other hand, now takes almost twice as long as EFDT. We assume this is due to the large number of background trees that EFHAT maintains. EFHAT is also slower than PLASTIC-A, which maintains at most one background tree.

4.3.3.6. Evaluation on partial feedback

Table 4.4 mirrors Table 4.3, but shows the results obtained on a partially labeled data stream with only 10 % available feedback. As expected, the accuracy of all approaches (except the no-change classifier) decreases due to the decrease in feedback. We also

observe that runtime decreases as limited feedback causes fewer updates to the tree. Most notably, however, PLASTIC's and PLASTIC-A's average accuracy, rank, and runtime improve compared to their strongest contender, EFHAT. We attribute this improvement to the higher feedback efficiency of our approaches: EFHAT frequently trains multiple background trees, which demands more feedback. PLASTIC, in contrast, only maintains a single tree, and PLASTIC-A uses at most one background tree.

Table 4.4.: Average accuracy on real-world data with 10 % labels

Approach	HT	EFDT	EFHAT	PLASTIC	PLASTIC-A	NC
RIALTO	10.8	22.8	**22.9**	**22.9**	**22.9**	5.9
SENSORS	6.1	16.0	16.7	**20.6**	20.3	0.9
COVTYPE	57.6	66.8	68.0	69.1	**69.7**	77.4
HARTH	75.8	81.6	80.3	**83.5**	81.4	99.0
PAMAP2	40.8	79.0	85.6	86.4	**89.2**	99.9
WISDM	50.0	59.0	60.9	62.1	**64.0**	99.7
HEPMASS	**83.8**	82.5	81.1	83.4	83.6	50.0
HIGGS	**67.4**	64.9	63.3	66.3	64.6	50.2
SUSY	**77.6**	77.3	76.1	**77.6**	**77.6**	50.4
AIRLINES	58.5	**61.2**	58.6	60.8	58.7	55.2
GAS	88.9	90.7	90.5	**93.9**	92.8	99.9
INSECTS	39.0	60.5	61.3	**61.4**	61.3	13.0
KDD	99.2	**99.5**	98.2	**99.5**	99.0	99.9
POKER	50.1	51.6	51.6	51.6	**55.6**	43.2
Avg. accuracy	57.5	65.2	65.4	67.1	**67.2**	60.3
Rank	4.14	3.29	3.79	**1.68**	2.11	–
Runtime [s]	**27.1**	30.3	60.5	38.4	36.3	23.0

4.4. Summary

This chapter introduced PLASTIC, an incremental decision tree that leverages the concept of *decision tree plasticity* to achieve feedback efficiency in both full and observation-based partial feedback settings. It also presented a change-adaptive variant, PLASTIC-A, which grows a new decision tree in the background when the accuracy of the current tree begins to degrade. While both algorithms are feedback-efficient, they still rely on the availability of feedback for training. Thus, we now shift our focus to the extreme case of absent feedback and introduce an unsupervised change detector and descriptor for monitoring high-dimensional data streams.

5. Characterizing Change in High-Dimensional Data Streams

The content of this chapter is based on the following publication.

- Marco Heyden et al. "Adaptive Bernstein change detector for high-dimensional data streams". In: *Data Mining and Knowledge Discovery* 38.3 (2024), pp. 1334–1363. DOI: 10.1007/S10618-023-00999-5

5.1. Chapter Overview

Change detection is of fundamental importance when analyzing data streams. Feedback about changes enables monitoring systems to react, e.g., by issuing an alarm or updating a learning algorithm. However, detecting changes is challenging in high-dimensional data, as detectors should not only identify when changes happen (R1), but also in which subspace (R2), and how severe they are (R3).

We formalize the problem of detecting changes in high-dimensional data streams so that R1–R3 can be tackled in combination.

We present ABCD, a change detector for high-dimensional data that satisfies R1–R3. It monitors the loss of an encoder-decoder model using an adaptive window size and statistical testing based on Bernstein's inequality. Adaptive windows allow us to detect severe changes quickly and less severe ones over longer periods. Bernstein's inequality tightens the confidence bounds on the loss distribution by incorporating the first- and second-order moments.

We propose aggregates for monitoring the first- and second-order moments of a univariate data stream. These aggregates allow detecting changes in adaptive windows efficiently.

We conduct experiments on ten data streams using real-world and synthetic data. The results indicate that ABCD consistently outperforms its competitors w.r.t. R1–R3 and is robust to high-dimensional data.

We share the code of our implementation[1] and experiments[2].

[1] github.com/adaptive-machine-learning/CapyMOA/blob/main/src/capymoa/drift/detectors/abcd.py
[2] github.com/heymarco/AdaptiveBernsteinChangeDetector

5.2. Problem Definition

We are interested in finding changes in the last t observations $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t\}$ from a stream of data. Each $\mathbf{x}_i$ is a d-dimensional vector independently drawn from a (unknown) distribution $\mathcal{S}_i$. We assume without loss of generality that each vector coordinate is bounded in $[0, 1]$, i.e., $\mathbf{x}_i \in [0, 1]^d$.

Definition 5.1 (Change)**.** A change occurs at time point s if the data-generating distribution changes at s: $\mathcal{S}_{s-1} \neq \mathcal{S}_s$.

In high-dimensional data, changes typically affect only a subset of dimensions, which we call the *change subspace*. Let $D = \{1, 2, \ldots, d\}$ be the set of dimensions and $\mathcal{S}_t^{D'}$ be the distribution of $\mathcal{S}_t$ restricted to the subspace $D' \subseteq D$ at time step t. We define the change subspace as follows:

Definition 5.2 (Change subspace)**.** The change subspace D^* at time s is the union of all $D' \subseteq D$ in which the joint distribution $\mathcal{S}^{D'}$ changed and which does not contain a subspace D'' for which $\mathcal{S}_{s-1}^{D''} \neq \mathcal{S}_s^{D''}$.

If the dimensions in D^* are uncorrelated, then changes will be visible on the marginal distributions, i.e., all D' are of size 1. However, changes may only be detectable w.r.t the joint distribution of D^* or the union of its subspaces of size greater than 1, which our definition accommodates. Note that the definition can also handle multiple co-occurring changes and considers them as one single change. Last, change severity measures the difference between $\mathcal{S}_{s-1}^{D^*}$ and $\mathcal{S}_s^{D^*}$:

Definition 5.3 (Change severity)**.** The *severity* of a change is a positive function θ of the mismatch between $\mathcal{S}_{s-1}^{D^*}$ and $\mathcal{S}_s^{D^*}$.

Since we do not know the true distributions $\mathcal{S}_{s-1}$ and $\mathcal{S}_s$, the best we can do is detecting changes and their characteristics based on the observed data.

5.3. ABCD

5.3.1. Principle of ABCD

Directly comparing high-dimensional distributions is impractical as it requires many samples [Gre+12]. However, the number of variables required to describe such data with high accuracy is often much smaller than d [LV07]. Dimensionality reduction techniques let us *encode* observations in fewer dimensions. The more information encodings retain, the better one can reconstruct (*decode*) the original data. However, if the distribution changes, the reconstruction will typically degrade and produce higher errors.

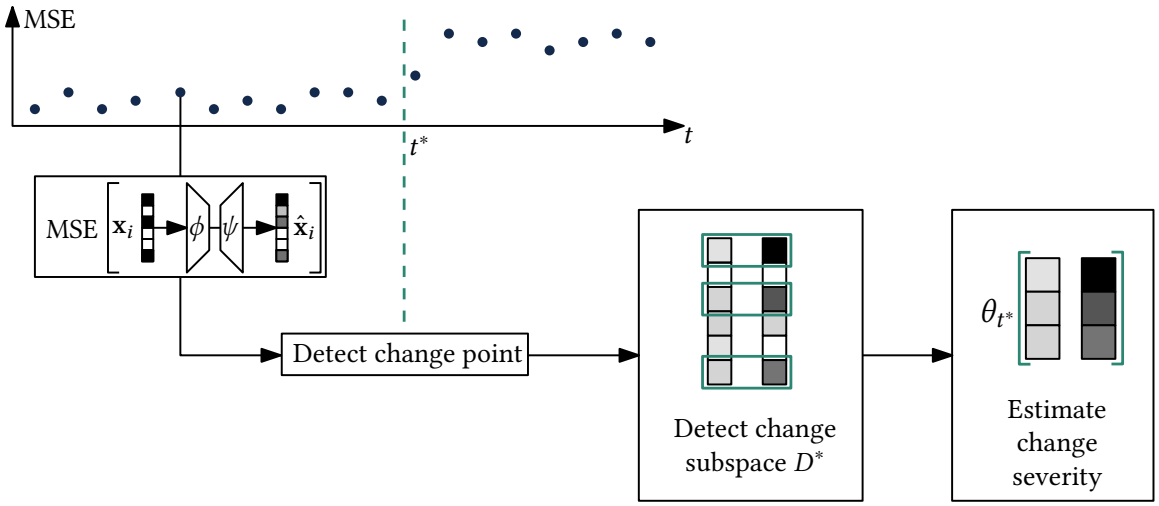


Figure 5.1.: Overview of ABCD

We leverage this principle in ABCD by monitoring the reconstruction loss of an encoder-decoder model $\psi \circ \phi$ for some encoder function ϕ and decoder function ψ. Figure 5.1 illustrates this. Specifically, we first learn $\phi : [0,1]^d \to [0,1]^{d'}$ with $d' = \lfloor \gamma d \rfloor < d$, $\gamma \in (1/d, 1)$, mapping the data to fewer dimensions, and $\psi : [0,1]^{d'} \to [0,1]^d$. Then, we monitor the loss between each $\mathbf{x}_i$ and its reconstruction $\hat{\mathbf{x}}_i = \psi \circ \phi(\mathbf{x}_i) = \psi(\phi(\mathbf{x}_i))$:

$$L_i = MSE(\mathbf{x}_i, \hat{\mathbf{x}}_i) = \frac{1}{d} \sum_{j=1}^{d} (\mathbf{x}_{ij} - \hat{\mathbf{x}}_{ij})^2 = \frac{1}{d} \sum_{j=1}^{d} L_{ij}$$

We hypothesize that distribution changes lead to outdated encoder-decoder models—see for example [JRA20] for empirical evidence. Hence, we assume that changes in the reconstruction affect the expected value μ_s of the loss at a change point s, because the model can no longer accurately reconstruct the input:

$$\mathcal{S}_{s-1} \neq \mathcal{S}_s \implies \mu_{s-1} \neq \mu_s$$

We can now replace the definition of change in high-dimensional data with an easier-to-evaluate, univariate proxy:

$$\exists s \in [1, \ldots, t] : \mu_{s-1} \neq \mu_s \tag{5.1}$$

It allows detecting arbitrary changes in the original (high-dimensional) distribution as long as they affect the average reconstruction loss of the encoder-decoder. Since the true μ_{s-1} and μ_s are unknown, we approximate them at different time points $\tau \leq t$:

$$\hat{\mu}_{1,\tau} = \frac{1}{\tau} \sum_{i=1}^{\tau} L_i, \quad \hat{\mu}_{\tau+1,t} = \frac{1}{t-\tau} \sum_{i=\tau+1}^{t} L_i.$$

5.3.2. Detecting the Change Point

ABCD detects a change whenever there exists a τ for which $\hat{\mu}_{1,\tau}$ differs *significantly* from $\hat{\mu}_{\tau,t}$. We refer to this time point as t^*. To quantify the significance, we derive a test based on Bernstein's inequality [Ber24]. It is often tighter than more general alternatives like Hoeffding's inequality [BLM13]. Let $\hat{\mu}_1, \hat{\mu}_2$ be the averages of two independent samples from two univariate random variables. One wants to evaluate whether both random variables have the same expected values: The null hypothesis H_0 is $\mu_1 = \mu_2$. Based on the two samples, one rejects H_0 if $\Pr\left(|\hat{\mu}_1 - \hat{\mu}_2| \geq \varepsilon\right) \leq \alpha$, where α is a preset significance level. The following theorem allows evaluating Eq. (5.1) using Bernstein's inequality.

Theorem 5.1 (Bound on $\Pr(|\hat{\mu}_1 - \hat{\mu}_2| \geq \varepsilon)$)**.** Given two independent samples $X_1^{(n_1)}, X_2^{(n_2)}$ of size n_1 and n_2 from two univariate, bounded random variables X_1, X_2 with unknown expected values μ_1, μ_2 and variances σ_1^2, σ_2^2. Let $\hat{\mu}_1, \hat{\mu}_2$ denote the sample means and let $|\mu_1 - x_i| < M$ for all $x_i \in X_1$ and $|\mu_2 - x_i| < M$ for all $x_i \in X_2$, respectively. Assuming $\mu_1 = \mu_2$, we have:

$$\Pr\left(|\hat{\mu}_1 - \hat{\mu}_2| \geq \varepsilon\right) \leq 2\exp\left\{-\frac{n_1(\kappa\varepsilon)^2}{2\left(\sigma_1^2 + \frac{1}{3}\kappa M\varepsilon\right)}\right\} + 2\exp\left\{-\frac{n_2((1-\kappa)\varepsilon)^2}{2\left(\sigma_2^2 + \frac{1}{3}(1-\kappa)M\varepsilon\right)}\right\} \in (0,4] \quad \forall\kappa \in [0,1]. \tag{5.2}$$

Proof. We follow the same steps as in [BG07; PSK14].

Recall Bernstein's inequality: Let $x_1, \ldots, x_n$ be independent random variables with sample mean $\hat{\mu} = 1/n \sum x_i$ and expected value μ s.t. $\forall x_i : |x_i - \mu| \leq M$. Then, for all $\varepsilon > 0$,

$$\Pr\left(|\hat{\mu} - \mu| \geq \varepsilon\right) \leq 2\exp\left\{-\frac{n\varepsilon^2}{2\left(\sigma^2 + \frac{1}{3}M\varepsilon\right)}\right\}.$$

We apply the union bound to $\Pr\left(|\hat{\mu}_1 - \hat{\mu}_2| \geq \varepsilon\right)$. For all $\kappa \in [0,1]$, we have:

$$\Pr\left(|\hat{\mu}_1 - \hat{\mu}_2| \geq \varepsilon\right) \leq \Pr\left(|\hat{\mu}_1 - \mu_1| \geq \kappa\varepsilon\right) + \Pr\left(|\hat{\mu}_2 - \mu_2| \geq (1-\kappa)\varepsilon\right)$$

Substituting the above with Bernstein's inequality completes the proof. □

With regard to change detection, one can use Eq. (5.2) to evaluate for a time point τ if a change occurred. The question is, however, how to choose ε to limit the probability of false alarm at any time t to a maximum α. Our approach is to set ε to the observed $|\hat{\mu}_{1,\tau} - \hat{\mu}_{\tau+1,t}|$ and to set $n_1 = \tau, n_2 = t - \tau$. The result bounds the probability of observing $|\hat{\mu}_{1,\tau} - \hat{\mu}_{\tau+1,t}|$ between two independent samples of sizes τ and $t - \tau$ under H_0. If this probability is very low, the distributions must have changed at τ. Then, we search for

changes at multiple time points τ in the current window. Hence, we obtain multiple such probability estimates; our change score is their minimum:

$$p = \min_{\tau} \Pr\left(|\hat{\mu}_1 - \hat{\mu}_2| \geq |\hat{\mu}_{1,\tau} - \hat{\mu}_{\tau+1,t}|\right) \tag{5.3}$$

The detected change point t^* splits $\{L_1, L_2, \ldots, L_t\}$ into the two subwindows with the statistically most different mean.

5.3.2.1. Choice of parameter κ

The bound in Eq. (5.2) holds for any $\kappa \in [0, 1]$. A good choice, however, provides a tighter estimate, resulting in faster change detection for a given rate of allowed false alarms α. [BG07] suggest to choose κ s.t. $Pr(|\hat{\mu}_1 - \mu_1| \geq \kappa\varepsilon) \approx Pr(|\hat{\mu}_2 - \mu_2| \geq (1-\kappa)\varepsilon)$, which approximately minimizes the upper bound. Substituting both sides with Bernstein's inequality, we get

$$\frac{n_1(\kappa\varepsilon)^2}{\sigma_1^2 + \frac{\kappa M\varepsilon}{3}} = \frac{n_2(1-\kappa)^2\varepsilon^2}{\sigma_2^2 + \frac{(1-\kappa)M\varepsilon}{3}}.$$

Setting $n_1 = rn_2$ and simplifying, we have

$$\frac{3\sigma_1^2 + \kappa M\varepsilon}{r\kappa^2} = \frac{3\sigma_2^2 + (1-\kappa)M\varepsilon}{(1-\kappa)^2}.$$

To solve for κ, note that $|\hat{\mu}_{1,\tau} - \hat{\mu}_{\tau+1,t}| \approx 0$ for large enough τ and $t - \tau$ if there is no change. This leads to a change score $p \gg \alpha$ for any choice of κ. Hence, choosing κ optimal is irrelevant while there is no change.

In contrast, if a change occurs, the change in the model's loss dominates the variance in both subwindows, leading to $M\varepsilon \gg \sigma_1^2, \sigma_2^2$. In that case, the influence of σ_1^2, σ_2^2 is negligible for sufficiently large κ and $1 - \kappa$:

$$\frac{\kappa M\varepsilon}{r\kappa^2} = \frac{(1-\kappa)M\varepsilon}{(1-\kappa)^2}. \tag{5.4}$$

Solving Eq. (5.4) for κ results in our recommendation for κ (Eq. (5.5)) which we restrict to $[\kappa_{min}, 1 - \kappa_{min}]$ with $\kappa_{min} = 0.05$.

$$\kappa = \frac{1}{1+r} = \frac{n_2}{n_1 + n_2} \tag{5.5}$$

5.3.2.2. Minimum sample sizes and outlier sensitivity

This section investigates the conditions under which ABCD detects changes.

We derive a minimum size of the first window above which ABCD detects a change. It bases on the fact that the number of observations before an evaluated time point τ remains

fixed while the number of observations after τ grows with t. Those counts are $n_1 = \tau$ and $n_2 = t - \tau$ in Eq. (5.2). Also, since we consider bounded random variables, their variance is also bounded. Hence, the second term in Eq. (5.2) approaches 0 for any $\varepsilon > 0$. With this, solving Eq. (5.2) for n_1 yields:

$$n_1 \geq \left\lceil 2 \log\left(\frac{2}{\alpha}\right)\left(\frac{\sigma_1^2}{(\kappa\varepsilon)^2} + \frac{M}{3\kappa\varepsilon}\right)\right\rceil$$

By setting $\varepsilon = |\hat{\mu}_1 - \hat{\mu}_2|$ we see that the required size of the first window decreases the larger the change in the average reconstruction error. For example, with $M = 1$, $\varepsilon = \sigma_1 = 0.1$, and $\alpha = 0.05$ our approach requires $n_1 \geq 32$.

Since ABCD detects changes in the average reconstruction loss of a bounded vector, it is stable with respect to outliers as long as they are reasonably rare. To see this, assume w.l.o.g. that window 1 contains n_{out} outliers and that $\varepsilon > 0$. One can show that the average of the outliers, $\hat{\mu}_{out}$, must exceed the average of the remaining inliers, $\hat{\mu}_{in}$, by $n_1\varepsilon/n_{out}$. In the example above, a single outlier would thus have to exceed $\hat{\mu}_{in}$ by a value of at least 3.2. This, however, is impossible because $M = 1$ bounds the reconstruction loss.

5.3.3. Detecting the Change Subspace

After detecting a change, we identify the change subspace. Restricting the encoding size to $d' < d$ forces the model to learn relationships between different input dimensions. As a result, the loss observed for dimension j contains not only information about the change in that dimension (i.e., the marginal distribution in j changes), but also about correlations influencing dimension j. Hence, we can detect changes in the marginal and joint distributions by evaluating in which dimensions the loss changed the most.

Algorithm 5.1 describes how we identify change subspaces. For each dimension j, we compute the average reconstruction loss (the squared error in dimension j) before and after t^*, denoted $\hat{\mu}^j_{1,t^*}, \hat{\mu}^j_{t^*+1,t}$ (lines 5 and 6), and the standard deviation $\hat{\sigma}^j_{1,t^*}, \hat{\sigma}^j_{t^*+1,t}$ (lines 6 and 7). We then evaluate Eq. (5.2), returning an upper bound on the p-value in the range $(0, 4]$ for dimension j (line 9). If $p_j < \zeta \in [0, 4]$, a hyperparameter for which we give a recommendation later on, we add j to the change subspace (lines 10 and 11).

5.3.4. Quantifying Change Severity

ABCD provides a measure of change severity in the affected subspace, based on the assumption that the loss in the change subspace increases with severity. Hence, we compute the average reconstruction loss observed in D^* before and after the change,

$$\hat{\mu}^{D^*}_{1,t^*} = \frac{1}{|D^*|(t^*)} \sum_{i=1}^{t^*} \sum_{j \in D^*} L_{ij}, \quad \hat{\mu}^{D^*}_{t^*+1,t} = \frac{1}{|D^*|(t-t^*)} \sum_{i=t^*+1}^{t} \sum_{j \in D^*} L_{ij}$$

Algorithm 5.1 Identification of change subspaces

Require: Window W, change point t^*, subspace thresholdζ

procedure SUBSPACE
  $D^* \leftarrow \emptyset$
  for all $j \in 1, \ldots, d$ **do**
    $\boldsymbol{\ell} \leftarrow \left[(\mathbf{x}_{ij} - \hat{\mathbf{x}}_{ij})^2\right]_{(\mathbf{x}_i, \hat{\mathbf{x}}_i) \in W}$
    $\hat{\mu}^j_{1,t^*} \leftarrow \frac{1}{t^*} \sum_{i=1}^{t^*} \boldsymbol{\ell}_i$
    $\hat{\mu}^j_{t^*+1,t} \leftarrow \frac{1}{t-t^*} \sum_{i=t^*+1}^{t} \boldsymbol{\ell}_i$
    $\hat{\sigma}^j_{1,t^*} \leftarrow \sqrt{\frac{1}{t^*-1} \sum_{i=1}^{t^*} \left(\boldsymbol{\ell}_i - \hat{\mu}^j_{1,t^*}\right)^2}$
    $\hat{\sigma}^j_{t^*+1,t} \leftarrow \sqrt{\frac{1}{t-t^*-1} \sum_{i=t^*+1}^{t} \left(\boldsymbol{\ell}_i - \hat{\mu}^j_{t^*,t}\right)^2}$
    $p_j \leftarrow$ Evaluate Eq. (5.2) ▷ Bernstein score
    if $p_j < \zeta$ **then**
      $D^* \leftarrow D^* \cup \{j\}$
  Return D^*

and the standard deviation observed before the change:

$$\hat{\sigma}^{D^*}_{1,t^*} = \sqrt{\frac{1}{t^*-1} \sum_{i=1}^{t^*} \left(\hat{\mu}^{D^*}_i - \hat{\mu}^{D^*}_{1,t^*}\right)^2} \quad \text{with} \quad \hat{\mu}^{D^*}_i = \frac{1}{|D^*|} \sum_{j \in D^*} L_{ij}$$

We then standard-normalize the average reconstruction loss $\hat{\mu}^{D^*}_{t^*,t}$ observed after the change and use it to measure change severity:

$$\theta_{t^*} = \frac{\left|\hat{\mu}^{D^*}_{t^*+1,t} - \hat{\mu}^{D^*}_{1,t^*}\right|}{\hat{\sigma}^{D^*}_{1,t^*}} \in \mathbb{R}^+$$

Intuitively, θ_{t^*} is the absolute change in average reconstruction error normalized to the standard deviation before the change.

5.3.5. Working with Adaptive Windows

In comparison to most approaches, ABCD evaluates multiple possible change points within an adaptive time interval $[1, \ldots, t]$. This frees the user from choosing the window size a-priori and allows to detect changes at variable time scales. Next, we discuss how to evaluate those time points efficiently.

5.3.5.1. Online monitoring of loss statistics

To avoid recomputing average reconstruction loss values and their variance for multiple time points every time a new observation arrives, we store aggregates $A_{1,\tau}$ summarizing

the stream in the interval $[1, \ldots, \tau]$. Each aggregate $A_{1,\tau}$ is a tuple containing the average reconstruction loss $\hat{\mu}_{1,\tau}$ and the sum of squared differences $ssd_{1,\tau} = \sum_{i=1}^{\tau}(L_i - \hat{\mu}_{1,\tau})^2$. We store these aggregates for the time interval $[1, \ldots, t]$.

Creating a new aggregate. Every time a new observation with loss L_t arrives, we create a new aggregate $A_{1,t}$ based on the previous aggregate $A_{1,t-1} = (\hat{\mu}_{1,t-1}, ssd_{1,t-1})$ in $\mathcal{O}(1)$ using Welford's algorithm [Knu97]:

$$\hat{\mu}_{1,t} = \hat{\mu}_{1,t-1} + \frac{1}{t}\left(L_t - \hat{\mu}_{1,t-1}\right)$$

$$ssd_{1,t} = ssd_{1,t-1} + \left(L_t - \hat{\mu}_{1,t-1}\right)\left(L_t - \hat{\mu}_{1,t}\right)$$

Computing the statistics. Two aggregates $A_{1,\tau}$ and $A_{1,t}$, $t \geq \tau$ overlap in the time interval $[1, \ldots, \tau]$. We leverage this overlap to derive an aggregate $A_{\tau,t} = (\hat{\mu}_{\tau,t}, ssd_{\tau,t})$ representing the time interval $[\tau, \ldots, t]$. Eq. (5.6) and Eq. (5.7) are based on Chan's method for combining variance estimates of non-overlapping samples [CGL82].

$$\hat{\mu}_{\tau+1,t} = \frac{1}{t-\tau}\left(t\hat{\mu}_{1,t} - \tau\hat{\mu}_{1,\tau}\right) \tag{5.6}$$

$$ssd_{\tau+1,t} = ssd_{1,t} - ssd_{1,\tau} - \frac{\tau(t-\tau)}{t}\left(\hat{\mu}_{1,\tau} - \hat{\mu}_{\tau+1,t}\right)^2 \tag{5.7}$$

From $ssd_{1,\tau}$ and $ssd_{\tau+1,t}$ we can compute the sample variances as follows:

$$\hat{\sigma}^2_{1,\tau} = \frac{ssd_{1,\tau}}{\tau - 1}, \quad \hat{\sigma}^2_{\tau+1,t} = \frac{ssd_{\tau+1,t}}{t - \tau - 1}$$

Proof. Let $X_A = \{x_1, \ldots, x_{n_A}\}$ and $X_B = \{x_1, \ldots, x_{n_B}\}$ be two non-overlapping samples of a real random variable. Let $\Sigma_A = \sum_{i=1}^{n_A} x_i$ and $\Sigma_B = \sum_{i=1}^{n_B} x_i$ be the sums of the samples and $ssd_A = \sum_{i=1}^{n_A}(x_i - n_A^{-1}\Sigma_A)^2$ and $ssd_B = \sum_{i=1}^{n_B}(x_i - n_B^{-1}\Sigma_B)^2$ be the sums of squared distances from the mean.

For the union of both sets $AB = A \cup B$ we have $\Sigma_{AB} = \Sigma_A + \Sigma_B$. Further, note that $\Sigma_A = n_A\hat{\mu}_A$, $\Sigma_B = n_B\hat{\mu}_B$, and $\Sigma_{AB} = n_{AB}\hat{\mu}_{AB} = (n_A + n_B)\hat{\mu}_{AB}$. Hence, the following equality holds:

$$(n_A + n_B)\hat{\mu}_{AB} = n_A\hat{\mu}_A + n_B\hat{\mu}_B$$

Solving for $\hat{\mu}_B$ gives

$$\hat{\mu}_B = \frac{n_A + n_B}{n_B}\hat{\mu}_{AB} - \frac{n_A}{n_B}\hat{\mu}_A.$$

Substituting $n_B = t - \tau$, $n_A = \tau$, $\hat{\mu}_A = \hat{\mu}_{1,\tau}$, $\hat{\mu}_B = \hat{\mu}_{\tau+1,t}$, and $\hat{\mu}_{AB} = \hat{\mu}_{1,t}$ gives Eq. (5.6).

Next, we derive Eq. (5.7). [CGL82] state:

$$ssd_{AB} = ssd_A + ssd_B + \frac{n_A}{n_B(n_A + n_B)} \left(\frac{n_B}{n_A}\Sigma_A - \Sigma_B \right)^2,$$

which is equivalent to

$$ssd_{AB} = ssd_A + ssd_B + \frac{n_A}{n_B(n_A + n_B)} \left(n_B \underbrace{\left(\frac{1}{n_A}\Sigma_A - \frac{1}{n_B}\Sigma_B \right)}_{=\hat{\mu}_A - \hat{\mu}_B} \right)^2.$$

Solving for ssd_B, applying the former substitutions, and setting $ssd_A = ssd_{1,\tau}$, $ssd_B = ssd_{\tau+1,t}$, and $ssd_{AB} = ssd_{1,t}$ results in Eq. (5.7). □

5.3.6. Implementation

Algorithm. One can implement ABCD as a recursive algorithm, see Eq. (5.2), which restarts every time a change occurs. We keep a data structure W that contains the aggregates, observations, and reconstructions. W can either be empty or, in the case of a recursive execution, already contain data from the previous run.

Prior to execution, our algorithm must first obtain a model of the current data from an initial sample of size n_{min}. If necessary, ABCD allows enough observations to arrive (lines 5–7). Larger choices of n_{min} allow for better approximations of the current distribution but delay change detection. Hence, we recommend setting n_{min} as small as possible to still learn the current distribution; a default of $n_{min} = 100$ has worked well for us.

Afterwards, the algorithm trains the model using the observations in W (lines 8–9). ABCD can in principle work with various encoder-decoder models; thus, we deal with tuning the model only on a high level. Nonetheless, we give recommendations in our sensitivity study later on.

After model training, ABCD detects changes. It reconstructs each new observation $\mathbf{x}_t$ (line 11), creates a new aggregate $A_{t_0,t}$ (line 12), and adds $w_t := (A_{t_0,t}, \hat{\mathbf{x}}_t, \mathbf{x}_t)$ to W (line 13). Our approach then computes the change score p and change point t^* (lines 14–15). If $p < \alpha$, it detects a change.

Once ABCD detects a change, it identifies the corresponding subspace and evaluates its severity (lines 17–18). Then, it adapts W by dropping the outdated part of the window (line 19), including all information obtained with the outdated model. At last, we restart ABCD with the adapted window (line 20).

Algorithm 5.2 Adaptive Bernstein Change Detector (ABCD)

Require: The model $\psi \circ \phi$, significance α, subspace threshold ζ

procedure ABCD(W)
  $\psi \circ \phi \leftarrow Null$ ▷ Model not yet trained
  $t_0 \leftarrow |W| + 1$
  while new observation $\mathbf{x}_t$ **do**
    if $t - t_0 < n_{min}$ **then** ▷ Warm up
      $w_t \leftarrow (-, -, \mathbf{x}_t)$
      $W \leftarrow W \mid\mid w_t$
    else if $\psi \circ \phi = Null$ **then**
      $\psi \circ \phi \leftarrow$ TrainModel(W)
    else
      $\hat{\mathbf{x}}_t \leftarrow \psi(\phi(\mathbf{x}_t))$ ▷ Reconstruct
      $A_{t_0,t} \leftarrow$ update aggregate $A_{t_0,t-1}$ with L_t
      $W \leftarrow W \mid\mid (A_{t_0,t}, \hat{\mathbf{x}}_t, \mathbf{x}_t)$
      $p \leftarrow$ Eq. (5.3) ▷ Bernstein score
      $t^* \leftarrow \arg\min_\tau$ of Eq. (5.3)
      if $p < \alpha$ **then** ▷ A change occurred
        $D^* \leftarrow$ Subspace(W, t^*, ζ)
        $\theta_{t^*} \leftarrow$ Severity(W, t^*, D^*)
        $W \leftarrow \{(-, -, x_i)\ \forall w_i \in W : i \geq t^*\}$
        ABCD(W) ▷ Restart

Discussion. In the worst case, our approach consumes linear time and memory as W grows linearly with t. However, we can simply restrict the size of W to n_{max} items for constant memory or evaluate only τ_{max} time points for constant runtime. In the latter case, we split W at every (t/τ_{max})-th time point. Regarding n_{max}, it is beneficial that the remaining aggregates still contain information about all observations in $[1, \ldots, t]$. Hence, ABCD considers the *entire* past since the last change despite restricting the size of W.

ABCD can work with any encoder-decoder model, such as deep neural networks. However, handling a high influx of new observations faster than the model's processing capability can be challenging. Assuming that $\psi \circ \phi \in \mathcal{O}(g(d))$ for some function g of dimensionality d, the processing time of a single observation during serial execution is in $\mathcal{O}\left(g(d) + \tau_{max}\right)$. Nevertheless, both the deep architecture components and the computation of the change score (cf. Equation 5.3) can be executed in parallel using specialized hardware.

Dimensionality reduction techniques are often already present in data stream mining pipelines, for example, as a preprocessing step to improve the accuracy of a classifier [Yan+06]. Reusing an existing dimensionality reduction model makes integrating ABCD into an existing pipeline easy.

Bernstein's inequality holds for zero-centered bounded random variables that take absolute values of at maximum M almost surely. While $M = 1$ serves as a theoretical upper limit of the zero-centered reconstruction error $L_i - \mathbb{E}[L_i]$ for $\mathbf{x}_i \in [0, 1]^d$, we observe that this

theoretical limit is very conservative in practice (cf. Figure 5.7). In fact, observing an error of 1 corresponds to an observation and reconstruction of $\mathbf{x} = [0]^d$ and $\hat{\mathbf{x}} = [1]^d$. This leads us to use $M = 0.1$ in our experiments.

5.3.7. Evaluation

This section describes our experiments and results. We first describe the experimental setting (Section 5.3.7.1). Then we analyze ABCD's change detection performance (Section 5.3.7.3), its ability to find change subspaces and quantify change severity (Section 5.3.7.4), and its parameter sensitivity (Section 5.3.7.5).

5.3.7.1. Algorithms

We evaluate ABCD with different encoder-decoder models:

1. PCA with $d' = \gamma d$.
2. Kernel-PCA (KPCA) with $d' = \gamma d$ and RBF-kernel.
3. A simple fully-connected autoencoder (AE) neural network with one hidden ReLU layer, $d' = \gamma d$, and an output layer with sigmoid activation function.

For (1) and (2), we rely on the default scikit-learn[3] implementations. We implement the autoencoder (3) in pytorch[4] and train it through gradient descent using E epochs and an Adam optimizer with default parameters according to [KB15]. Refer to Appendix A.3.1 for pseudocode of the autoencoder training procedure.

We compare ABCD with AdwinK, IKS, IBDD, WATCH, and D3 (c.f. Section 2.4). We evaluate for each approach a large grid of parameters, shown in Table 5.1. Whenever possible, the evaluated grids of hyperparameters for competitors base on recommendations in respective papers. Otherwise, we choose them based on preliminary experiments. For ABCD, we evaluate larger and smaller values for α, γ and E to observe our approach's sensitivity to those parameters. The choice of $\zeta = 2.5$ is our recommended default based on our sensitivity study in Section 5.3.7.5. Last, we set $n_{min} = 100$ and $\tau_{max} = 20$, minimum values that have worked well in preliminary experiments.

5.3.7.2. Datasets

There are not many public benchmark data streams for change detection. Thus, we generate our own from seven real-world (rw) and synthetic (syn) classification datasets, similar to [FDK19; Fab+21]. Unless stated otherwise, we simulate changing data streams by sorting the data by label. If the label changes, a change has occurred. In real-world

[3] scikit-learn.org
[4] pytorch.org

Table 5.1.: Evaluated change detectors and their hyperparameters

Algorithm	Parameter	Values
ABCD	model	PCA, Kernel-PCA (KPCA), Autoencoder (AE)
	α	0.05
	γ	$0, 3, 0.5, 0.7$
	$E^{\dagger}$	$20, 50, 100$
	$n_{min}; \tau_{max}; \zeta$	100; 20; 2.5
AdwinK	k	$0.01^{*}, 0.05^{*}, 0.1^{*}, 0.2^{*}, 0.3^{*}, 0.4^{*}, 0.5^{*}$
	α	0.05
D3	ω	$100^{*}, 250^{*}, 500^{*}$
	ρ	$0.1^{*}, 0.2^{*}, 0.3^{*}, 0.4^{*}, 0.5^{*}$
	τ	$0.6^{*}, 0.7^{*}, 0.8^{*}, 0.9^{*}$
	model	Logistic Regression*, Decision Tree
	tree depth	1, 3, 5
IBDD	ω	$100, 200, 300$
	m	$10, 20, 50, 100$
IKS	W	$100^{*}, 200, 500^{*}$
	α	0.05
WATCH‡	ω	$500, 1000$
	κ	100
	ϵ	$2, 3$
	μ	$1000, 2000$

* used or recommended in the respective papers
† only relevant for autoencoders
‡ authors did not recommend parameters for their approach

data streams, the number of observations between changes depends on each dataset, as reported below. In the synthetic streams, we introduce changes every 2000 observations, which is a relatively large interval, to assess whether some approaches generate many false alarms. The generators are based on the following datasets:

- **HAR (rw):** The dataset *Human Activity Recognition with Smartphones* [Ang+13] ($d = 561$) is based on smartphone accelerometer and gyroscope readings for different actions a person performs. A change occurs on average every 1768 observations.
- **GAS (rw):** This data set [Ver+11] ($d = 128$) contains data from 16 sensors exposed to 6 gases at various concentrations. A change occurs on average every 2265 observations.
- **LED (syn):** The LED generator [Bre+84] ($d = 23$) samples observations representing digits on a seven-segment display. It contains 17 additional random dimensions. We add changes by varying the probability of bit-flipping in the relevant dimensions.

- **RBF (syn):** The RBF generator [BHP10] starts by drawing a fixed number of centroids. For each new observation, the generator chooses a centroid at random and adds Gaussian noise. To create changes, we increment the seed of the generator, resulting in different centroids. We then use samples from the new generator in a subspace of random size.
- **MNIST, FMNIST, and CIFAR (syn):** Those data generators sample from the image recognition datasets MNIST [LCB10], Fashion MNIST (FMNIST) [XRV17] ($d = 784$), and CIFAR [Kri09] ($d = 1024$, grayscale).

Changes can occur rapidly ("abrupt" or "sudden") or in time intervals ("gradual" or "incremental"). The shorter the interval, the more sudden the change. We vary the interval size between 1 and 300 unless stated otherwise. Real-world and image data do not have a ground truth for change subspaces and severity. Thus, we generate three additional data streams:

- **HSphere (syn):** This generator drAW from a d^*-dimensional hypersphere bound to $[0, 1]$ and adds $d - d^*$ random dimensions. We vary the radius and center of the hypersphere to introduce changes. The change subspace contains those dimensions that define the hypersphere.
- **Normal-M/V (syn):** These generators sample from a d^*-dimensional normal distribution and add $d - d^*$ random dimensions. For type **M**, changes affect the distribution's mean, for **V**, we change the distribution's variance.

5.3.7.3. Change point detection

We use precision, recall, and F1-score to evaluate the performance of the approaches at detecting changes. We define true positives (TP), false positives (FP) and false negatives (FN) as follows:

- **TP:** A change was detected before the next change.
- **FN:** A change was not detected before the next change.
- **FP:** A change was detected, although no change occurred.

Also, we report the mean time until detection (MTD) indicating the average number of observations until a change is detected.

Figure 5.2 shows F1-score, precision, recall, and MTD for all datasets and algorithms, as well as a column 'Average', which summarizes across datasets. Each box contains the results for the grid of hyperparameters shown in Table 5.1. We see that our approach outperforms its competitors w.r.t. F1-score and precision. It is also competitive in terms of recall, though it loses against IKS, IBDD, and WATCH. These approaches seem overly sensitive. The results also indicate that ABCD works well for a wide range of hyperparameters. One reason is that ABCD uses adaptive windows, thereby eliminating the effect of a window size parameter (demonstrated in Section 5.3.7.7). Another reason is that ABCD detects

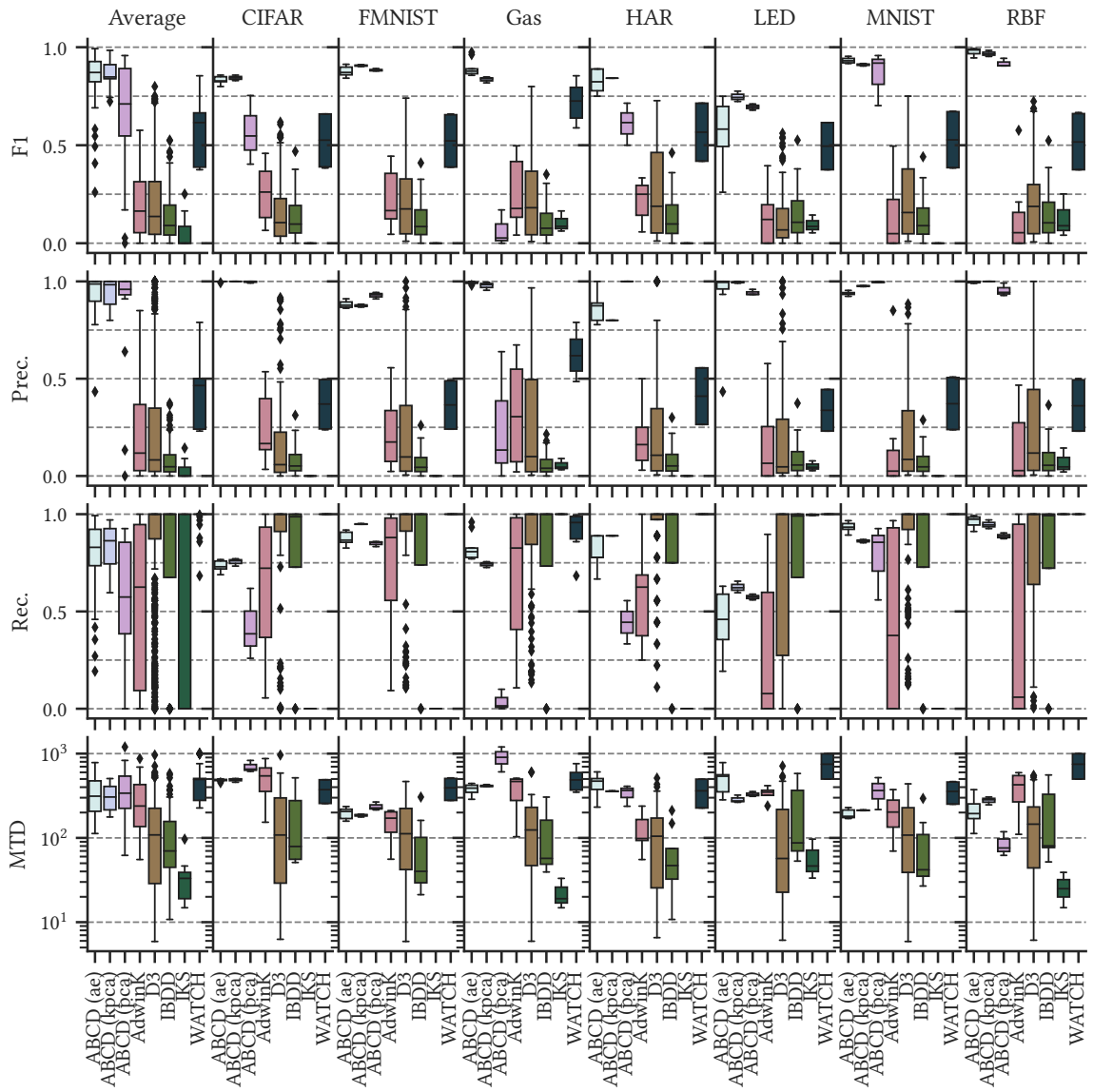


Figure 5.2.: Change point detection: Results for different algorithms and datasets; each box contains the results for the evaluated grid of parameters

changes in reconstruction loss irrespective of the actual quality of the reconstructions. For instance, Kernel-PCA and PCA produce reconstructions of different accuracy in our experiments. However, for both models, the average accuracy changes when the concept changes, which is what our algorithm detects. Refer to Figure 5.7 for an illustration of the models' reconstruction loss over time. Note that our reported results do not yield information about the actual accuracy of the underlying encoder-decoder models.

ABCD has a higher MTD than D3, IBDD, and IKS, i.e., it requires more data to detect changes. However, those competitors are much less conservative and detect many more changes than exist in the data. Hence, they have low precision but high recall—this leads to a lower MTD.

Table 5.2 reports the results of all approaches with their best hyperparameters w.r.t. F1-score. WATCH and D3 achieve relatively high F1-score and precision. In fact, those approaches are our strongest competitors, although we still outperform them by at least 3 %. Further, WATCH has an MTD of 626, which is more than ABCD, while D3 and ABCD have a comparable MTD.

ABCD has much higher precision than its competitors. We assume this is because it (1) leverages the relationships between dimensions, in comparison to AdwinK, IKS, and IBDD, and (2) learns those relationships more effectively than, say, D3 and WATCH. For example, we observed in our experiments that WATCH was frequently unable to approximate the Wasserstein distance in high-dimensional data accurately.

ABCD has lower recall than most competitors, partly due to their over-sensitivity. In this regard, our approach might benefit from application-specific encoder-decoder models that leverage structure in the data, such as spatial relationships between the pixels of an image, more effectively.

Table 5.2.: Average change detection results with optimal hyperparameters

Approach	F1	Prec.	Rec.	MTD
ABCD (ae)	0.90	0.96	0.87	250
ABCD (kpca)	0.88	0.95	0.84	312
ABCD (pca)	0.73	0.93	0.65	442
AdwinK	0.46	0.48	0.57	400
D3	0.70	0.63	0.82	251
IBDD	0.45	0.30	0.97	396
IKS	0.08	0.04	0.43	24
WATCH	0.69	0.54	1.00	626

5.3.7.4. Change subspace and severity

We now evaluate the identification of change subspaces and the estimation of change severity. We set $d = \{24, 100, 500\}$ and vary the change subspace size d^* randomly in $[1, d]$ (except for LED, here the subspace always contains dimensions 1–7). We set the ground truth for the severity to the absolute difference between the parameters that define the concepts, e.g., the hypersphere-radius in HAR before and after the change. We report an approach's subspace detection accuracy (SAcc.), where true positives (true negatives) represent those dimensions that were correctly classified as being members (not being members) of the change subspace. We use Spearman's correlation between the detected severity and the ground truth. We also report the F1-score for detecting change points.

Figure 5.3 shows our results. As before, each box summarizes the results for the evaluated hyperparameter grid. Comparing the two approaches that monitor each dimension separately, AdwinK and IKS, we see that the former can only detect changes that affect the

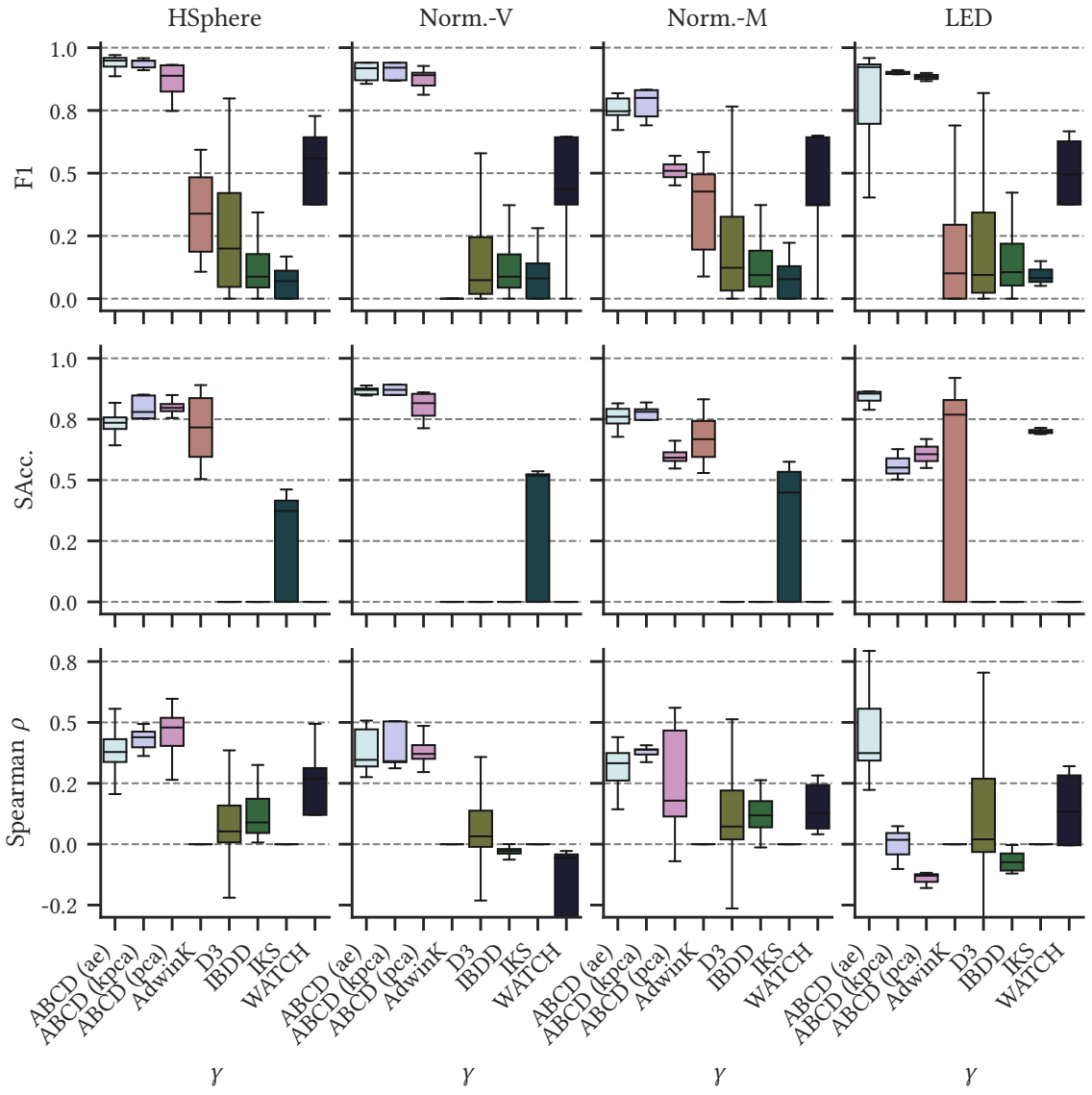


Figure 5.3.: Results for evaluating change subspace and severity

mean of the marginal distributions (i.e., on Norm-M, LED). At the same time, the latter can also detect other changes (e.g., changes in variance). This is expected since AdwinK compares the mean in two windows while IKS compares the empirical distributions.

Regarding subspace detection, our approach achieves an accuracy of 72 % for PCA, 78 % for autoencoders, and 79 % for Kernel-PCA. AdwinK performs similarly well when changes affect the mean of the marginal distributions. Except on LED, IKS performs worse than ABCD and AdwinK, presumably because IKS issues an alarm as soon as a single dimension changes.

The estimates of our approach correlate more strongly with the ground truth than those of competitors, with an average of 31 % for PCA, 36 % for Kernel-PCA, and 37 % for Autoencoders. However, we expect more specialized models to perform better than our

tested models. On LED, PCA-based models seem to struggle to separate patterns from noise, resulting in poor noise level estimates and low correlation scores.

5.3.7.5. Parameter sensitivity of ABCD

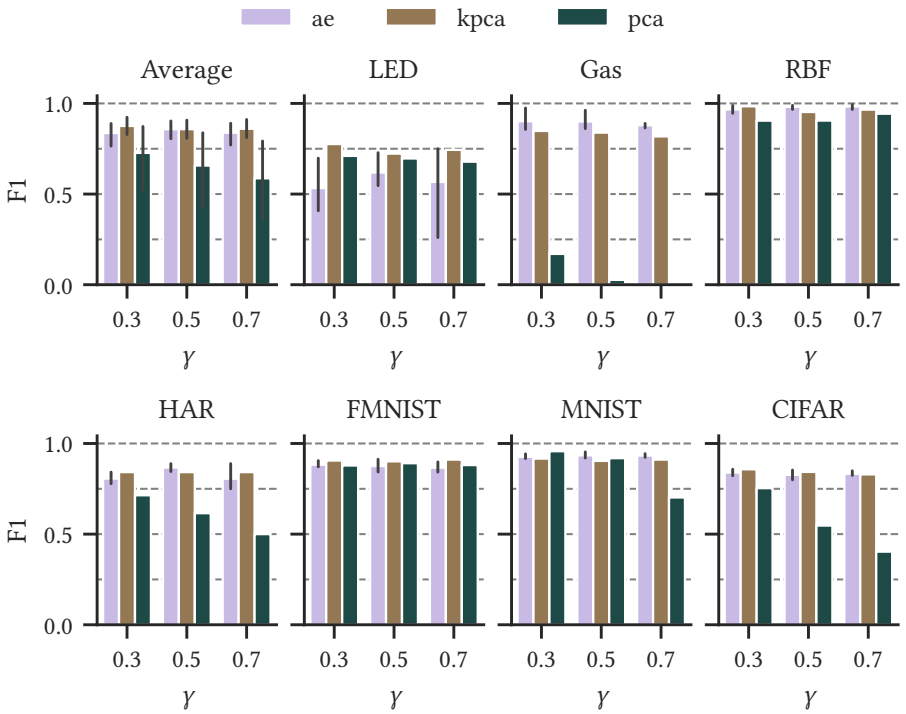
Sensitivity to γ. Figure 5.4a plots F1 for different datasets over γ. We observe that the size of the bottleneck does not significantly impact the change detection performance of ABCD (ae) and ABCD (kpca). For PCA, however, too large bottlenecks seem to inhibit change detection on CIFAR, Gas, and MNIST. For those datasets, we assume that the change occurs along the retained main components, rendering it undetectable; see Appendix A.3.2 for an illustration. Figure 5.4b shows the subspace detection accuracy and Spearman's ρ. The influence of γ on both metrics is low. As mentioned earlier, we assume that a *change* in reconstruction loss, rather than the quality of reconstruction itself, is crucial for ABCD. An exception is the LED dataset, on which PCA and Kernel-PCA are unable to provide a measure that positively correlates with change severity. As discussed before, we hypothesize that these methods face difficulties in distinguishing patterns from noise, leading to inaccurate noise level estimates and reduced correlation scores.

Sensitivity to E. Figure 5.5 plots the performance of ABCD (ae) for different choices of E. Overall, our approach seems to be robust to the choice of E. On LED, however, larger choices of E lead to substantial improvements in F1-score. The reason may be that the autoencoder does not converge to a proper representation of the data for small E. To avoid this, we recommend choosing $E \geq 50$ and increasing the value if one observes that the model has not yet converged sufficiently.

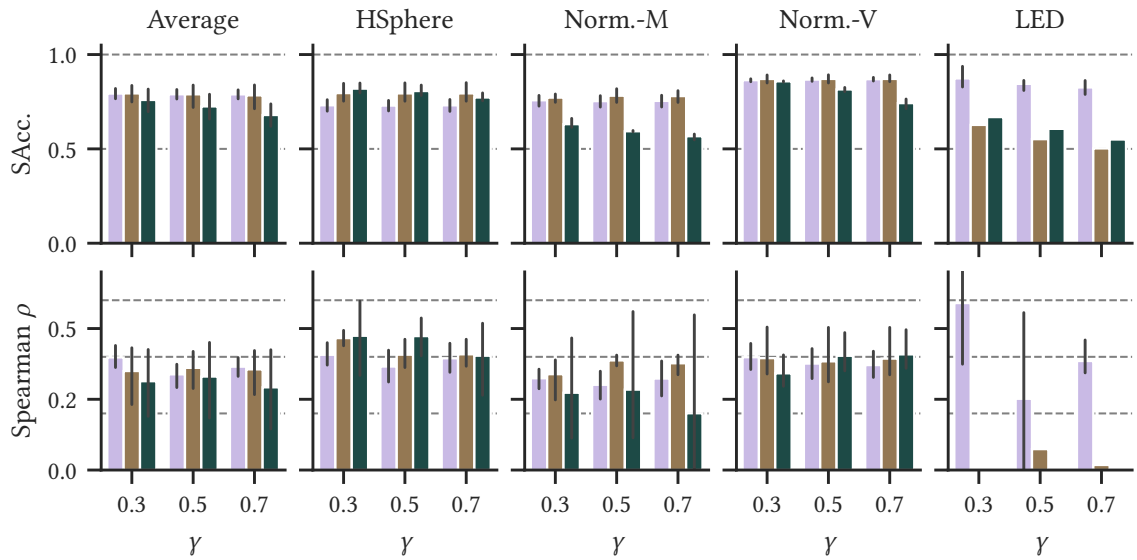
Sensitivity to ζ. Figure 5.6 investigates how the choice of ζ affects the performance of ABCD at detecting subspaces. Since the change score in Eq. (5.2) provides an upper bound on the probability that a change occurred, the function can return values greater than 1, i.e., in the range $(0, 4]$. Hence, we vary ζ in that range and record the obtained subspace detection accuracy. For all approaches, we achieve the best accuracy at $\zeta \approx 2.5$. This is probably because some dimensions could change more severely than others, resulting in variations of the change scores observed in the different dimensions of the change subspace. Based on our findings, we recommend $\zeta = 2.5$ as default.

5.3.7.6. Reconstruction loss over time

Figure 5.7 shows the reconstruction loss of the evaluated encoder-decoder models over the length of the stream. We observe that, indeed, the reconstruction loss decreases with increasing bottleneck size (controlled by γ) and with increasing number of training epochs E (first three columns). Further, we see that regardless of E, γ, and the type of model, the reconstruction loss typically changes after a change point. After the change was detected, ABCD learns the new concept, which mostly leads to a decreased reconstruction loss. Last,



(a) Influence of γ on the identification of changes



(b) Influence of γ on the estimation of change subspaces and severity

Figure 5.4.: Sensitivity of ABCD w.r.t. γ

we observe that the theoretical limit of $M = 1$ for the absolute difference between the reconstruction loss and its expected value is overly conservative. A value of $M = 0.1$ seems to be a more realistic choice.

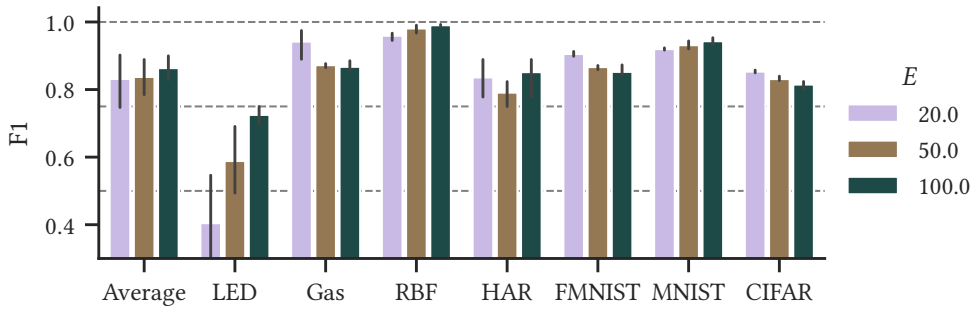


Figure 5.5.: Change detection performance of ABCD (ae) depending on E

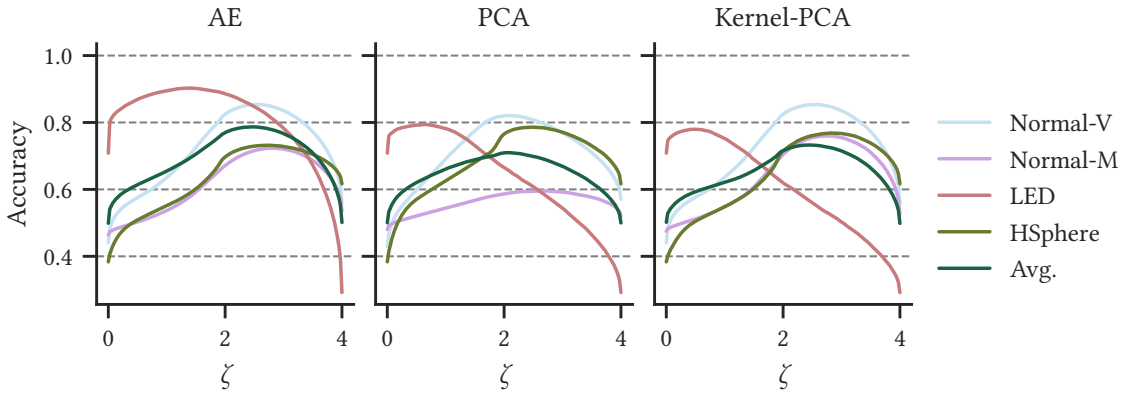


Figure 5.6.: Subspace detection accuracy of ABCD depending on ζ

5.3.7.7. Ablation study on window types

We now investigate the effect of different window types on change detection performance. We evaluate those commonly found in change detection literature (and in our competitors) and couple them with encoder-decoder models and the bound in Eq. (5.2). In particular, we compare: (1) Adaptive windows (AW), as in ADWIN, AdwinK, and our approach, (2) fixed reference windows (RW), as in IKS, (3) sliding windows (SW), as in WATCH, and (4) jumping windows (JW), as in D3. The latter "jump" every $\rho|W|$ observations.

We evaluate the hyperparameters mentioned in Table 5.1. For example, because D3 uses jumping windows, we include the evaluated hyperparameters for D3 in our evaluation of jumping windows. In addition, we extend the grid with other reasonable choices since we already preselected those in Table 5.1 for our competitors in a preliminary study. For ABCD we use $\gamma = 0.5$ and $E = 50$.

Table 5.3 reports the average over all hyperparameter combinations. AW yield higher F1-score and recall than other techniques, while precision remains high (≥ 0.95). SW have a lower MTD than AW and hence seem to require fewer observations until they detect a change. This is expected: in contrast to sliding windows, adaptive windows allow the detection of even slight changes after a longer period of time, resulting in both higher MTD and recall.

Table 5.3.: Ablation study – Using encoder-decoder models with different window types

Model	Window	F1	Prec.	Rec.	MTD
AE	AW	**0.83**	0.95	**0.78**	455.6
	RW	0.53	**1.00**	0.21	403.6
	SW	0.62	**1.00**	0.40	**207.2**
	JW	0.52	0.79	0.46	239.1
KPCA	AW	**0.83**	0.99	**0.75**	309.0
	RW	0.56	**1.00**	0.23	456.3
	SW	0.68	**1.00**	0.49	**202.8**
	JW	0.50	0.77	0.33	266.2
PCA	AW	**0.72**	0.98	**0.55**	355.3
	RW	0.36	**1.00**	0.09	400.0
	SW	0.53	**1.00**	0.33	**206.7**
	JW	0.46	0.75	0.20	239.9

5.3.7.8. Runtime analysis

Comparison with competitors. Figure 5.8a shows the mean time per observation (MTPO) of ABCD and its competitors for $d \in \{10, 100, 1000, 10\,000\}$ running single-threaded. The results are averaged over all evaluated hyperparameters (Table 5.1). ABCD (id) replaces the encoder-decoder model with the identity which does not cause overhead. This allows measuring how much the encoder-decoder model influences ABCD's runtime. The results confirm that the runtime of ABCD alone, i.e., without the encoding-decoding process, remains unaffected by a stream's dimensionality.

We observe that our approach is able to process around 10,000 observations per second for $d \leq 100$. This is more than IKS, WATCH and AdwinK (except at $d = 10$) but slower than D3 and IBDD. The reason is that our approach evaluates τ_{max} possible change points in each time step. In high-dimensional data, our competitors' MTPO grows faster than ABCD with PCA or KPCA; in fact, ABCD (pca) is the second fastest after D3 for $d \geq 1000$. An exception is WATCH at $d = 10\,000$. This is due to an iteration cap for approximating the Wasserstein distance restricting the approach's MTPO.

Runtime depending on window size. Next, we investigate ABCD's runtime for different choices of τ_{max} and γ. We run this experiment on a single CPU thread. For all three evaluated models, the encoding-decoding process for one observation has a time complexity of $\mathcal{O}(\gamma d^2)$. Hence, ABCD's processing time of one observation is in $\mathcal{O}(\gamma d^2 + \tau_{max})$. We therefore expect a quadratic increase in execution time with dimensionality and a linear increase with γ and τ_{max} when running on a single core.

The results in Figure 5.8b show the influence of τ_{max} on the execution time: τ_{max} effectively restricts the MTPO as soon as $|W| = \tau_{max}$. Afterwards, MTPO remains unaffected by $|W|$.

This also confirms that one can evaluate multiple possible change points in constant time using the proposed aggregates.

We show the runtime for different choices of bottleneck-size γ in Figure 5.8c. γ has little influence on the runtime of ABCD with PCA and Kernel-PCA . However, coupled with an autoencoder (implemented in pytorch) we observe the expected linear increase in execution time from 0.1 ms for $\gamma = 0.3$ to 0.3 ms for $\gamma = 0.7$. Considering that change detection performance has shown to remain stable even for smaller choices of γ, we recommend $\gamma \leq 0.5$ as default.

5.4. Summary

This chapter presented the Adaptive Bernstein Change Detector (ABCD), an unsupervised change detector and descriptor for high-dimensional data streams. ABCD measures the information loss during dimensionality reduction and triggers an alarm when a significant change occurs. After detecting a change, it identifies the affected dimensions and quantifies the change's severity within them. The algorithm is agnostic to the choice of dimensionality reduction technique, offering flexibility across various applications and data types.

ABCD complements our previous contributions, ω-UCB, and PLASTIC, by enabling decision-makers to monitor data streams without receiving feedback from the environment. To further enhance decision-making, future research may integrate ABCD with these algorithms, as the next chapter explores in more detail.

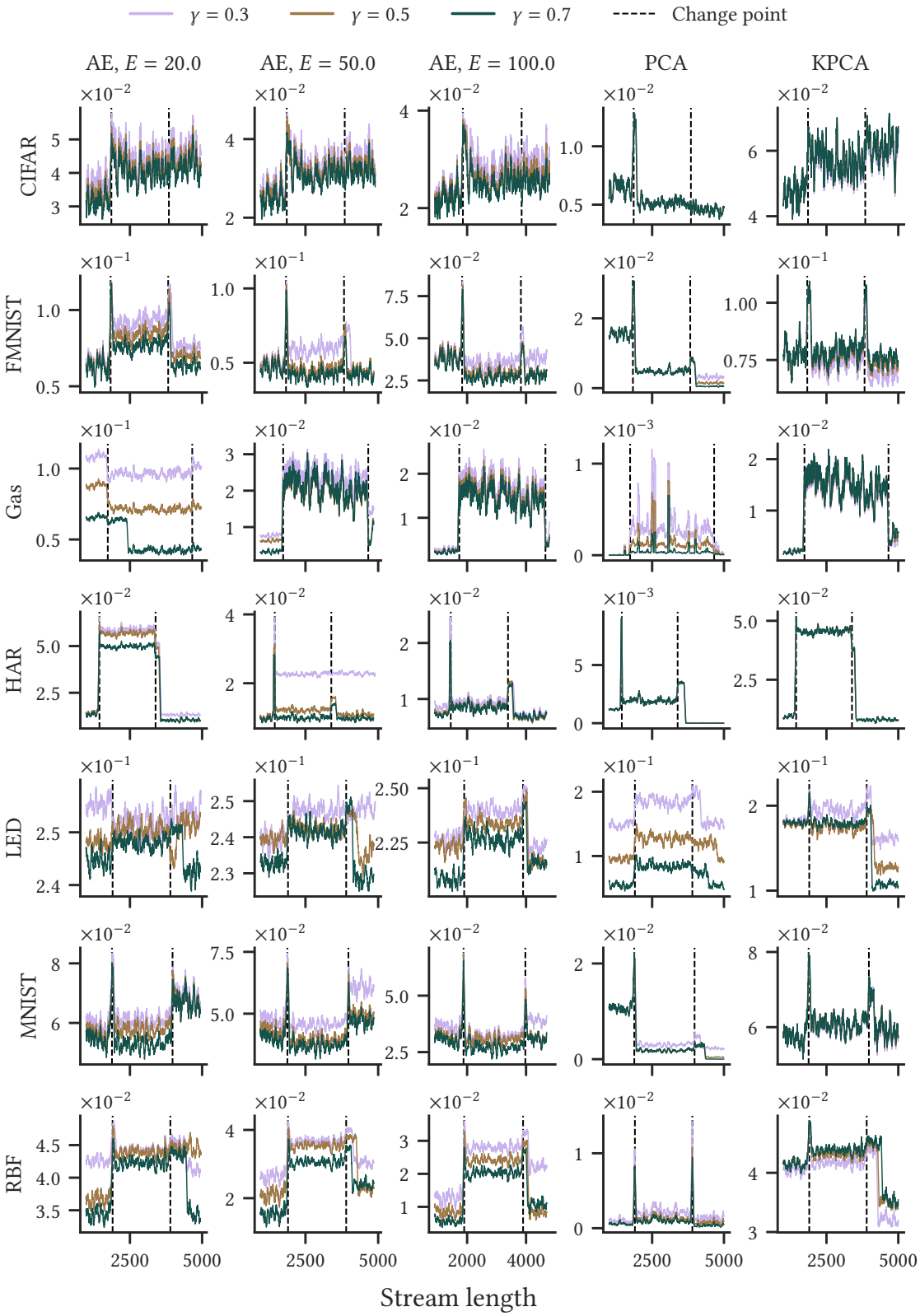
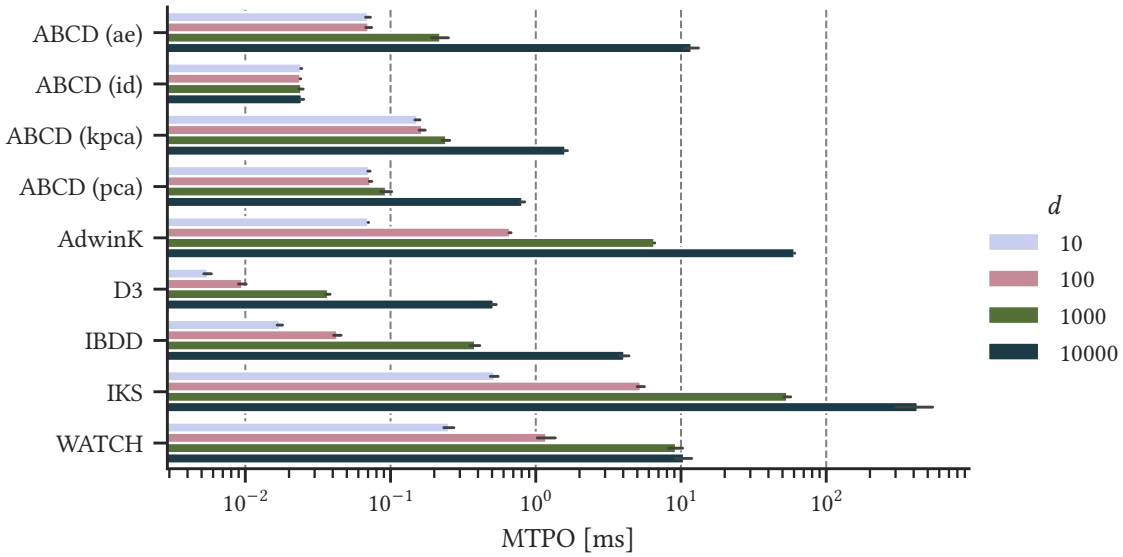
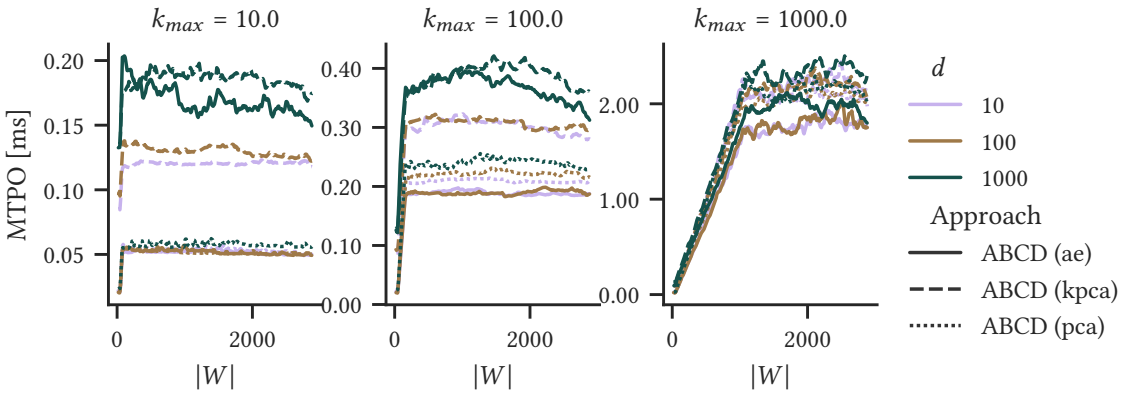


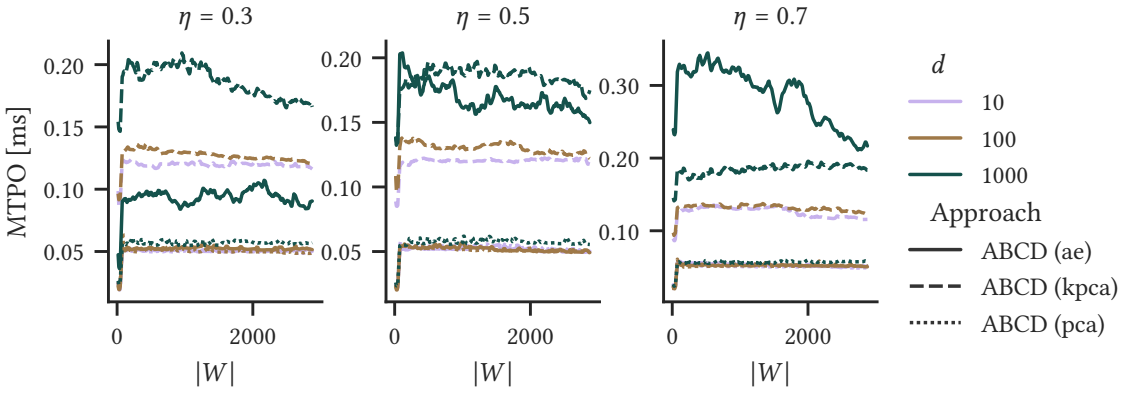
Figure 5.7.: Reconstruction loss over the length of the stream



(a) Mean time per observation in milliseconds



(b) MTPO of ABCD over $|W|$ using $E = 50$ and $\gamma = 0.5$ varying τ_{max}



(c) MTPO of ABCD over $|W|$ using $E = 50$ and $\tau_{max} = 100$ varying γ

Figure 5.8.: Runtime analysis of ABCD

Part III.

Conclusion and Outlook

6. Conclusion

Decision-making is a fundamental task for any intelligent agent. However, it is challenging due to the complexity and changing nature of the environment, as well as the uncertainty and randomness in the outcome of different decisions. Decision-makers thus leverage decision support systems to guide their decisions.

ML algorithms for data streams (MLDS) are well suited as decision support systems, as they are typically able to learn incrementally, adapt to changes in the environment, and provide feedback in near-real-time. However, most existing MLDS algorithms require plenty and cheap feedback for training, which makes them impractical for important applications (see Examples 1.1, 1.2, and 1.3). Our research questions address these shortcomings from three complementary perspectives: how to design decision support systems that are effective under (**Q1**) *decision-based partial feedback*, (**Q2**) *observation-based partial feedback*, and (**Q3**) *absence of feedback*.

In Chapter 3, we tackled the challenge of learning from decision-based partial feedback when the decisions have (unknown and random) costs associated with them (**Q1**). We explained and illustrated that—although there already exists a variety of algorithms that address this—existing algorithms struggle to balance the tradeoff between exploring the outcomes of different decisions and exploiting the collected knowledge. We then developed a novel UCB algorithm, called ω-UCB, which balances this tradeoff effectively by leveraging asymmetric confidence intervals. Our theoretical analysis and experimental evaluation indicate that ω-UCB not only has strong theoretical guarantees but also superior empirical performance. In particular, we illustrated on social-media advertisement data that one can use our algorithm to optimize marketing campaigns effectively.

We then studied one of the most widely used algorithms for learning from full and observation-based partial feedback in data streams: incremental decision trees (**Q2**, Chapter 4). While decision trees are beneficial as decision support systems (due to their interpretability, robustness to outliers, and ability to handle different data types), common algorithms are either feedback-inefficient (HT) or unstable w.r.t. their predictive performance (EFDT). We addressed this with PLASTIC, an evolution of EFDT that improves learning stability by restructuring the otherwise pruned subtrees. Our algorithm leverages the concept of decision tree plasticity that allows changing the structure of a decision tree without affecting its predictions. Our experiments show that PLASTIC boosts EFDT's worst-case accuracy and also outperforms the current state of the art incremental decision trees in terms of average accuracy on synthetic and real-world data streams.

Finally, in Chapter 5, we addressed scenarios in which the environment provides no feedback to the decision-maker (**Q3**), a common challenge in complex, high-dimensional

environments. Here, unsupervised algorithms can provide the decision-maker with surrogate feedback, including information about whether the process generating the data has changed. Detecting changes in high-dimensional data is particularly difficult as they may only affect a subset of attributes or their correlations. To address this, we proposed ABCD, an unsupervised algorithm for detecting and characterizing changes in high-dimensional data streams. The approach uses dimensionality reduction to encode observations in fewer dimensions with minimal information loss. An increase in information loss indicates that the data distribution has changed. ABCD identifies not only when a change occurs but also which attributes are affected and how severely. Our evaluation on image and sensor data demonstrates that ABCD detects changes effectively. Experiments on synthetic data show that ABCD characterizes the change subspace accurately and provides a severity measure that aligns more strongly with the ground truth than that of existing algorithms.

In conclusion, this dissertation presents fundamental contributions to the field of machine learning for data streams in the context of decision-making. The contributions focus on different aspects of learning from limited feedback and were published in renowned peer-reviewed conferences and journals. We presented ω-UCB [Hey+24b] at the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'24), and PLASTIC [Hey+24c] at the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2024). We published ABCD [Hey+24a] in the 'Data Mining and Knowledge Discovery' (DAMI) journal (Springer) as part of the 'ECML PKDD 2023' special issue.

Our contributions raised several interesting directions for future research. In particular, how to incorporate sophisticated change adaptation mechanisms in our algorithms and how to tailor them to other practically relevant but so far underexplored scenarios.

7. Outlook

We outline the following open research directions:

Contextual budgeted multi-armed bandits. Real-world environments often provide additional context, such as customers' age or interests, to guide decision-making. However, incorporating context into BMAB has received limited attention [Xia+15a], despite extensive research in related resource-constrained settings [BLS14; AD16; ZY24]. Given the theoretical and empirical effectiveness of ω-UCB in the classical BMAB setting, extending it to contextual scenarios would be a promising direction. In doing so, one may also investigate ways to tighten the confidence interval on the ratio between rewards and costs by exploiting their correlation. This could ultimately lead to lower regret, both theoretically and empirically.

Adapting ω-UCB to non-stationary environments. The current formulation of ω-UCB assumes that rewards and costs are i.i.d. samples from two unknown distributions. While this assumption may hold for small budgets (and consequently short time horizons), it becomes less realistic when the budget far exceeds the average decision cost. In such cases, we can expect the reward and cost distributions to change unexpectedly—e.g., due to a sudden increase in a product's popularity in online marketing. The challenge lies in the limited feedback: the decision-maker observes outcomes only for chosen decisions, making changes hard to detect without sufficient exploration of seemingly suboptimal decisions. Non-stationarity not only affects ω-UCB's empirical performance but also impacts its theoretical guarantees. Incorporating change adaptation mechanisms into our approach would thus constitute a stand-alone research project. To begin, one could combine ω-UCB with our change detection algorithm, ABCD, to monitor rewards and costs over time, similar to [FKB19; KFH24]. The algorithms are likely to integrate well, as ABCD's adaptive windows store all the information needed to compute ω-UCB's confidence intervals.

Extending PLASTIC to the delayed-feedback setting. The decision-making process considered in this dissertation (Figure 1.1) assumes that feedback becomes available without significant delay. However, this assumption may not hold in real-world scenarios. For example, a financial transaction provider might only receive feedback about fraudulent transactions after several days. Meanwhile, it has already classified thousands of other transactions. Delayed feedback introduces additional complexities, particularly for monitoring the performance of a DSS and updating it using first unlabeled and later labeled data [Gom+23]. We believe that our incremental decision tree, PLASTIC, is well-suited for

this scenario due to its ability to restructure subtrees as needed. A natural starting point could involve using semi-supervision to assign pseudo-feedback to new observations, allowing immediate updates to the counters at the tree nodes. Once the true feedback arrives, the algorithm could correct the counters and restructure the tree if required.

Improving PLASTIC in non-stationary environments. In Chapter 4, we showed that coupling PLASTIC with a change detection algorithm produces a change-adaptive model, PLASTIC-A, which already achieves higher accuracy than the previous state-of-the-art. However, we believe one can improve PLASTIC-A even further by adopting more advanced change adaptation mechanisms from the literature [BG09; MSW22]. For instance, the algorithm could monitor accuracy at internal nodes, whereas PLASTIC-A currently monitors only the root node. This could lead to more nuanced reactions to changes by either pruning, retraining, or restructuring the affected subtrees.

Studying and refining ABCD's severity measure. With ABCD, we took an initial step toward quantifying change severity in multivariate data. However, the factors that influence the reliability of this severity measure have not been thoroughly explored. Currently, the measure shows only a moderate correlation with the ground-truth severity in our synthetic data streams. This might be due to several aspects. These include the algorithm's accuracy in identifying the change subspace, the design of the severity measure itself, the characteristics of the change (e.g., whether it involves a shift in mean, correlation, or both), and the ability of the encoder-decoder model to capture complex relationships, such as non-linear dependencies. Investigating these factors in detail would constitute a standalone research project and is therefore beyond the scope of this dissertation. Nonetheless, we see it as an interesting direction for future research.

Detecting gradual changes and quantify change rate. Currently, ABCD provides feedback about when and in which subspace a change occurred, and how strong it was. To further enhance this feedback, one could extend ABCD to distinguish between sudden and gradual changes, which occur over an extended period [Bif+18c]. One could achieve this by splitting ABCD's adaptive windows into smaller sub-windows and checking whether multiple sub-windows contain change points. Detecting such patterns would indicate that, and in which time period, a gradual change occurred.

Bibliography

[ACF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem". In: *Machine Learning* 47 (2002), pp. 235–256. DOI: doi.org/10.1023/A:1013689704352 (pages 18, 38, 39).

[AD16] Shipra Agrawal and Nikhil Devanur. "Linear contextual bandits with knapsacks". In: *NeurIPS*. Vol. 29. Barcelona, Spain: Curran Associates, Inc., 2016, pp. 3458–3467. ISBN: 978-1-5108-3881-9. DOI: 10.5555/3157382.3157484 (page 89).

[Ang+13] Davide Anguita et al. "A public domain dataset for human activity recognition using smartphones". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. Bruges, Belgium: i6doc.com, 2013. ISBN: 978-2-87419-081-0 (page 72).

[AP19] Andri Ashfahani and Mahardhika Pratama. "Autonomous deep learning: Continual learning approach for dynamic environments". In: *SDM 2019*. Calgary, Canada: SIAM, 2019, pp. 666–674. DOI: 10.1137/1.9781611975673.75 (pages 15, 23).

[APP11] Danilo Ardagna, Barbara Panicucci, and Mauro Passacantando. "A game theoretic formulation of the service provisioning problem in cloud systems". In: *WWW '11*. Hyderabad, India: ACM, 2011, pp. 177–186. ISBN: 978-1-4503-0632-4. DOI: 10.1145/1963405.1963433 (page 9).

[Ash+20] Andri Ashfahani et al. "DEVDAN: Deep evolving denoising autoencoder". In: *Neurocomputing* 390 (2020), pp. 297–314. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2019.07.106 (page 23).

[ASM07] Hanady Abdulsalam, David B. Skillicorn, and Patrick Martin. "Streaming random forests". In: *IDEAS '07*. Banff, Canada: IEEE, 2007, pp. 225–232. ISBN: 978-1-4503-0632-4. DOI: 10.1109/IDEAS.2007.4318108 (page 23).

[Ava+21] Vashist Avadhanula et al. "Stochastic bandits for multi-platform budget optimization in online advertising". In: *WWW '21*. Ljubljana, Slovenia: ACM / IW3C2, 2021, pp. 2805–2817. ISBN: 978-1-4503-8312-7. DOI: 10.1145/3442381.3450074 (page 21).

[Bah+20] Maroua Bahri et al. "Efficient batch-incremental classification using UMAP for evolving data streams". In: *IDA 2020*. Vol. 12080. Springer, 2020, pp. 40–53. ISBN: 978-3-030-44584-3. DOI: 10.1007/978-3-030-44584-3_4 (page 22).

[Bah+21] Maroua Bahri et al. "Data stream analysis: Foundations, major tasks and tools". In: *WIREs Data Mining Knowl. Discov.* 11.3 (2021). DOI: 10.1002/WIDM.1405 (pages 15, 22, 23).

[BD00] Rajendra Bhatia and Chandler Davis. “A better bound on the variance”. In: *American Mathematical Monthly* 107.4 (2000), pp. 353–357. DOI: 10.2307/2589180 (pages 35, 36, 38).

[BD16] Peter J. Brockwell and Richard A. Davis. *Introduction to time series and forecasting*. 3rd ed. Springer, 2016. ISBN: 978-3-319-29854-2. URL: https://doi.org/10.1007/978-3-319-29854-2 (page 16).

[BD99] Jock A. Blackard and Denis J. Dean. “Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables”. In: *Computers and Electronics in Agriculture* 24.3 (1999), pp. 131–151. DOI: 10.1016/S0168-1699(99)00046-0 (page 54).

[Ber24] Sergei Natanowitsch Bernstein. “On a modification of Chebyshev’s inequality and of the error formula of Laplace”. Russian. In: vol. 1. Ann. Sci. Inst. Savantes Ukraine, Sect. Math., 1924, pp. 38–49 (pages 27, 64).

[BG07] Albert Bifet and Ricard Gavaldà. “Learning from time-changing data with adaptive windowing”. In: *SDM*. Minneapolis, USA: SIAM, 2007, pp. 443–448. DOI: 10.1137/1.9781611972771.42 (pages 26–29, 64, 65).

[BG09] Albert Bifet and Ricard Gavaldà. “Adaptive learning from evolving data streams”. In: *IDA 2009*. Vol. 5772. Springer, 2009, pp. 249–260. DOI: 10.1007/978-3-642-03915-7_22 (pages 24, 25, 90).

[BHP10] Albert Bifet, Geoffrey Holmes, and Bernhard Pfahringer. “Leveraging Bagging for Evolving Data Streams”. In: *ECML PKDD 2010*. Vol. 6321. Barcelona, Spain: Springer, 2010, pp. 135–150. ISBN: 978-3-642-15880-3. DOI: 10.1007/978-3-642-15880-3_15 (page 73).

[Bif+10a] Albert Bifet et al. “Fast perceptron decision tree learning from evolving data streams”. In: *PAKDD 2010*. Vol. 6119. Springer, 2010, pp. 299–310. DOI: 10.1007/978-3-642-13672-6_30 (page 24).

[Bif+10b] Albert Bifet et al. “MOA: Massive online analysis”. In: *JMLR* 11 (2010), pp. 1601–1604. DOI: 10.5555/1756006.1859903 (page 53).

[Bif+18a] Albert Bifet et al. “Big data stream mining”. In: *Machine learning for data streams: with practical examples in MOA*. The MIT Press, 2018. ISBN: 978-0-262-34604-7. URL: https://doi.org/10.7551/mitpress/10654.003.0006 (pages 6, 9, 15, 16).

[Bif+18b] Albert Bifet et al. “Classification”. In: *Machine learning for data streams: with practical examples in MOA*. The MIT Press, 2018. ISBN: 978-0-262-34604-7. URL: https://doi.org/10.7551/mitpress/10654.003.0011 (pages 24, 48).

[Bif+18c] Albert Bifet et al. “Dealing with change”. In: *Machine learning for data streams: with practical examples in MOA*. The MIT Press, 2018. ISBN: 978-0-262-34604-7. URL: https://doi.org/10.7551/mitpress/10654.003.0010 (pages 6, 12, 90).

[Bif+18d] Albert Bifet et al. “Introduction”. In: *Machine learning for data streams: with practical examples in MOA*. The MIT Press, 2018. ISBN: 978-0-262-34604-7. URL: https://doi.org/10.7551/mitpress/10654.003.0005 (pages 6, 15).

[Bif+18e] Albert Bifet et al. *Machine learning for data streams: with practical examples in MOA*. The MIT Press, 2018. ISBN: 978-0-262-34604-7. URL: https://doi.org/10.7551/mitpress/10654.001.0001 (pages 6, 7).

[Bif+18f] Albert Bifet et al. “Regression”. In: *Machine learning for data streams: with practical examples in MOA*. The MIT Press, 2018. ISBN: 978-0-262-34604-7. URL: https://doi.org/10.7551/mitpress/10654.003.0013 (page 22).

[BKS13] Ashwinkumar Badanidiyuru, Robert Kleinberg, and Aleksandrs Slivkins. “Bandits with knapsacks”. In: *FOCS ’13*. IEEE, 2013, pp. 207–216. ISBN: 978-0-7695-5135-7. DOI: 10.1109/FOCS.2013.3 (pages 18, 19, 21, 22, 35).

[BLM13] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities – a nonasymptotic theory of independence*. Oxford University Press, 2013. ISBN: 978-0-19-174710-6. URL: https://doi.org/10.1093/acprof:oso/9780199535255.001.0001 (page 64).

[BLS14] Ashwinkumar Badanidiyuru, John Langford, and Aleksandrs Slivkins. “Resourceful contextual bandits”. In: *COLT 2014*. Vol. 35. Barcelona, Spain: PMLR, 2014, pp. 1109–1134. URL: https://proceedings.mlr.press/v35/badanidiyuru14.html (page 89).

[BMB18] Maroua Bahri, Silviu Maniu, and Albert Bifet. “A sketch-based naive bayes algorithms for evolving data streams”. In: *BigData 2018*. Seattle, USA: IEEE, 2018, pp. 604–613. DOI: 10.1109/BigData.2018.862217 (page 22).

[Bor+07] Christian Borgs et al. “Dynamics of bid optimization in online advertisement auctions”. In: *WWW ’07*. Banff, Canada: ACM, 2007, pp. 531–540. ISBN: 978-1-59593-654-7. DOI: 10.1145/1242572.1242644 (page 9).

[BP03] Jushan Bai and Pierre Perron. “Critical values for multiple structural change tests”. In: *The Econometrics Journal* 6.1 (2003). Publisher: Oxford University Press Oxford, UK, pp. 72–78. DOI: 10.1111/1368-423X.00102 (page 26).

[Bre+84] Leo Breiman et al. *Classification and regression trees*. Wadsworth, 1984. ISBN: 0-534-98053-8 (pages 10, 72).

[Bre96] Leo Breiman. “Bagging predictors”. In: *Machine Learning* 24.2 (1996), pp. 123–140. DOI: 10.1007/BF00058655 (page 23).

[BSW14] P. Baldi, P. Sadowski, and D. Whiteson. “Searching for exotic particles in high-energy physics with deep learning”. In: *Nature Communications* 5.1 (2014), p. 4308. DOI: 10.1038/ncomms5308 (page 54).

[Car+18] Fabrizio Carcillo et al. “SCARFF: A scalable framework for streaming credit card fraud detection with spark”. In: *Information Fusion* 41 (2018), pp. 182–194. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2017.09.005 (page 3).

[Cec+20] Michelangelo Ceci et al. “ECHAD: Embedding-based change detection from multivariate time series in smart grids”. In: *IEEE Access* 8 (2020), pp. 156053–156066. DOI: 10.1109/ACCESS.2020.3019095 (pages 27, 28).

[CES19] Semih Cayci, Atilla Eryilmaz, and R. Srikant. “Learning to control renewal processes with bandit feedback”. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3.2 (2019), 43:1–43:32. DOI: 10.1145/3341617.3326158 (page 21).

[CES20] Semih Cayci, Atilla Eryilmaz, and R. Srikant. “Budget-constrained bandits over general cost and reward distributions”. In: *AISTATS 2020*. Vol. 108. PMLR. PMLR, 2020, pp. 4388–4398. URL: http://proceedings.mlr.press/v108/cayci20a.html (pages 19–21, 35, 126).

[CFT21] Anamitra Chaudhuri, Georgios Fellouris, and Ali Tajer. “Sequential change detection of a correlation structure under a sampling constraint”. In: *IEEE International Symposium on Information Theory*. Melbourne, Australia: IEEE, 2021, pp. 605–610. DOI: 10.1109/ISIT45174.2021.9517736 (page 29).

[CGL82] T. F. Chan, G. H. Golub, and R. J. LeVeque. “Updating formulae and a pairwise algorithm for computing sample variances”. In: *COMPSTAT 1982*. Toulouse, France: Physica, 1982, pp. 30–41. ISBN: 978-3-642-51461-6. DOI: 10.1007/978-3-642-51461-6_3 (pages 68, 69).

[Cha+17] S. Chakar et al. “A robust approach for estimating change-points in the mean of an AR(1) process”. In: *Bernoulli* 23.2 (2017), pp. 1408–1447. DOI: 10.3150/15-BEJ782 (page 26).

[Das+06] Tamraparni Dasu et al. “An information-theoretic approach to detecting changes in multi-dimensional data streams”. In: *Interface 2006*. Pasadena, USA: Curran Associates, Inc., 2006. ISBN: 978-1-62276-715-1 (pages 27–29).

[Das+19] Monidipa Das et al. “MUSE-RNN: a multilayer self-evolving recurrent neural network for data stream classification”. In: *ICDM 2019*. Beijing, China: IEEE, 2019, pp. 110–119. DOI: 10.1109/ICDM.2019.00021 (page 23).

[DH00] Pedro M. Domingos and Geoff Hulten. “Mining high-speed data streams”. In: *KDD ’00*. ACM, 2000, pp. 71–80. ISBN: 1-58113-233-6. DOI: 10.1145/347090.347107 (pages 10, 23, 52).

[Din+13] Wenkui Ding et al. “Multi-armed bandit with budget constraint and variable costs”. In: *AAAI ’13*. Vol. 27. Bellevue, Washington, USA: AAAI Press, 2013, pp. 232–238. DOI: 10.1609/aaai.v27i1.8637 (pages 18, 21, 22).

[DJG22] Debojit Das, Shweta Jain, and Sujit Gujar. “Budgeted combinatorial multi-armed bandits”. In: *AAMAS ’22*. Virtual Event, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2022, pp. 345–353. ISBN: 978-1-4503-9213-6. DOI: 10.5555/3535850.3535890 (page 21).

[DMP21] Pietro Ducange, Francesco Marcelloni, and Riccardo Pecori. “Fuzzy Hoeffding decision tree for data stream classification”. In: 14.1 (2021), pp. 946–964. DOI: 10.2991/ijcis.d.210212.001 (page 24).

[EH20] Jesper E van Engelen and Holger H Hoos. "A survey on semi-supervised learning". In: *Machine Learning* 109.2 (2020), pp. 373–440. ISSN: 1573-0565. DOI: 10.1007/s10994-019-05855-6 (page 16).

[Fab+21] Kamil Faber et al. "WATCH: Wasserstein change point detection for high-dimensional time series data". In: *BigData 2021*. IEEE, 2021, pp. 4450–4459. DOI: 10.1109/BIGDATA52589.2021.9671962 (pages 26–29, 71).

[FCM22] Marta Fernandes, Juan Manuel Corchado, and Goreti Marreiros. "Machine learning techniques applied to mechanical fault diagnosis and fault prognosis in the context of real industrial manufacturing use-cases: a systematic literature review". In: *Applied Intelligence* 52.12 (2022), pp. 14246–14280. DOI: 10.1007/S10489-022-03344-3 (pages 3, 8).

[FDK19] William J. Faithfull, Juan José Rodríguez Diez, and Ludmila I. Kuncheva. "Combining univariate approaches for ensemble change detection in multivariate data". In: *Information Fusion* 45 (2019), pp. 202–214. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2018.02.003 (pages 27–29, 71).

[FGG97] Nir Friedman, Dan Geiger, and Moises Goldszmidt. "Bayesian network classifiers". In: *Machine Learning* 29.2 (Nov. 1997), pp. 131–163. ISSN: 1573-0565. DOI: 10.1023/A:1007465528199 (page 22).

[FKB19] Edouard Fouché, Junpei Komiyama, and Klemens Böhm. "Scaling multi-armed bandit algorithms". In: *KDD '19*. Anchorage, AK, USA: ACM, 2019, pp. 1449–1459. ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330862 (pages 7, 26, 89).

[Fon15] Jordi Fonollosa. *Gas sensor array under dynamic gas mixtures*. 2015. DOI: 10.24432/C5WP4C (page 54).

[Fox01] John Fox. "Decision support systems". In: *International encyclopedia of the social & behavioral sciences*. Ed. by Neil J. Smelser and Paul B. Baltes. Oxford: Pergamon, 2001, pp. 3323–3327. ISBN: 978-0-08-043076-8. URL: https://doi.org/10.1016/B0-08-043076-7/00554-4 (pages 3, 4).

[FS97] Yoav Freund and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1504 (page 23).

[GA18] Damien Garreau and Sylvain Arlot. "Consistent change-point detection with kernels". In: *Electronic Journal of Statistics* 12.2 (2018), pp. 4440–4486. DOI: 10.1214/18-EJS1513 (page 26).

[Gam10] João Gama. *Knowledge discovery from data streams*. 1st ed. Chapman & Hall/CRC, 2010. ISBN: 1-4398-2611-0. URL: https://dl.acm.org/doi/10.5555/1855075 (pages 6, 7, 9).

[GFR06] João Gama, Ricardo Fernandes, and Ricardo Rocha. "Decision trees for mining data streams". In: *Intelligent Data Analysis* 10.1 (2006), pp. 23–45. DOI: 10.3233/IDA-2006-10103 (page 24).

[Gom+17a] Heitor Murilo Gomes et al. “A survey on ensemble learning for data stream classification”. In: *ACM Computing Surveys* 50.2 (2017), 23:1–23:36. DOI: 10.1145/3054925 (page 23).

[Gom+17b] Heitor Murilo Gomes et al. “Adaptive random forests for evolving data stream classification”. In: *Machine Learning* 106.9 (2017), pp. 1469–1495. DOI: 10.1007/s10994-017-5642-8 (pages 7, 23).

[Gom+20] Heitor Murilo Gomes et al. “On ensemble techniques for data stream regression”. In: *IJCNN 2020*. 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9206756 (pages 10, 22).

[Gom+23] Heitor Murilo Gomes et al. “A survey on semi-supervised learning for delayed partially labelled data streams”. In: *ACM Computing Surveys* 55.4 (2023), 75:1–75:42. DOI: 10.1145/3523055 (pages 7–9, 16, 89).

[Göz+19] Ömer Gözüaçik et al. “Unsupervised concept drift detection with a discriminative classifier”. In: *CIKM ’19*. Beijing, China: ACM, 2019, pp. 2365–2368. ISBN: 978-1-4503-6976-3. DOI: 10.1145/3357384.3358144 (pages 26–29).

[GRB19] Heitor Murilo Gomes, Jesse Read, and Albert Bifet. “Streaming random patches for evolving data stream classification”. In: *ICDM 2019*. Beijing, China: IEEE, 2019, pp. 240–249. DOI: 10.1109/ICDM.2019.00034 (pages 7, 23).

[Gre+12] Arthur Gretton et al. “A kernel two-sample test”. In: *JMLR* 13.25 (2012), pp. 723–773. URL: https://jmlr.org/papers/v13/gretton12a.html (page 62).

[GRM03] João Gama, Ricardo Rocha, and Pedro Medas. “Accurate decision trees for mining high-speed data streams”. In: *KDD ’03*. Washington, DC, USA: ACM, 2003, pp. 523–528. ISBN: 1-58113-737-0. DOI: 10.1145/956750.956813 (page 24).

[Gun+22] Nuwan Gunasekara et al. “Online hyperparameter optimization for streaming neural networks”. In: *IJCNN 2022*. 2022, pp. 1–9. DOI: 10.1109/IJCNN55064.2022.9891953 (page 23).

[Gun+23] Nuwan Gunasekara et al. “Survey on online streaming continual learning”. In: *IJCAI-23*. Macao: IJCAI Organization, 2023, pp. 6628–6637. DOI: 10.24963/IJCAI.2023/743 (page 15).

[Gun+24] Nuwan Gunasekara et al. “Gradient boosted trees for evolving data streams”. In: *Machine Learning* 113.5 (2024), pp. 3325–3352. DOI: 10.1007/S10994-024-06517-Y (page 23).

[GW19] Igor Goldenberg and Geoffrey I. Webb. “Survey of distance measures for quantifying concept drift and shift in numeric data”. In: *Knowledge and Information Systems* 60.2 (Aug. 2019), pp. 591–615. ISSN: 0219-3116. DOI: 10.1007/s10115-018-1257-z (page 27).

[HC07] Zaid Harchaoui and Olivier Cappé. "Retrospective mutiple change-point estimation with kernels". In: *2007 IEEE/SP 14th Workshop on Statistical Signal Processing*. IEEE, 2007, pp. 768–772. DOI: 10.1109/SSP.2007.4301363 (page 26).

[Hey+24a] Marco Heyden et al. "Adaptive Bernstein change detector for high-dimensional data streams". In: *Data Mining and Knowledge Discovery* 38.3 (2024), pp. 1334–1363. DOI: 10.1007/S10618-023-00999-5 (pages 12, 13, 28, 61, 88).

[Hey+24b] Marco Heyden et al. "Budgeted multi-armed bandits with asymmetric confidence intervals". In: *KDD '24: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Barcelona, Spain: ACM, 2024, pp. 1073–1084. ISBN: 9798400704901. DOI: 10.1145/3637528.3671833 (pages 12, 13, 33, 88).

[Hey+24c] Marco Heyden et al. "Leveraging plasticity in incremental decision trees". In: *Machine Learning and Knowledge Discovery in Databases. Research Track. ECML PKDD 2024*. Vilnius, Lithuania: Springer Nature, 2024, pp. 38–54. ISBN: 978-3-031-70362-1. DOI: 10.1007/978-3-031-70362-1_3 (pages 12, 13, 22, 47, 88).

[HSD01] Geoff Hulten, Laurie Spencer, and Pedro M. Domingos. "Mining time-changing data streams". In: *KDD '01*. San Francisco, California, USA: ACM, 2001, pp. 97–106. ISBN: 1-58113-391-X. DOI: 10.1145/502512.502529 (page 24).

[HY09] Sattar Hashemi and Ying Yang. "Flexible decision tree for data stream classification in the presence of concept change, noise and missing values". In: *Data Mining and Knowledge Discovery* 19.1 (2009), pp. 95–131. DOI: 10.1007/s10618-009-0130-9 (page 24).

[ICC19] Angelo Impedovo, Michelangelo Ceci, and Toon Calders. "Efficient and accurate non-exhaustive pattern-based change detection in dynamic networks". In: *Discovery Science. DS 2019*. Vol. 11828. Split, Croatia: Springer, 2019, pp. 396–411. DOI: 10.1007/978-3-030-33778-0_30 (page 27).

[IGD11] Elena Ikonomovska, João Gama, and Sašo Džeroski. "Learning model trees from evolving data streams". In: *Data Mining and Knowledge Discovery* 23.1 (2011), pp. 128–168. DOI: 10.1007/s10618-010-0201-y (page 54).

[Imp+20a] Angelo Impedovo et al. "Condensed representations of changes in dynamic graphs through emerging subgraph mining". In: *Engineering Applications of Artificial Intelligence* 94 (2020), p. 103830. DOI: 10.1016/j.engappai.2020.103830 (page 27).

[Imp+20b] Angelo Impedovo et al. "Simultaneous process drift detection and characterization with pattern-based change detectors". In: *Discovery Science. DS 2020*. Vol. 12323. Thessaloniki, Greece: Springer, 2020, pp. 451–467. ISBN: 978-3-030-61527-7. DOI: 10.1007/978-3-030-61527-7_30 (page 27).

[JCG18] Yuchen Jiao, Yanxi Chen, and Yuantao Gu. "Subspace change-point detection: a new model and solution". In: *IEEE Journal of Selected Topics in Signal Processing* 12.6 (2018), pp. 1224–1239. DOI: 10.1109/JSTSP.2018.2873147 (page 29).

[JRA20] Maciej Jaworski, Leszek Rutkowski, and Plamen Angelov. "Concept drift detection using autoencoders in data streams processing". In: *Artificial Intelligence and Soft Computing. ICAISC 2020*. Zakopane, Poland: Springer International Publishing, 2020, pp. 124–133. ISBN: 978-3-030-61401-0. DOI: 10.1007/978-3-030-61401-0_12 (pages 27, 63).

[KB15] Diederik P. Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *ICLR*. 2015. URL: http://arxiv.org/abs/1412.6980 (page 71).

[KFE12] Rebecca Killick, Paul Fearnhead, and Idris A Eckley. "Optimal detection of changepoints with a linear computational cost". In: *Journal of the American Statistical Association* 107.500 (2012), pp. 1590–1598. DOI: 10.1080/01621459.2012.737745 (page 26).

[KFH24] Junpei Komiyama, Edouard Fouché, and Junya Honda. "Finite-time analysis of globally nonstationary multi-armed bandits". In: *JMLR* 25 (2024), 112:1–112:56. URL: https://jmlr.org/papers/v25/21-0916.html (pages 26, 89).

[Kha+15] Imen Khamassi et al. "Self-adaptive windowing approach for handling complex concept drift". In: *Cognitive Computation* 7.6 (2015), pp. 772–790. DOI: 10.1007/s12559-015-9341-0 (page 26).

[Kir+17] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526. DOI: 10.1073/pnas.1611835114 (page 15).

[KK21] Łukasz Korycki and Bartosz Krawczyk. "Class-incremental experience replay for continual learning under concept drift". In: *CVPR Workshops 2021*. Montreal, BC, Canada: Computer Vision Foundation / IEEE, 2021, pp. 3644–3653. DOI: 10.1109/CVPRW53098.2021.00404 (page 15).

[Knu97] Donald E. Knuth. *The art of computer programming: seminumerical algorithms*. 3rd ed. Vol. 2. USA: Addison-Wesley Professional, 1997. ISBN: 0-201-89684-2 (page 68).

[Kra+17] Bartosz Krawczyk et al. "Ensemble learning for data stream analysis: A survey". In: *Information Fusion* 37 (2017), pp. 132–156. ISSN: 1566-2535. DOI: 10.1016/j.inffus.2017.02.004 (page 23).

[Kri09] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009 (page 73).

[LBA14] Rémi Lajugie, Francis R. Bach, and Sylvain Arlot. "Large-margin metric learning for constrained partitioning problems". In: *ICML 2014*. Vol. 32(1). Bejing, China: PMLR, 2014, pp. 297–305. URL: https://proceedings.mlr.press/v32/lajugie14.html (page 26).

[LCB10] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. *MNIST handwritten digit database*. 2010. URL: https://yann.lecun.com/exdb/mnist/ (page 73).

[Lem17] Madis Lemsalu. *Facebook ad campaign*. 2017. URL: https://www.kaggle.com/madislemsalu/facebook-ad-campaign (visited on 12/02/2024) (pages 43, 44).

[LG12] Gilles Louppe and Pierre Geurts. "Ensembles on random patches". In: *ECML PKDD 2012*. Vol. 7523. Springer, 2012, pp. 346–361. ISBN: 978-3-642-33460-3. DOI: 10.1007/978-3-642-33460-3_28 (page 23).

[LHW16] Viktor Losing, Barbara Hammer, and Heiko Wersing. "KNN classifier with self adjusting memory for heterogeneous concept drift". In: *ICDM 2016*. Barcelona, Spain: IEEE, 2016, pp. 291–300. DOI: 10.1109/ICDM.2016.0040 (pages 22, 54).

[Liu+17] Anjin Liu et al. "Regional concept drift detection and density synchronized drift adaptation". In: *IJCAI-17*. Melbourne, Australia: ijcai.org, 2017, pp. 2280–2286. DOI: 10.24963/ijcai.2017/317 (pages 28, 29).

[LJA14] Victor E. Lee, Ruoming Jin, and Gagan Agrawal. "Frequent pattern mining in data streams". In: *Frequent pattern mining*. Ed. by Charu C. Aggarwal and Jiawei Han. Springer, 2014, pp. 199–224. ISBN: 978-3-319-07821-2. URL: https://doi.org/10.1007/978-3-319-07821-2_9 (page 25).

[LLC15] Alexandre Lung-Yut-Fong, Céline Lévy-Leduc, and Olivier Cappé. "Homogeneity and change-point detection tests for multivariate data using rank statistics". In: *Journal de la Société Française de Statistique* 156.4 (2015), pp. 133–162 (page 26).

[Log+18] Corrado Loglisci et al. "Mining microscopic and macroscopic changes in network data streams". In: *Knowledge-Based Systems* 161 (2018), pp. 294–312. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2018.07.011 (page 27).

[Log+21] Aleksej Logacjov et al. "HARTH: a human activity recognition dataset for machine learning". In: *Sensors* 21.23 (2021), p. 7853. ISSN: 1424-8220. DOI: 10.3390/s21237853 (page 54).

[LS20] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge: Cambridge University Press, 2020. ISBN: 978-1-108-57140-1. URL: https://doi.org/10.1017/9781108571401 (pages 4, 6, 18).

[Lu+19] Jie Lu et al. "Learning under concept drift: a review". In: *IEEE Transactions on Knowledge and Data Engineering* 31.12 (2019), pp. 2346–2363. DOI: 10.1109/TKDE.2018.2876857 (pages 11, 27–29).

[LV07] John A. Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. 1st ed. Springer New York, NY, 2007. ISBN: 978-0-387-39351-3. URL: https://doi.org/10.1007/978-0-387-39351-3 (page 62).

[LZ15] Yu-Feng Li and Zhi-Hua Zhou. "Towards making unlabeled data never hurt". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.1 (2015), pp. 175–188. DOI: 10.1109/TPAMI.2014.2299812 (page 16).

[MJ14] David S. Matteson and Nicholas A. James. "A nonparametric approach for multiple change point analysis of multivariate data". In: *Journal of the American Statistical Association* 109.505 (2014), pp. 334–345. DOI: 10.1080/01621459.2013.849605 (page 26).

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. 2nd ed. The MIT press, 2018. ISBN: 978-0-262-35136-2 (page 5).

[MSW22] Chaitanya Manapragada, Mahsa Salehi, and Geoffrey I. Webb. "Extremely fast Hoeffding adaptive tree". In: *ICDM 2022*. Orlando, FL, USA: IEEE, 2022, pp. 319–328. DOI: 10.1109/ICDM54844.2022.00042 (pages 24, 25, 52, 55, 90).

[MWS18] Chaitanya Manapragada, Geoffrey I. Webb, and Mahsa Salehi. "Extremely fast decision tree". In: *KDD '18*. ACM, 2018, pp. 1953–1962. ISBN: 978-1-4503-5552-0. DOI: 10.1145/3219819.3220005 (pages 10, 24, 52).

[Oli+18] Avital Oliver et al. "Realistic evaluation of deep semi-supervised learning algorithms". In: *NeurIPS*. Montréal, Canada: Curran Associates Inc., 2018, pp. 3239–3250. DOI: 10.5555/3327144.3327244 (page 16).

[Oza05] Nikunj C. Oza. "Online bagging and boosting". In: *2005 IEEE international conference on systems, man and cybernetics*. Vol. 3. Waikoloa, Hawaii, USA: IEEE, 2005, pp. 2340–2345. DOI: 10.1109/ICSMC.2005.1571498 (page 23).

[Pag54] Ewan S. Page. "Continuous inspection schemes". In: *Biometrika* 41.1/2 (June 1954), pp. 100–115. ISSN: 00063444. DOI: 10.2307/2333009 (page 28).

[PHK07] Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. "New options for Hoeffding trees". In: *Advances in Artificial Intelligence. AI 2007*. Springer, 2007, pp. 90–99. ISBN: 978-3-540-76928-6. DOI: 10.1007/978-3-540-76928-6_11 (page 24).

[PSK14] Russel Pears, Sripirakas Sakthithasan, and Yun Sing Koh. "Detecting concept change in dynamic data streams – a sequential approach based on reservoir sampling". In: *Machine Learning* 97.3 (2014), pp. 259–293. ISSN: 1573-0565. DOI: 10.1007/s10994-013-5433-9 (pages 27–29, 64).

[Qah+15] Abdulhakim A. Qahtan et al. "A PCA-based change detection framework for multidimensional data streams: change detection in multidimensional data streams". In: *KDD '15*. Sydney, Australia: ACM, Aug. 2015, pp. 935–944. ISBN: 978-1-4503-3664-2. DOI: 10.1145/2783258.2783359 (pages 26–29).

[Qui93] J. Ross Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993. ISBN: 1-55860-240-2. URL: https://doi.org/10.5555/583200 (page 10).

[Rei+16] Denis Moreira dos Reis et al. “Fast unsupervised online drift detection using incremental Kolmogorov-Smirnov test”. In: *KDD ’16*. San Francisco, California, USA: ACM, 2016, pp. 1545–1554. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939836 (pages 26–29).

[Rei12] Attila Reiss. *PAMAP2 physical activity monitoring*. 2012. DOI: 10.24432/C5NW2H (page 54).

[RO06] Cattral Robert and Franz Oppacher. *Poker hand*. 2006. DOI: 10.24432/C5KW38 (page 54).

[Rut+13] Leszek Rutkowski et al. “Decision trees for mining data streams based on the McDiarmid’s bound”. In: *IEEE Transactions on Knowledge and Data Engineering* 25.6 (2013), pp. 1272–1279. DOI: 10.1109/TKDE.2012.66 (page 24).

[Rut+14a] Leszek Rutkowski et al. “Decision trees for mining data streams based on the Gaussian approximation”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.1 (2014), pp. 108–119. DOI: 10.1109/TKDE.2013.34 (page 24).

[Rut+14b] Leszek Rutkowski et al. “The CART decision tree for mining data streams”. In: *Information Sciences* 266 (2014), pp. 1–15. ISSN: 0020-0255. DOI: 10.1016/j.ins.2013.12.060 (page 24).

[Rut+15] Leszek Rutkowski et al. “A new method for data stream mining based on the misclassification error”. In: *IEEE Transactions on Neural Networks and Learning Systems* 26.5 (2015), pp. 1048–1059. DOI: 10.1109/TNNLS.2014.2333557 (page 24).

[Sàn22] Miquel Sànchez-Marrè. “Intelligent decision support systems”. In: *Intelligent decision support systems*. 1st ed. Springer, 2022, pp. 77–116. ISBN: 978-3-030-87790-3. URL: https://doi.org/10.1007/978-3-030-87790-3 (page 4).

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. 2nd ed. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0-262-03924-9. URL: https://dl.acm.org/doi/10.5555/3312046 (page 17).

[SCM20] Vinícius M. A. de Souza, Farhan Asif Chowdhury, and Abdullah Mueen. “Unsupervised drift detection on high-speed data streams”. In: *BigData 2020*. Atlanta, GA, USA: IEEE, 2020, pp. 102–111. DOI: 10.1109/BIGDATA50022.2020.9377880 (pages 26–29).

[SGW24] Rui Su, Husheng Guo, and Wenjian Wang. “Elastic online deep learning for dynamic streaming data”. In: *Information Sciences* 676 (2024), p. 120799. ISSN: 0020-0255. DOI: doi.org/10.1016/j.ins.2024.120799 (page 23).

[She31] Walter A. Shewhart. *Economic control of quality of manufactured product*. Van Nostrand, 1931. ISBN: 1-61427-811-3 (page 28).

[SNZ08] Aarti Singh, Robert Nowak, and Jerry Zhu. “Unlabeled data: Now it helps, now it doesn’t”. In: *NeurIPS*. Ed. by D. Koller et al. Vol. 21. Vancouver, BC, Canada: Curran Associates, Inc., 2008. URL: https://proceedings.neurips.cc/paper_files/paper/2008/file/07871915a8107172b3b5dc15a6574ad3-Paper.pdf (page 16).

[SOB22] Imen Souiden, Mohamed Nazih Omri, and Zaki Brahmi. "A survey of outlier detection in high dimensional data streams". In: *Computer Science Review* 44 (2022), p. 100463. ISSN: 1574-0137. DOI: 10.1016/J.COSREV.2022.100463 (page 25).

[Sou+20] Vinicius M. A. Souza et al. "Challenges in benchmarking stream learning algorithms with real-world data". In: *Data Mining and Knowledge Discovery* 34.6 (2020), pp. 1805–1858. DOI: 10.1007/s10618-020-00698-5 (pages 53, 54).

[SS14] Harald Schöpf and Peter Supancic. "On Bürmann's theorem and its application to problems of linear and nonlinear heat transfer and diffusion". In: *The Mathematica Journal* 16 (2014), pp. 1–44. DOI: 10.3888/tmj.16-11 (page 39).

[SS17] Robert H. Shumway and David S. Stoffer. *Time series analysis and its applications*. 4th ed. Springer, 2017. ISBN: 978-3-319-52452-8. URL: https://doi.org/10.1007/978-3-319-52452-8 (page 16).

[Sto+98] Salvatore Stolfo et al. *KDD cup 1999 data*. 1998. DOI: 10.24432/C51C7N (page 54).

[Sun+16] Yange Sun et al. "Online ensemble using adaptive windowing for data streams with concept drift". In: *International Journal of Distributed Sensor Networks* 12.5 (2016), 4218973:1–4218973:9. DOI: 10.1155/2016/4218973 (page 26).

[Sur+22] Shubhangi Suryawanshi et al. "Adaptive windowing based recurrent neural network for drift adaption in non-stationary environment". In: *Journal of Ambient Intelligence and Humanized Computing* 14 (2022), pp. 14125–14139. DOI: 10.1007/s12652-022-04116-0 (page 26).

[Tho33] William R. Thompson. "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples". In: *Biometrika* 25.3/4 (1933), pp. 285–294. DOI: https://doi.org/10.2307/2332286 (page 18).

[Tho35] William R. Thompson. "On the theory of apportionment". In: *American Journal of Mathematics* 57.2 (1935), pp. 450–456. DOI: 10.2307/2371219 (page 18).

[TK74] Amos Tversky and Daniel Kahneman. "Judgment under uncertainty: Heuristics and biases". In: *Science* 185.4157 (1974), pp. 1124–1131. DOI: 10.1126/science.185.4157.1124 (page 4).

[TOV20] Charles Truong, Laurent Oudre, and Nicolas Vayatis. "Selective review of offline change point detection methods". In: *Signal Processing* 167 (2020), p. 107299. ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2019.107299 (page 26).

[Tra+10] Long Tran-Thanh et al. "Epsilon-first policies for budget-limited multi-armed bandits". In: *AAAI '10*. Vol. 24. Atlanta, GA, USA: AAAI Press, 2010, pp. 1211–1216. DOI: 10.1609/aaai.v24i1.7758 (pages 9, 18, 21, 22, 34).

[Tra+12] Long Tran-Thanh et al. "Knapsack based optimal policies for budget-limited multi-armed bandits". In: *AAAI '12*. Vol. 26. Toronto, Canada: AAAI Press, 2012, pp. 1134–1140. DOI: 10.1609/aaai.v26i1.8279 (pages 9, 18, 21, 22).

[UBC97] Paul E. Utgoff, Neil C. Berkman, and Jeffery A. Clouse. "Decision tree induction based on efficient tree restructuring". In: *Machine Learning* 29.1 (1997), pp. 5–44. DOI: 10.1023/A:1007413323501 (page 25).

[Utg89] Paul E. Utgoff. "Incremental induction of decision trees". In: *Machine Learning* 4 (1989), pp. 161–186. DOI: 10.1023/A:1022699900025 (page 25).

[Ver+11] Alexander Vergara et al. "Gas sensor drift mitigation using classifier ensembles". In: *SensorKDD '11: Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data*. San Diego, California: ACM, 2011, pp. 16–24. ISBN: 978-1-4503-0832-8. DOI: 10.1145/2003653.2003655 (page 72).

[Wan+24] Liyuan Wang et al. "A comprehensive survey of continual learning: Theory, method and application". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.8 (2024), pp. 5362–5383. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2024.3367329 (page 15).

[Wat+17] Ryo Watanabe et al. "KL-UCB-based policy for budgeted multi-armed bandits with stochastic action costs". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E100.A.11 (2017), pp. 2470–2486. DOI: 10.1587/transfun.E100.A.2470 (pages 9, 18, 19, 21, 22, 34, 42).

[Wat+18] Ryo Watanabe et al. "UCB-SC: A fast variant of KL-UCB-SC for budgeted multi-armed bandit problem". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E101.A.3 (2018), pp. 662–667. DOI: 10.1587/transfun.E101.A.662 (pages 9, 18, 19, 21, 22, 34, 42, 43, 126).

[Web+18] Geoffrey I. Webb et al. "Analyzing concept drift and shift from sample data". In: *Data Mining and Knowledge Discovery* 32.5 (2018), pp. 1179–1199. ISSN: 1384-5810. DOI: 10.1007/s10618-018-0554-1 (pages 11, 28).

[Wei19] Gary Weiss. *WISDM smartphone and smartwatch activity and biometrics dataset*. 2019. DOI: 10.24432/C5HK59 (page 54).

[Wei23] William W. S. Wei. *Time series analysis: Univariate and multivariate methods (classic version)*. 2nd ed. Pearson, 2023. ISBN: 978-0-13-798146-5 (page 16).

[Whi14] Daniel Whiteson. *SUSY*. 2014. DOI: 10.24432/C54606 (page 54).

[Whi16] Daniel Whiteson. *HEPMASS*. 2016. DOI: 10.24432/C5PP5W (page 54).

[Wil27] Edwin B. Wilson. "Probable inference, the law of succession, and statistical inference". In: *Journal of the American Statistical Association* 22.158 (1927), pp. 209–212. DOI: doi.org/10.2307/2276774 (pages 33, 35, 38).

[WLH12] Xindong Wu, Pei-Pei Li, and Xuegang Hu. "Learning from concept drifting data streams with unlabeled data". In: *Neurocomputing* 92 (2012), pp. 145–155. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2011.08.041 (page 24).

[WW21] Edmund T. Whittaker and George N. Watson. *A course of modern analysis*. 5th ed. Cambridge: Cambridge University Press, 2021. ISBN: 1-316-51893-0 (page 39).

[Xia+15a] Yingce Xia et al. “Budgeted bandit problems with continuous random costs”. In: *Proceedings of The 7th Asian Conference on Machine Learning. ACML 2015*. Vol. 45. Hong Kong: JMLR.org, 2015, pp. 317–332. URL: http://proceedings.mlr.press/v45/Xia15.html (pages 9, 18, 20–22, 34, 35, 42, 43, 89, 126).

[Xia+15b] Yingce Xia et al. “Thompson sampling for budgeted multi-armed bandits”. In: *IJCAI-15*. Buenos Aires, Argentina: AAAI Press, 2015, pp. 3960–3966. URL: https://ijcai.org/Abstract/15/556 (pages 18, 21, 22, 34, 42, 43, 126).

[Xia+16] Yingce Xia et al. “Budgeted multi-armed bandits with multiple plays”. In: *IJCAI-16*. New York, New York, USA: AAAI Press, 2016, pp. 2210–2216. ISBN: 978-1-57735-770-4 (pages 9, 19, 35, 42, 43).

[Xia+17] Yingce Xia et al. “Finite budget analysis of multi-armed bandit problems”. In: *Neurocomputing* 258 (2017), pp. 13–29. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2016.12.079 (pages 9, 18–22, 34, 35, 41–43, 124, 126).

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms”. In: *CoRR* abs/1708.07747 (2017). URL: https://arxiv.org/abs/1708.07747 (page 73).

[XXM20] Liyan Xie, Yao Xie, and George V. Moustakides. “Sequential subspace change point detection”. In: *Sequential Analysis* 39.3 (2020), pp. 307–335. DOI: 10.1080/07474946.2020.1823191 (page 29).

[Yan+06] Jun Yan et al. “Effective and efficient dimensionality reduction for large-scale and streaming data preprocessing”. In: *IEEE Transactions on Knowledge and Data Engineering* 18.2 (2006), pp. 320–333. DOI: 10.1109/TKDE.2006.45 (page 70).

[Yua+12] Shuai Yuan et al. “Internet advertising: An interplay among advertisers, online publishers, ad exchanges and web users”. In: *CoRR* abs/1206.1754 (2012) (pages 3, 8).

[ZA21] Alaettin Zubaroglu and Volkan Atalay. “Data stream clustering: a review”. In: *Artificial Intelligence Review* 54.2 (2021), pp. 1201–1236. DOI: 10.1007/S10462-020-09874-X (page 25).

[Zho+23] Yuan Zhong et al. “Dynamically evolving deep neural networks with continuous online learning”. In: *Information Sciences* 646 (2023), p. 119411. ISSN: 0020-0255. DOI: 10.1016/j.ins.2023.119411 (page 15).

[Zhu10] Xingquan Zhu. *Sensor stream*. 2010. URL: https://www.cse.fau.edu/~xqzhu/stream.html (page 54).

[ZY24] Yafei Zhao and Long Yang. “Constrained contextual bandit algorithm for limited-budget recommendation system”. In: *Engineering Applications of Artificial Intelligence* 128 (2024), p. 107558. ISSN: 0952-1976. DOI: 10.1016/j.engappai.2023.107558 (page 89).

List of Figures

List of Tables

List of Theorems

List of Algorithms

Acronyms

ABCD Adaptive Bernstein Change Detector.

AE Autoencoder.

BMAB Budgeted multi-armed bandit.

BTS Budgeted Thompson sampling.

BwK Bandits with knapsacks.

CB Contextual multi-armed bandits.

CL Continual learning.

DSS Decision support system.

EFDT Extremly Fast Decision Tree.

EFHAT Extremely Fast Hoeffding Adaptive Tree.

HT Hoeffding Tree.

i.i.d. Independent and identically distributed.

KDE Kernel density estimation.

KL Kullback–Leibler.

kNN K-nearest neighbor.

KPCA Kernel-PCA.

LCB Lower confidence bound.

MAB Multi-armed bandit.

ML Machine learning.

MLDS ML for data streams.

MTD Mean time until detection.

MTPO Mean time per observation.

NB Naive Bayes.

NC No-change classifier.

NN Neural network.

PCA Principal component analysis.

ReLU Rectified linear unit, $\text{ReLU}(x) = \max\{0, x\}$.

RL Reinforcement learning.

SSL Semi-supervised learning.

TSA Time series analysis.

UCB Upper confidence bound.

Notation

General

X	A random variable.
$X^{(n)}$	A sample of size n from X.
n	The size of a sample from X.
μ	The expected value of a random variable X.
$\hat{\mu}$	The arithmetic mean of a sample from X.
σ	The standard deviation of a random variable X.
$\hat{\sigma}$	The sample standard deviation of a sample from X.
α	The significance level of a statistical test, $\alpha \in [0, 1]$.
m	The lower bound of a univariate bounded random variable, $m \in \mathbb{R}$.
M	The upper bound of a univariate bounded random variable, $M \in \mathbb{R}$.

Sequential Decision Making

t	The current time step, $t \geq 1$.
τ	A time step (auxiliary notation).
$\mathbf{x}_i$	The i-th observation, a d-dimensional vector of attributes $\mathbf{x}_i = \langle \mathbf{x}_{ij} \rangle_{j \in [1,\ldots,d]}$.
$\mathbf{x}_{ij}$	The j-th value of observation $\mathbf{x}_i$.
D	The set of dimensions $\{1, 2, \ldots, d\}$.
y_i	The feedback for observation $\mathbf{x}_i$.
$\hat{y}_i$	The surrogate feedback for observation $\mathbf{x}_i$.
K	The number of possible decisions.
$[K]$	The set of decisions, $[K] = \{1, \ldots, K\}$.
k_t	The decision at time t, $k_t \in [K]$.
S	A data stream $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_t, y_t), \ldots\}$.
s_i	The i-th change point in a data stream.

$\mathcal{S}_{s_i,s_{i+1}}$	The concept of the data stream in the time interval $[s_i, s_{i+1})$.
$\mathcal{S}_t$	The concept of the data stream at time t.
m_t	A machine learning model m at time t.

Chapter 3 Notation

B	The budget available to the decision maker, $B > 0$.
T	The time horizon.
T_B	The time horizon until the budget B is depleted; a random variable.
μ_k^r	The expected rewards of decision k, $\mu_k^r \in [0, 1)$.
μ_k^c	The expected costs of decision k, $\mu_k^c \in (0, 1]$.
r_t	The reward at time t, $r_t \in [0, 1]$.
c_t	The cost at time t, $c_t \in [0, 1]$.
$\hat{\mu}_k^r(t)$	The average rewards of decision k observed until time t, $\hat{\mu}_k^r(t) \in [0, 1]$.
$\hat{\mu}_k^c(t)$	The average costs of decision k observed until time t, $\hat{\mu}_k^c(t) \in [0, 1]$.
Δ_k	The suboptimality gap between arm 1 and arm k, $\Delta_k = \mu_1^r/\mu_1^c - \mu_k^r/\mu_k^c$.
$n_k(t)$	The number of times decision k was chosen until time t.
$\omega_{\pm}(\alpha)$	Upper (+) and lower (-) confidence bounds (UCB) with significance α for a random variable's expected value.
$\Omega(\alpha)$	A UCB for the ratio of expected values of two random variables.
z	The number of standard deviations that lead to a a $1 - \alpha$ coverage of the normal distribution.
$\Omega_k(\alpha, t)$	A UCB for the reward-cost ratio for decision k at time t for significance level α. Since α is a function of t, we write $\Omega_k(t)$ to simplify notation.
$\omega_{k+}^r(\alpha, t)$	A UCB for the rewards of arm k at time t for significance level α. Since α is a function of t, we write $\omega_{k+}^r(t)$ to simplify notation.
$\omega_{k-}^c(\alpha, t)$	A lower confidence bound for the costs of arm k at time t with significance α. Since α is a function of t, we write $\omega_{k-}^c(t)$ to simplify notation.
$z_\rho(t)$	The number of standard deviations as a function of time, $z_\rho(t) = \sqrt{2\rho \log t}$. Results in a time-dependent significance level of $\alpha(t) < 1 - \sqrt{1 - t^{-\rho}}$.
η	Variance factor that allows tightening the confidence intervals in ω-UCB. We denote with η_k^r and η_k^c the variance factor for the rewards and costs of arm k, respectively. $\eta = \sigma^2/((M - \mu)(\mu - m)) \in [0, 1]$.
ρ	The scaling parameter of the confidence interval in ω-UCB, $\rho > 0$.

τ_B	A number of time steps that is logarithmic w.r.t. B, $\tau_B = \lfloor 2B / \min_{k \in [K]} \mu_k^c \rfloor$.
δ_k	The proportional δ-gap between arm 1 and arm k, $\delta_k = \Delta_k / (\Delta_k + 1/\mu_k^c)$.
$n_k^*(\tau)$	A number of plays of arm k that is logarithmic w.r.t. τ, see Eq. (3.9).
$\xi(\tau, \rho)$	The term that governs ω-UCB's asymptotic behavior, see Eq. (3.8) and Eq. (3.10).

Chapter 4 Notation

a_j	The j-th dimension (attribute).
ν_j	The number of categories of (a categorical) dimension a_j.
v	The split value associated with a decision tree node.
λ	Maximum depth of the restructured subtree.

Chapter 5 Notation

ϕ	An encoder function, $\phi : [0,1]^d \to [0,1]^{d'}$.
ψ	A decoder function, $\psi : [0,1]^{d'} \to [0,1]^d$.
$\hat{\mathbf{x}}_i$	A reconstructed observation, $\hat{\mathbf{x}}_i = \psi(\phi(\mathbf{x}_i))$.
d'	The dimensionality of an encoded observation, $0 < d' < d$.
γ	The encoding factor of ABCD, $\gamma = d'/d$.
L_i	The reconstruction error between observation $\mathbf{x}_i$ and its reconstruction $\hat{\mathbf{x}}_i$.
$\hat{\mu}_{t_1,t_2}$	The average reconstruction error in a time interval $[t_1, t_2]$.
E	The number of training epochs when using autoencoders.
ζ	The subspace threshold. If $p_{ij} < \zeta$, add dimension j to the change subspace.
t^*	The detected change point.
D^*	The detected change subspace.
θ	The change severity measure.
ssd_{t_1,t_2}	The sum of squared differences of the reconstruction loss observed in the interval $[t_1, t_2]$, $ssd_{t_1,t_2} = \sum_{i=t_1}^{t_2} (L_i - \hat{\mu}_{t_1,t_2})^2$.
A_{t_1,t_2}	An aggregate of the time interval $[t_1, t_2]$, $A_{t_1,t_2} = (\hat{\mu}_{t_1,t_2}, ssd_{t_1,t_2})$.
W	A window; internal data structure of ABCD, $W = \{w_1, w_2, \ldots, w_t\}$.
w_i	An item in W, $w_i = (A_{1,i}, \hat{\mathbf{x}}_i, \mathbf{x}_i)$.
τ_{max}	The number of possible change points evaluated in each time step.
n_{min}	The minimum number of samples to observe before training ϕ and ψ.
n_{max}	The maximum size of W before dropping old aggregates.

A. Appendix

A.1. Bandit Feedback

A.1.1. Proof of Theorem 3.4

Recall Theorem 3.4:

Theorem 3.4 (Number of suboptimal plays)**.** For ω-UCB, the expected number of plays of a suboptimal arm $k > 1$ before time step τ, $\mathbb{E}[n_k(\tau)]$, is upper-bounded by

$$\mathbb{E}[n_k(\tau)] \leq 1 + n_k^*(\tau) + \xi(\tau, \rho), \tag{3.7}$$

where

$$\xi(\tau, \rho) = (\tau - K)\left(2 - \sqrt{1 - \tau^{-\rho}}\right) - \sum_{\mathrm{t=K+1}}^{\tau} \sqrt{1 - t^{-\rho}}, \tag{3.8}$$

$$n_k^*(\tau) = \frac{8\rho \log \tau}{\delta_k^2} \max\left\{\frac{\eta_k^r \mu_k^r}{1 - \mu_k^r}, \frac{\eta_k^c(1 - \mu_k^c)}{\mu_k^c}\right\}, \tag{3.9}$$

$\delta_k = \Delta_k/(\Delta_k + 1/\mu_k^c)$, and K and Δ_k are defined as before, cf. Section 3.2.

Proof. The proof starts with a general expression for the number of plays of a suboptimal arm $k > 1$, where $\mathbb{1}\{\cdot\}$ denotes the indicator function.

$$n_k(\tau) \leq 1 + \sum_{\mathrm{t=K+1}}^{\tau} \mathbb{1}\left\{\Omega_k(t) \geq \Omega_j(t), \forall j \neq k\right\} \leq 1 + \sum_{\mathrm{t=K+1}}^{\tau} \mathbb{1}\left\{\Omega_k(t) \geq \Omega_1(t)\right\}$$

This is upper-bounded by

$$n_k(\tau) \leq 1 + \sum_{\text{t=K+1}}^{\tau} \left[\mathbb{1}\left\{ \Omega_k(t) \geq \Omega_1(t), \Omega_1(t) < \frac{\mu_1^r}{\mu_1^c} \right\} + \mathbb{1}\left\{ \Omega_k(t) \geq \Omega_1(t), \Omega_1(t) \geq \frac{\mu_1^r}{\mu_1^c} \right\} \right]$$
$$\leq 1 + \sum_{\text{t=K+1}}^{\tau} \left[\mathbb{1}\left\{ \Omega_1(t) < \frac{\mu_1^r}{\mu_1^c} \right\} + \mathbb{1}\left\{ \Omega_k(t) \geq \frac{\mu_1^r}{\mu_1^c} \right\} \right]$$

The expected number of plays $\mathbb{E}[n_k(\tau)]$ is given by the probabilities of the individual events:

$$\mathbb{E}[n_k(\tau)] \leq 1 + \sum_{\text{t=K+1}}^{\tau} \left[\underbrace{\Pr\left\{ \Omega_1(t) < \frac{\mu_1^r}{\mu_1^c} \right\}}_{\Pr[A]} + \underbrace{\Pr\left\{ \Omega_k(t) \geq \frac{\mu_1^r}{\mu_1^c} \right\}}_{\Pr[B]} \right] \tag{A.1}$$

We now evaluate the sum in the equation above.

Sum of $\Pr[A]$. We apply Theorem 3.2:

$$\sum_{\text{t=K+1}}^{\tau} \Pr[A] < \sum_{\text{t=K+1}}^{\tau} \left[1 - \sqrt{1 - t^{-\rho}} \right] = (\tau - K) - \sum_{\text{t=K+1}}^{\tau} \sqrt{1 - t^{-\rho}} \tag{A.2}$$

Sum of $\Pr[B]$. For this step, let us first introduce a helpful lemma: Lemma A.1 bounds the probability that $\Omega_k(t) \geq \mu_1^r/\mu_1^c$ after a minimum number of plays $n_k^*(\tau)$, which is logarithmic w.r.t. τ.

Lemma A.1 (Bound on $\Pr(\Omega_k(t) \geq \mu_1^r/\mu_1^c)$)**.** *Define* $\delta_k = \frac{\Delta_k}{\Delta_k + 1/\mu_k^c}$ *and* $n_k^*(\tau)$ *as follows:*

$$n_k^*(\tau) = \frac{8\rho \log \tau}{{\delta_k}^2} \max\left\{ \frac{\eta_k^r \mu_k^r}{1 - \mu_k^r}, \frac{\eta_k^c (1 - \mu_k^c)}{\mu_k^c} \right\}$$

The following inequality holds whenever $n_k(t) \geq n_k^*(\tau)$*:*

$$\Pr\left[\Omega_k(t) \geq \frac{\mu_1^r}{\mu_1^c} \right] < 1 - \sqrt{1 - \tau^{-\rho}}$$

Appendix A.1.3 contains the proof of Lemma A.1, Appendix A.1.2 the derivation of δ_k.

The lemma allows to decompose $\sum_{\mathrm{t=K+1}}^{\tau} \Pr[B]$ into 'initial plays' ($n_k(t) < n_k^*(\tau)$) and 'later plays' ($n_k(t) \geq n_k^*(\tau)$):

$$\begin{aligned}\sum_{\mathrm{t=K+1}}^{\tau} \Pr[B] &= \sum_{\mathrm{t=K+1}}^{\tau} \Pr\left\{\Omega_k(t) \geq \frac{\mu_1^r}{\mu_1^c}\right\} \\ &= n_k^*(\tau) + \sum_{\mathrm{t=K+1}}^{\tau} \Pr\left\{\Omega_k(t) \geq \frac{\mu_1^r}{\mu_1^c}, n_k(t) \geq n_k^*(\tau)\right\}\end{aligned}$$

We now apply Lemma A.1 to evaluate the sum in above equation,

$$\begin{aligned}\sum_{\mathrm{t=K+1}}^{\tau} \Pr[B] &\leq n_k^*(\tau) + \sum_{\mathrm{t=K+1}}^{\tau} \left[1 - \sqrt{1-\tau^{-\rho}}\right] \\ &= n_k^*(\tau) + (\tau - K)\left(1 - \sqrt{1-\tau^{-\rho}}\right). \end{aligned} \tag{A.3}$$

Inserting the results of Eq. (A.2) and Eq. (A.3) in Eq. (A.1) yields a bound on $\mathbb{E}[n_k(\tau)]$:

$$\mathbb{E}[n_k(\tau)] \leq 1 + n_k^*(\tau) + \underbrace{(\tau - K)\left(2 - \sqrt{1-\tau^{-\rho}}\right) - \sum_{\mathrm{t=K+1}}^{\tau} \sqrt{1-t^{-\rho}}}_{\xi(\tau,\rho)}$$

□

A.1.2. Derivation of Proportional δ-Gap

Let δ_k be the proportional δ-gap of arm k. It measures how much one must increase μ_k^r and decrease μ_k^c to make arm k have the same reward-cost ratio as arm 1, or in other words, to bridge the suboptimality gap Δ_k between arm k and arm 1. The δ-gap is proportional to the possible increase in rewards (decrease in costs) without violating their range $[0, 1]$. We start our derivation with Eq. (A.4), which states this mathematically.

$$\frac{\mu_k^r + \delta_k(1-\mu_k^r)}{\mu_k^c - \delta_k \mu_k^c} = \frac{\mu_1^r}{\mu_1^c} \tag{A.4}$$

Rearranging the equation yields

$$\frac{\mu_1^r}{\mu_1^c} - \frac{\mu_k^r}{\mu_k^c} = \delta_k \left(\frac{\mu_1^r}{\mu_1^c} - \frac{\mu_k^r}{\mu_k^c} + \frac{1}{\mu_k^c}\right).$$

Now, recall an arm's suboptimality, $\Delta_k = \mu_1^r/\mu_1^c - \mu_k^r/\mu_k^c$, and solve for δ_k:

$$\delta_k = \frac{\Delta_k}{\Delta_k + \frac{1}{\mu_k^c}}$$

A.1.3. Proof of Lemma A.1

Recall Lemma A.1:

Lemma A.1 (Bound on $\Pr(\Omega_k(t) \geq \mu_1^r/\mu_1^c)$)**.** *Define* $\delta_k = \frac{\Delta_k}{\Delta_k + 1/\mu_k^c}$ *and* $n_k^*(\tau)$ *as follows:*

$$n_k^*(\tau) = \frac{8\rho \log \tau}{{\delta_k}^2} \max\left\{\frac{\eta_k^r \mu_k^r}{1-\mu_k^r}, \frac{\eta_k^c(1-\mu_k^c)}{\mu_k^c}\right\}$$

The following inequality holds whenever $n_k(t) \geq n_k^*(\tau)$*:*

$$\Pr\left[\Omega_k(t) \geq \frac{\mu_1^r}{\mu_1^c}\right] < 1 - \sqrt{1-\tau^{-\rho}}$$

Proof. The proof is structured in four steps. First, we derive an expression for the maximum deviation between the observed sample mean and the unknown expected value of an arm's rewards and costs that we use later on. Second, we decompose the probability that $\Omega_k(t) \geq \mu_1^r/\mu_1^c$. Third, we evaluate the decomposed probabilities for cases where $n_k(t)$ is sufficiently large, that is, $n_k(t) \geq n_k^*(\tau)$. Finally, we recombine the decomposed probabilities to obtain the final result.

Deviation between sample mean and expected value. Let $\hat{\mu}_k(t)$ be the sample mean and μ_k the expected value of arm k's rewards or costs at time t. To quantify the deviation between mean $\hat{\mu}_k(t)$ and μ_k we start from Eq. (3.5) (central limit theorem) and set $[m, M] = [0, 1]$ and $n = n_k(t)$):

$$(\hat{\mu}_k(t) - \mu_k)^2 \leq \frac{\eta_k \mu_k(1-\mu_k)}{n_k(t)} z^2 \tag{A.5}$$

The above inequality holds with the same probability as our confidence interval since it is the basis of the confidence interval derivation. This allows us to bound the deviation between the sample mean and expected value, denoted $\varepsilon_k(t)$, for our choice of $z_\rho(t) = \sqrt{2\rho \log t}$:

$$\Pr\left[|\hat{\mu}_k(t) - \mu_k| > \varepsilon_k(t)\right] \leq \alpha(t) \tag{A.6}$$

with

$$\varepsilon_k(t) = \sqrt{\frac{2\eta_k \mu_k(1-\mu_k)\rho \log t}{n_k(t)}} \quad \text{and} \quad \alpha(t) < 1 - \sqrt{1-t^{-\rho}}.$$

Whenever we refer to $\varepsilon_k(t)$ w.r.t. rewards or costs we use the notations $\varepsilon_k^r(t)$ and $\varepsilon_k^c(t)$.

Decomposition of $\Pr[\Omega_k(t) \geq \mu_1^r/\mu_1^c]$. Next, we decompose the probability that $\Omega_k(t) \geq \mu_1^r/\mu_1^c$. The decomposition is analogous to the one in the proof of Theorem 3.2 and thus omitted here:

$$\Pr\left[\Omega_k(t) \geq \frac{\mu_1^r}{\mu_1^c}\right] = \Pr\left[\frac{\omega_{k+}^r(t)}{\omega_{k-}^c(t)} \geq \frac{\mu_1^r}{\mu_1^c}\right] = \Pr\left[\frac{\omega_{k+}^r(t)}{\omega_{k-}^c(t)} \geq \frac{\mu_k^r + (1-\mu_k^r)\delta_k}{\mu_k^c - \mu_k^c\delta_k}\right] \tag{A.7}$$

$$\leq \Pr\left[\omega_{k+}^r(t) \geq \mu_k^r + (1-\mu_k^r)\delta_k\right] + \Pr\left[\omega_{k-}^c(t) \leq \mu_k^c - \mu_k^c\delta_k\right] \tag{A.8}$$

For the next step, note that if $\mu_k^r \leq \omega_{k+}^r(t)$ (and $\mu_k^c \geq \omega_{k-}^c(t)$), we have that $\mu_k^r - \hat{\mu}_k^r(t) \leq \varepsilon_k^r(t)$ (and $\hat{\mu}_k^c(t) - \mu_k^c \leq \varepsilon_k^c(t)$). This allows us to rewrite the terms in Eq. (A.8) as follows:

$$\begin{aligned}\Pr\left[\omega_{k+}^r(t) \geq \mu_k^r + (1-\mu_k^r)\delta_k\right] &= \Pr[\mu_k^r + (1-\mu_k^r)\delta_k - \hat{\mu}_k^r(t) \leq \varepsilon_k^r(t)] \\ &= \Pr[\hat{\mu}_k^r(t) - \mu_k^r \geq (1-\mu_k^r)\delta_k - \varepsilon_k^r(t)]\end{aligned} \tag{A.9}$$

$$\begin{aligned}\Pr\left[\omega_{k-}^c(t) \leq \mu_k^c - \mu_k^c\delta_k\right] &= \Pr[\hat{\mu}_k^c(t) - (\mu_k^c - \mu_k^c\delta_k) \leq \varepsilon_k^c(t)] \\ &= \Pr[\mu_k^c - \hat{\mu}_k^c(t) \geq \mu_k^c\delta_k - \varepsilon_k^c(t)]\end{aligned} \tag{A.10}$$

In the next two paragraphs, we evaluate Eq. (A.9) (confidence bound of rewards) and Eq. (A.10) (confidence bound of costs). We combine both results afterwards.

Evaluation of Eq. (A.9) for $n_k(t) \geq n_k^{*,r}(\tau)$. We now consider the cases in which the number of times arm k was played is at least logarithmic w.r.t. τ, i.e., $n_k(t) \geq n_k^{*,r}(\tau)$ with

$$n_k^{*,r}(\tau) = \frac{2\rho\log\tau}{{\delta_k}^2(1-\kappa)^2}\frac{\eta_k^r\mu_k^r}{1-\mu_k^r}, \quad \text{for any } \kappa \in (0,1).$$

In those cases, $\varepsilon_k^r(t) \leq (1-\kappa)\delta_k(1-\mu_k^r)$. One can verify this by inserting $n_k^{*,r}(\tau)$ in the definition of $\varepsilon_k^r(t)$, cf. Eq. (A.6). This gives the following inequality for the right side of Eq. (A.9):

$$\Pr\left[\omega_{k+}^r(t) \geq \mu_k^r + (1-\mu_k^r)\delta_k\right] \leq \Pr[\hat{\mu}_k^r(t) - \mu_k^r \geq \kappa(1-\mu_k^r)\delta_k]$$

Last, we compute the number of standard deviations z^* that corresponds to a deviation between $\hat{\mu}_k^r(t)$ and μ_k^r of at maximum $\kappa(1-\mu_k^r)\delta_k$ based on Eq. (A.5). In particular, we solve the right-most inequality in the expression below:

$$(\hat{\mu}_k^r(t) - \mu_k^r)^2 \leq \frac{\eta_k^r\mu_k^r(1-\mu_k^r)}{n_k(t)}z^{*2} \leq \frac{\eta_k^r\mu_k^r(1-\mu_k^r)}{n_k^{*,r}(\tau)}z^{*2} \leq (\kappa(1-\mu_k^r)\delta_k)^2$$

With our choice of $n_k^{*,r}(\tau)$, this yields $z^* = (2\rho \log \tau \kappa^2 (1-\kappa)^{-2})^{\frac{1}{2}}$. Inserting z^* in Eq. (3.6) (upper bound for $\alpha(t)$) results in the following bound:

$$\Pr\left[\omega_{k+}^r(t) \geq \mu_k^r + (1-\mu_k^r)\delta_k\right] < \frac{1}{2}\left(1 - \sqrt{1 - \tau^{-\frac{\kappa^2\rho}{(1-\kappa)^2}}}\right) \tag{A.11}$$

Evaluation of Eq. (A.10) for $n_k(t) > n_k^{*,c}(\tau)$. Again, we consider the cases in which the number of times arm k was played is at least logarithmic in τ, i.e., $n_k(t) \geq n_k^{*,c}(\tau)$ with

$$n_k^{*,c}(\tau) = \frac{2\rho \log \tau}{{\delta_k}^2 (1-\kappa)^2} \frac{\eta_k^c(1-\mu_k^c)}{\mu_k^c}, \quad \kappa \in (0,1).$$

In those cases, $\varepsilon_k^c(t) \leq (1-\kappa)\delta_k \mu_k^c$. Following analogous steps as in the previous paragraph yields Eq. (A.12):

$$\Pr\left[\omega_{k-}^c(t) \leq \mu_k^c - \mu_k^c\delta_k\right] < \frac{1}{2}\left(1 - \sqrt{1 - \tau^{-\frac{(1-\kappa)^2\rho}{\kappa^2}}}\right) \tag{A.12}$$

Obtaining the final result. With the results in Eq. (A.11) and Eq. (A.12) we can finally evaluate Eq. (A.7). A choice of $\kappa = 0.5$ and

$$n_k^*(\tau) = \frac{8\rho \log \tau}{{\delta_k}^2} \max\left\{\frac{\eta_k^r \mu_k^r}{1-\mu_k^r}, \frac{\eta_k^c(1-\mu_k^c)}{\mu_k^c}\right\}$$

yields the bound given in Lemma A.1:

$$\Pr\left[\Omega_k(t) \geq \frac{\mu_1^r}{\mu_1^c}\right] < 1 - \sqrt{1-\tau^{-\rho}}, \quad n_k(t) \geq n_k^*(\tau)$$

□

A.1.4. Proof of Theorem 3.6

Recall Theorem 3.6:

Theorem 3.6 (Asymptotic regret)**.** The regret of ω-UCB is in

$$\text{Regret} \in \begin{cases} \mathcal{O}\left(B^{1-\rho}\right), & \text{if } 0 < \rho < 1, \\ \mathcal{O}(\log B), & \text{if } \rho \geq 1. \end{cases}$$

Proof. First note that the latter two terms and $n_k^*(\tau_B)$ in Eq. (3.10) are in $\mathcal{O}(\log B)$ [Xia+17]. Next we show that $\xi(\tau, \rho)$ is in $\mathcal{O}(\log B)$ for $\rho \geq 1$ and in $\mathcal{O}(B^{1-\rho})$ for $0 < \rho < 1$.

We exploit two inequalities in our proof; the latter is based on the integral test for convergence and holds for continuous, positive, decreasing functions.

$$\sqrt{1-t^{-\rho}} \geq 1-t^{-\rho}, \quad t \geq 1, \rho > 0 \tag{A.13}$$

$$\sum_{t=K+1}^{\tau_B} t^{-\rho} \leq (K+1)^{-\rho} + \int_{t=K+1}^{\tau_B} t^{-\rho}\, dt \tag{A.14}$$

We use Eq. (A.13) to obtain an integrable expression for the sum in $\xi(\tau, \rho)$. We replace the sum with an integral-based upper bound as in Eq. (A.14):

$$\begin{aligned}
\xi(\tau,\rho) &= (\tau_B - K)\left(2 - \sqrt{1-{\tau_B}^{-\rho}}\right) - \sum_{t=K+1}^{\tau_B} \sqrt{1-t^{-\rho}} \\
&\leq (\tau_B - K)\left(2 - \sqrt{1-{\tau_B}^{-\rho}}\right) - \sum_{t=K+1}^{\tau_B} 1 - t^{-\rho} \\
&= (\tau_B - K)\left(1 - \sqrt{1-{\tau_B}^{-\rho}}\right) + \sum_{t=K+1}^{\tau_B} t^{-\rho} \\
&\leq (\tau_B - K)\left(1 - \sqrt{1-{\tau_B}^{-\rho}}\right) + (K+1)^{-\rho} + \int_{t=K+1}^{\tau_B} t^{-\rho}\, dt
\end{aligned}$$

Next, we evaluate the integral for the cases $\rho = 1$ and $\rho \neq 1$.

Case 1: $\rho = 1$.

$$\xi(\tau,\rho) \leq (\tau_B - K)\left(1 - \sqrt{1-{\tau_B}^{-1}}\right) + (K+1)^{-1} + \log \tau_B - \log(K+1)$$

For $\rho = 1$, the first term in above equation converges, so $\xi(\tau_B, \rho = 1)$ is in $\mathcal{O}(\log B)$. This implies that the overall regret of ω-UCB is in $\mathcal{O}(\log B)$ for $\rho = 1$.

Case 2: $\rho \neq 1$.

$$\xi(\tau,\rho) \leq (\tau_B - K)\left(1 - \sqrt{1-{\tau_B}^{-\rho}}\right) + (K+1)^{-\rho} + \frac{1}{1-\rho}\left({\tau_B}^{1-\rho} - (K+1)^{1-\rho}\right)$$

For $\rho > 1$, $\xi(\tau, \rho)$ converges and thus the overall regret is in $\mathcal{O}(\log B)$. For $0 < \rho < 1$, one can show that $\xi(\tau, \rho)$ is in $\mathcal{O}(B^{1-\rho})$. Hence, the regret of ω-UCB is in $\mathcal{O}(B^{1-\rho})$ for $0 < \rho < 1$. Combining cases 1 and 2 yield the statement given in Theorem 3.6. □

A.1.5. Related UCB Approaches

Table A.1 summarizes our direct competitors and how they compute the arm selection indexes $\Omega_k(t)$.

Table A.1.: Overview of BMAB algorithms

Policy	$\Omega_k(t)$	Comment
BTS [Xia+15b]	$\frac{\text{sample from Beta}(\alpha_r(t),\beta_r(t))}{\text{sample from Beta}(\alpha_c(t),\beta_c(t))}$	$\alpha_r(t) = n_k(t)\hat{\mu}_k^r(t) + 1$ $\beta_r(t) = n_k(t) + 2 - \alpha_r(t)$ $\alpha_c(t) = n_k(t)\hat{\mu}_k^c(t) + 1$ $\beta_c(t) = n_k(t) + 2 - \alpha_c(t)$ Discretization of continuous values
m-UCB [Xia+17]	$\frac{\min\{\hat{\mu}_k^r(t)+\varepsilon_k(t),1\}}{\max\{\hat{\mu}_k^c(t)-\varepsilon_k(t),0\}}$	$\varepsilon_k(t) = \alpha\sqrt{\frac{\log(t-1)}{n_k(t)}}$ Recommendation: $\alpha = 2^{-4}$
c-UCB [Xia+17]	$\frac{\hat{\mu}_k^r(t)}{\hat{\mu}_k^c(t)} + \frac{\varepsilon_k(t)}{\hat{\mu}_k^c(t)}$	$\varepsilon_k(t) = \alpha\sqrt{\frac{\log(t-1)}{n_k(t)}}$ Recommendation: $\alpha = 2^{-3}$
i-UCB [Xia+17]	$\frac{\hat{\mu}_k^r(t)}{\hat{\mu}_k^c(t)} + \varepsilon_k(t)$	$\varepsilon_k(t) = \alpha\sqrt{\frac{\log(t-1)}{n_k(t)}}$ Recommendation: $\alpha = 2^{-2}$
Budget-UCB [Xia+15a]	$\frac{\hat{\mu}_k^r(t)}{\hat{\mu}_k^c(t)} + \frac{\varepsilon_k(t)}{\hat{\mu}_k^c(t)}\left(1 + \frac{\min\{\hat{\mu}_k^r(t)+\varepsilon_k(t),1\}}{\max\{\hat{\mu}_k^c(t)-\varepsilon_k(t),\lambda\}}\right)$	$\varepsilon_k(t) = \sqrt{\frac{\log(t-1)}{n_k(t)}}$ $\lambda > 0$: minimum cost
UCB-SC+ [Wat+18]	$\frac{\hat{\mu}_k^r(t)+\alpha_k(t)\hat{\mu}_k^c(t)}{\hat{\mu}_k^c(t)-\alpha_k(t)\hat{\mu}_k^r(t)}$, if $\hat{\mu}_k^c(t)^2 > \frac{\log\frac{t}{n_k(t)}}{2n_k(t)}$ ∞, else	$\alpha_k(t) = \sqrt{\frac{\log\frac{t}{n_k(t)}}{2\kappa n_k(t) - \log\frac{t}{n_k(t)}}}$ with $\kappa = \hat{\mu}_k^r(t)^2 + \hat{\mu}_k^c(t)^2$
UCB-B2 [CES20]	$1.4\frac{\epsilon_{k,t}+\hat{r}_{k,t}}{\hat{\mu}_k^c(t)} + \hat{r}_{k,t}$ if condition 7 in [CES20] ∞, else	$\hat{r}_{k,t} = \frac{\max\{0,\hat{\mu}_k^r(t)\}}{\max\{\text{minimum cost},\hat{\mu}_k^c(t)\}}$ $\epsilon_{k,t} = \sqrt{\frac{2\hat{V}_{k,t}^r \log t^\alpha}{n_k(t)}} + \frac{3\log t^\alpha}{n_k(t)}$ $\eta_{k,t} = \sqrt{\frac{2\hat{V}_{k,t}^c \log t^\alpha}{n_k(t)}} + \frac{3\log t^\alpha}{n_k(t)}$ $\hat{V}_{k,t}^r, \hat{V}_{k,t}^c$: variance estimates $\alpha > 2$; small choices preferable

A.2. Observation-Based Feedback

A.2.1. Additional Pseudocode for PLASTIC

Algorithm A.1 contains the complete pseudocode of PLASTIC's restructuring procedure. See also Section 4.3.2 in the paper.

Algorithm A.1 PLASTIC – all procedures

```
1: procedure DecoupleBranches(queues Q and F, maximum depth λ, desired split attribute a* and split value v*)
2:     for b ∈ Q do
3:         (lastNode, –) ← b
4:         isFinished ← lastNode splits at a* or b.length ≥ λ
5:         if isFinished then
6:             Force desired split at lastNode if necessary
7:             newBranches ← ExtendBranch(b)
8:             Move 2nd-last element in b to position 0
9:             F.add(newBranches)
10:            Q.remove(b)
11:            return
12:        else
13:            newBranches ← ExtendBranch(b)
14:            Q.remove(b)
15:            Q.add(newBranches)

16: procedure AddBranchToTree(root copy r', branch b)
17:    current node c ← r'
18:    currentDepth ← 0
19:    while currentDepth < b.length do
20:        (node n, key v) ← b[currentDepth]
21:        if c and n have the same split attribute then
22:            if key v already in successors of c then                ▷ Move to successor
23:                c ← get successor of c for key v
24:            else                ▷ Add new successor to c, then move to new successor
25:                (newSuccessor, –) ← b[currentDepth + 1]
26:                Add newSuccessor to c with key v
27:                c ← newSuccessor
28:        Increment currentDepth

29: procedure ExtendBranch(branch b)
30:    newBranches ← {}
31:    l ← last node in b
32:    for all child keys v_i of l do
33:        newBranches.add(b + (l, v_i))
34:    return newBranches
```

A.3. Generating Feedback

A.3.1. Autoencoder Training in ABCD

Algorithm A.2 describes the training of the autoencoder model as done in our experiments. First, we collect the training data from the current window W (line 2). Afterwards, we perform gradient descent on X_{train} for E epochs at a learning rate of lr.

Algorithm A.2 Training procedure of the AE in ABCD

Require: window W, learning rate lr, number of training epochs E

1: **procedure** TrainAE(W, lr, E)
2: $X_{train} \leftarrow \{\mathbf{x}_i \ : \ (-,-,\mathbf{x}_i) \in W\}$
3: $\phi, \psi \leftarrow$ NewEncoder(), NewDecoder()
4: **for all** epochs E **do**
5: GradientDescent($\psi \circ \phi, X_{train}, lr$)
6: **return** ϕ, ψ

A.3.2. Detectable and Undetectable Change for ABCD (pca)

This section illustrates two kinds of changes which ABCD (pca) is able and unable to detect. Figure A.1 shows data from two distributions: black points (e.g., before the change) plus the associated main principle component, and green points (e.g., after the change). On the left, the change affects the correlation between dimension 1 and dimension 2. This leads to an increased reconstruction error for the points highlighted in green. On the right, the change occurs along the main principle component. I.e., the variance along that component has increased. Such a change is undetectable by ABCD (pca) as the new data concept does not affect the reconstruction error.

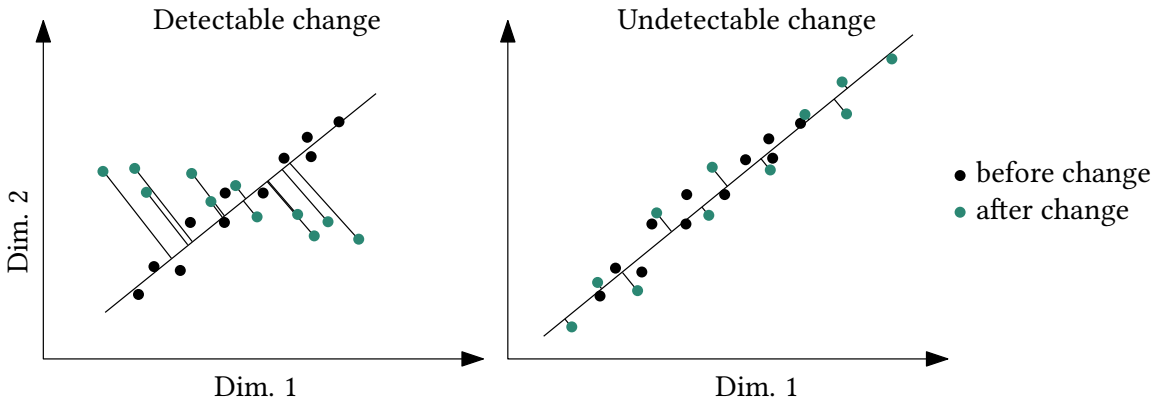


Figure A.1.: Illustration of detectable and undetectable change using PCA