Towards Reliable Machine Learning in Evolving Data Streams

Towards Reliable Machine Learning in Evolving Data Streams

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Johannes Christian Haug
aus Tübingen

Tübingen
2022

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	05.09.2022
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Gjergji Kasneci
2. Berichterstatter:	Prof. Dr. Philipp Hennig

Abstract

Data streams are ubiquitous in many areas of modern life. For example, applications in healthcare, education, finance, or advertising often deal with large-scale and evolving data streams. Compared to stationary applications, data streams pose considerable additional challenges for automated decision making and machine learning. Indeed, online machine learning methods must cope with limited memory capacities, real-time requirements, and drifts in the data generating process. At the same time, online learning methods should provide a high predictive quality, stability in the presence of input noise, and good interpretability in order to be reliably used in practice. In this thesis, we address some of the most important aspects of machine learning in evolving data streams. Specifically, we identify four open issues related to online feature selection, concept drift detection, online classification, local explainability, and the evaluation of online learning methods. In these contexts, we present new theoretical and empirical findings as well as novel frameworks and implementations. In particular, we propose new approaches for online feature selection and concept drift detection that can account for model uncertainties and thus achieve more stable results. Moreover, we introduce a new incremental decision tree that retains valuable interpretability properties and a new change detection framework that allows for more efficient explanations based on local feature attributions. In fact, this is one of the first works to address intrinsic model interpretability and local explainability in the presence of incremental updates and concept drift. Along with this thesis, we provide extensive open resources related to online machine learning. Notably, we introduce a new Python framework that enables simplified and standardized evaluations and can thus serve as a basis for more comparable online learning experiments in the future. In total, this thesis is based on six publications, five of which were peer-reviewed at the time of publication of this thesis. Our work touches all major areas of predictive modeling in data streams and proposes novel solutions for efficient, stable, interpretable and thus reliable online machine learning.

Kurzfassung

Datenströme sind in vielen Bereichen des modernen Lebens allgegenwärtig. Beispielsweise haben Anwendungen im Gesundheitswesen, im Bildungswesen, im Finanzwesen oder in der Werbung häufig mit großen und sich verändernden Datenströmen zu tun. Im Vergleich zu stationären Anwendungen stellen Datenströme eine erhebliche zusätzliche Herausforderung für die automatisierte Entscheidungsfindung und das maschinelle Lernen dar. So müssen Online Machine Learning-Verfahren mit begrenzten Speicherkapazitäten, Echtzeitanforderungen und Veränderungen des Daten-generierenden Prozesses zurechtkommen. Gleichzeitig sollten Online Learning-Verfahren eine hohe Vorhersagequalität, Stabilität bei Eingangsrauschen und eine gute Interpretierbarkeit aufweisen, um in der Praxis zuverlässig eingesetzt werden zu können. In dieser Arbeit befassen wir uns mit einigen der wichtigsten Aspekte des maschinellen Lernens in sich entwickelnden Datenströmen. Insbesondere identifizieren wir vier offene Fragen im Zusammenhang mit Online Feature Selection, Concept Drift Detection, Online-Klassifikation, lokaler Erklärbarkeit und der Bewertung von Online Learning-Methoden. In diesem Kontext präsentieren wir neue theoretische und empirische Erkenntnisse sowie neue Frameworks und Implementierungen. Insbesondere schlagen wir neue Ansätze für Online Feature Selection und Concept Drift Detection vor, die Unsicherheiten im Modell berücksichtigen und dadurch stabilere Ergebnisse erzielen können. Darüber hinaus stellen wir einen neuen inkrementellen Entscheidungsbaum vor, der wertvolle Eigenschaften hinsichtlich der Interpretierbarkeit einhält, sowie ein neues Framework zur Erkennung von Veränderungen, das effizientere Erklärungen auf der Grundlage lokaler Feature Attributions ermöglicht. Tatsächlich ist dies eine der ersten Arbeiten, die sich mit intrinsischer Interpretierbarkeit von Modellen und lokaler Erklärbarkeit bei inkrementellen Aktualisierungen und Concept Drift befasst. Gemeinsam mit dieser Arbeit stellen wir umfangreiche Ressourcen für Online Machine Learning zur Verfügung. Insbesondere stellen wir ein neues Python-Framework vor, das vereinfachte und standardisierte Auswertungen ermöglicht und künftig somit als Grundlage für vergleichbare Online Learning-Experimente dienen kann. Insgesamt stützt sich diese Arbeit auf sechs Publikationen, von denen fünf zum Zeitpunkt der Veröffentlichung der Dissertation bereits im Peer-Review Format begutachtet wurden. Unsere Arbeit berührt alle wichtigen Bereiche der prädiktiven Modellierung in Datenströmen und schlägt neuartige Lösungen für effizientes, stabiles, interpretierbares und damit zuverlässiges Online Machine Learning vor.

Acknowledgments

At the time I am writing this thesis, it has almost been four years since I joined the Data Science and Analytics group. This time has been one of the most challenging, but also most fulfilling periods of my life. I have met many new people along the way and have been supported by colleagues, friends, and family members to whom I am forever grateful.

Most of all, I would like to thank my supervisor, Gjergji Kasneci. Gjergji put in an extraordinary amount of time and effort to help me develop as a researcher. He was always there when I needed advice and helped me stay on top of things during particularly difficult times. At the same time, Gjergji gave me a lot of freedom to experiment and develop my own ideas. It is mainly thanks to Gjergji that I will always have fond memories of my time as a PhD student, for which I am deeply indebted to him.

I would also like to thank all my colleagues who accompanied me during the time at the University of Tübingen. In particular, I would like to thank Martin Pawelczyk, Vadim Borisov, Hamed Jalali, and Tobias Leemann, with whom I had many interesting discussions about research, but who also made my time in the office much more entertaining. Special thanks also go to Klaus Broelemann, who improved many of my papers by helping me sharpen my understanding of clean and concise formalizations. Finally, I thank all my other colleagues in the Data Science and Analytics group and the HCI chair, as well as all the students I have met and worked with over the years.

After an eventful week, I always look forward to spending the weekend with my friends from the Rottenburger Domsingknaben. Although we do not always engage in the most relaxing activities, spending time with my friends has given me some much-needed distance from work and kept me motivated over the past few years. In that sense, special thanks go to Lukas, Josch, Cornel, Bifi, Yannick, Janis, and Patrick. A tiny thanks also goes to Alois, who always cheered me up after a long day at work.

The past years have been quite eventful and challenging for my entire family. Yet, I could always count on the support of my grandpas Sieger and Oskar, my brother Markus and especially my parents Jürgen and Sylvia. There are many things that are more important than academic results, and I am happy to say that this year has brought great news and achievements for all of us. I could not be more grateful for my family.

Finally, I would like to thank Katha, who became my wife in 2020. Although she clearly regretted that decision before paper deadlines, she never stopped motivating me. Without her support, this PhD would not have been possible. Katha has made my life much more exciting and joyful over the past ten years, and I look forward to what awaits us in the future.

Contents

List of Publications

This thesis is based on six publications, five of which have been accepted and one of which was under review at the time of publication of this thesis.

Accepted Publications

Paper 1

Johannes Haug, Martin Pawelczyk, Klaus Broelemann, and Gjergji Kasneci. 2020. **"Leveraging Model Inherent Variable Importance for Stable Online Feature Selection"**. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2020). `https://doi.org/10.1145/3394486.3403200`. [78]

Paper 2

Johannes Haug and Gjergji Kasneci. 2021. **"Learning Parameter Distributions to Detect Concept Drift in Data Streams"**. In Proceedings of the 25th International Conference on Pattern Recognition (ICPR 2020). `https://doi.org/10.1109/ICPR48806.2021.9412499`. [77]

Paper 3

Johannes Haug, Stefan Zürn, Peter El-Jiz, and Gjergji Kasneci. 2021. **"On Baselines for Local Feature Attributions"**. Explainable Agency in Artificial Intelligence Workshop at the 35th AAAI Conference on Artificial Intelligence (AAAI 2021). `https://doi.org/10.48550/arXiv.2101.00905` (workshop proceedings at `https://sites.google.com/view/xaiworkshop/proceedings`). [79]

Paper 4

Johannes Haug, Klaus Broelemann, and Gjergji Kasneci. 2022. **"Dynamic Model Tree for Interpretable Data Stream Learning"**. In Proceedings of the 38th IEEE International Conference on Data Engineering (ICDE 2022). `https://doi.org/10.1109/ICDE53745.2022.00237`. [81]

Paper 5

Johannes Haug, Alexander Braun, Stefan Zürn, and Gjergji Kasneci. 2022. **"Change Detection for Local Explainability in Evolving Data Streams"**. In Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM 2022). https://doi.org/10.48550/arXiv.2209.02764 (the conference proceedings were not yet available at the time of publication of this thesis). [80]

Manuscript Under Review

Paper 6

Johannes Haug, Effi Tramountani, and Gjergji Kasneci. 2022. **"Standardized Evaluation of Machine Learning Methods for Evolving Data Streams"**. Under review at the Springer Journal of Big Data. https://doi.org/10.48550/arXiv.2204.13625. [82]

Additional Co-Authored Papers

During my time as a PhD student, I also contributed to a number of other publications that are not part of this thesis:

Vadim Borisov, Johannes Haug, and Gjergji Kasneci. 2019. **"Cancelout: A Layer for Feature Selection in Deep Neural Networks"**. In Proceedings of the 28th International Conference on Artificial Neural Networks (ICANN 2019). https://doi.org/10.1007/978-3-030-30484-3_6. [28]

Martin Pawelczyk, Johannes Haug, Klaus Broelemann, and Gjergji Kasneci. 2019. **"Towards User Empowerment"**. Workshop on Human-Centric Machine Learning at the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). An updated version without my participation was later accepted at the Web Conference 2020 (https://doi.org/10.1145/3366423.3380087). [132]

Enkelejda Kasneci, Gjergji Kasneci, Tobias Appel, Johannes Haug, Franz Wortha, Maike Tibus, Ulrich Trautwein, and Peter Gerjets. 2021. **"TüEyeQ, a Rich IQ Test Performance Data Set with Eye Movement, Educational and Socio-Demographic Information"**. Nature Scientific Data 8, 154 (2021). `https://doi.org/10.1038/s41597-021-00938-3`.[95]

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. 2021. **"Deep Neural Networks and Tabular Data: A Survey"**. Published on ArXiv. `https://doi.org/10.48550/arXiv.2110.01889`. [29]

Chapter 1

Introduction to Online Machine Learning

Evolutionary and incremental changes in data are abundant in real-world applications. We find them in biological and physical processes, sensory measurements, financial transactions and user behavior, in applications such as credit scoring, e-commerce, autonomous driving, fraud detection or social media [13, 8, 45, 58, 16, 126, 117, 11, 41]. In such dynamic environments, that operate with or generate continuous and evolving data streams, machine learning models face considerable challenges. This thesis addresses some of the most pressing issues on the path to more reliable online machine learning.

1.1 Challenges in Learning From Evolving Data Streams

Machine learning techniques are successfully used to process large amounts of data and automate decisions in a variety of applications. However, traditional machine learning approaches often employ a notion of stationary distributions that does not apply to real-world scenarios. In practice, training observations often arrive sequentially and over a potentially infinite period of time. Moreover, intrinsic process properties or external events may influence the data generating process and make previously learned concepts obsolete. For example, network attackers will constantly try to find new ways to overcome security measures, so an intrusion detection application must adapt accordingly [175]. Similarly, medical treatments may need to be adjusted to resistances and mutations that develop over time [175]. As a consequence, traditional batch learning techniques are often too inflexible to be applied to evolving data streams. For example, preprocessing techniques such as feature selection usually require the entire training data to be available in main memory. In addition, online applications typically work with limited resources. Therefore, powerful but complex architectures such as deep neural networks or large ensembles may not be feasible in a real streaming scenario [8].

In online machine learning, we aim to overcome these limitations. To this end, we require dedicated solutions for preprocessing incoming data [141], detecting and managing changes in the data generating distribution [108, 69], and providing powerful predictions over time.

The goal of a holistic online learning approach is to represent and adjust to the dynamics of a data stream well at any point in time. In this context, data stream applications impose extensive requirements [62, 63, 126, 58, 117, 48, 99]. Indeed, to achieve truly reliable machine learning in evolving data streams, a number of important abstract requirements must be met:

- **Predictive Performance:** To be reliably used for automated decision-making in data streams, online learning methods should achieve the desired predictive quality (i.e., according to expert-based and application-dependent quality measures) at all times.
- **Efficiency:** To meet real-time demands and deal with limited hardware capacities, online learning methods should be efficient in terms of computation time and computation resources.
- **Robustness:** In order to rely on the model and its predictions, online learning methods should be robust to noisy inputs, abrupt changes in data, and outliers.
- **Adaptation to Concept Drift:** As data streams can be subject to changing data distributions, online learning methods need to ensure that they can quickly adapt accordingly.
- **Interpretability and Explainability:** To enable trustworthy user-centered application of online learning methods and allow their use in sensitive areas with strict regulations, the mechanics of the model should be interpretable and any prediction should be explainable to a human.

It is generally difficult to address all requirements simultaneously. In fact, there are various trade-offs. For example, using sliding windows to store past observations often has advantages in terms of predictive power and robustness, but requires additional resources [8, 22]. Similarly, adjusting for distributional changes using heuristic mechanisms could reduce the predictive performance and interpretability. In practice, the importance of each requirement also often depends on the application at hand. For example, high stakes applications in healthcare or banking might prioritize accurate predictions and high interpretability over efficiency. Conversely, fraud detection and spam filtering usually aim for high throughput, accepting the risk of false alarms. Ideally, online learning approaches should be flexible enough to be adapted to the requirements of a particular application [99]. However, the weakly specified and dynamic nature of data streams makes reliable online machine learning a major challenge.

Compared to areas such as deep learning or reinforcement learning, online learning has lost momentum despite its impact on various applications. The state of the art in online learning has indeed not changed much in recent years, although existing approaches are known to have their limitations with regard to the requirements mentioned above. Therefore, there are a number of new and open problems that still need to be explored.

1.2 Objects of Study

Similar to other areas of machine learning, online learning involves many different activities. In this thesis, we address five particularly important ones: online feature selection, concept drift detection, online classification, local explainability in data streams, and the evaluation of online learning methods. Given the general desiderata and requirements introduced in Section 1.1, there are a number of open questions related to each of these activities [99, 141]. In this thesis, we address the following issues:

Online feature selection and concept drift detection techniques often do not account for high model uncertainty. Many online feature selection and concept drift detection methods rely on an underlying predictive model. As online predictive models evolve over time, they can be subject to a high degree of uncertainty, which manifests itself, for example, in drastic parameter updates or predictive quality degradation between time steps. Indeed, data streams are subject to various sources of uncertainty [44, 160] that may propagate to the online learning model. These include missing and adversarial data, unknown data distributions and concept drift, as well as noise. Existing online feature selection and concept drift detection methods often do not take this uncertainty into account, resulting in non-robust behavior.

Online classification approaches typically do not address the interpretability of the learned model over time. Due to increased public awareness and new legislation, such as the European Union's General Data Protection Regulation, the interpretability of machine learning models is becoming increasingly important [129, 32]. Intrinsic interpretability describes a comprehensible inner mechanics of the predictive model [51, 36]. Interpretability can thus not be measured objectively [116, 146] and is instead often linked to a heuristic quantification of model complexity [15, 71]. Accordingly, simple models such as linear models, rule-based learners, and shallow decision trees are widely considered to be interpretable [71]. The complexity of most online classifiers changes over time. For example, online ensembles typically remove old components and add new ones to adjust to concept drift [100, 67]. Similarly, incremental decision trees change by adding new leaf nodes and pruning or replacing outdated branches [46, 88, 18, 111]. However, the effect of such changes on interpretability is often neglected. As a result, many common online classifiers rely on heuristic updating procedures or become increasingly complex and thus do not provide the level of interpretability required in practice.

The behavior of feature attribution methods for local explainability in data streams is largely unexplored. Explanation methods for machine learning allow us to explain the predictions of complex, i.e., non-interpretable, models [71]. For example, we typically require a form of post-hoc explainability to understand the predictions of deep neural networks or large ensembles. Explanation methods can also be used as an effective tool for detecting bias in the data or model [129]. Local attribution methods are one of the most widely used post-hoc explanation techniques [96, 109, 110, 143, 151, 158]. Given an observation to be explained, local attributions aim to quantify the local importance of each input feature in the prediction produced by a trained complex model. However, local attributions in data streams have not received much attention in the past [30, 43, 159]. In particular, it is unclear how local attributions may change as a result of (local) model updates and concept drift, and whether an attribution is still meaningful after the time step in which it was generated.

There is little standardization in the evaluation of online learning methods. Due to the lack of publicly available non-stationary real-world data sets [154] and the multitude of different programming languages and libraries used in the literature [90, 8, 21, 118, 120, 73], the evaluation of online learning methods is not properly standardized. Indeed, authors often use different assumptions and experimental evaluation strategies, leading to confusing or non-reproducible results. Although various best practices have been proposed in the past [48, 62, 24, 63, 117], there is not yet a comprehensive set of evaluation standards and requirements including all core components of the online learning process. In particular, there is little standardization in the evaluation of online feature selection models and concept drift detection methods.

1.3 Our Contribution

In this thesis, we address four fundamental problems in online learning (see Section 1.2). We demonstrate the practical relevance of these issues through real-world examples, leading to new theoretical and empirical insights. On this basis, we present novel frameworks and implementations that overcome the limitations of previous work. Overall, our contributions cover major parts of the online learning process, solving known problems and thus contributing to more reliable machine learning in evolving data streams. In this sense, we also hope to provide an impulse for future work.

This thesis is based on six publications, each contributing to one of the four specific problems mentioned above. In the following, we briefly summarize our contributions:

Uncertainty-Aware Online Feature Selection and Concept Drift Detection (Paper 1 and 2, Chapter 2): We show that existing online feature selection and concept drift detection methods can often be outperformed by using information about the uncertainty of the predictive model. Specifically, we propose a flexible and probabilistic framework that allows us to quantify the uncertainty of model parameters, using an efficient incremental optimization on the marginal likelihood of the parameters (Paper 1). Our framework can be used with differentiable models such as Generalized Linear Models [124], neural networks, and Soft Decision Trees [91, 57]. On this basis, we present FIRES, a novel approach for stable online feature weighting and selection (Paper 1). By taking parameter uncertainty into account, FIRES is able to produce discriminative and much more stable feature sets than previous methods. In this way, FIRES improves the overall reliability of the obtained feature sets. Based on the same probabilistic framework, we propose ERICS, a novel concept drift detection approach. ERICS detects concept drift by monitoring changes in the estimated parameter distribution. Since concept drift typically manifests itself in more uncertain model parameters, ERICS is often able to detect concept drift more reliably than related methods. In fact, ERICS is one of few drift detection methods that can detect concept drift at the input feature level, thus also contributing to better interpretability.

Reliable and Interpretable Online Classification (Paper 4, Chapter 3): Heuristic updating procedures and increasing model complexities often impair the predictive reliability and the interpretability of state of the art online classifiers. As an alternative, we present the Dynamic Model Tree (DMT) (Paper 4). The DMT is a novel framework for more interpretable online learning based on Model Trees [33, 140, 138, 89]. Model Trees, unlike most decision trees, contain simple (linear) predictors at the leaf nodes. The DMT extends this idea by incrementally training simple models at both the leaf and inner nodes. Using gain functions based on the estimated loss of the simple models, updates to the DMT can be tied to changes in the approximate data concept. In this way, split and pruning decisions become much more understandable. Indeed, compared to existing incremental decision trees, the DMT offers considerable advantages in terms of predictive quality, complexity and intrinsic interpretability.

Efficient and Effective Local Explainability in Data Streams (Papers 3 and 5, Chapter 4): Explanation techniques such as local feature attributions have not received much attention in the context of online learning. As one of the first works, we investigate the behavior of local feature attribution methods under incremental model updates and concept drift (Paper 5). Specifically, we examine the local accuracy of attribution methods [109], a common property that expresses the adherence of the local explanation with the prediction of the model. As it turns out, attributions may become invalid with every model update, e.g., after a concept drift. Therefore, in order to obtain meaningful and reliable explanations over time, outdated attributions must be identified and recalculated. To this end, we propose an effective and model-agnostic change detection framework called CDLEEDS (Paper 5). With CDLEEDS, both global and local change can be detected, identifying outdated attributions and enabling more targeted and efficient recalculations.

In addition, we empirically investigate the role of the baseline for local feature attributions (Paper 3). The baseline represents missing discriminative information, e.g., a black pixel in image classification [158]. A baseline hyperparameter can be found in feature coalition-based attribution methods [109], but also in gradient-based attribution methods [151, 158] and evaluation tests with feature ablation [84, 144]. Although the corresponding paper has a broader focus, it allows us to draw interesting implications for meaningful baselines in data streams where our understanding of missingness might change over time.

Standardized Evaluation and Comparison of Online Learning Methods (Paper 6, Chapter 5): Without a common standard, online learning experiments are often not comparable or provide unreliable (i.e., non-reproducible or non-generalizable) and unrealistic results (e.g., if a model was optimized under extremely simplified or unrealistic assumptions for a given data set). We aim to fill this gap with a comprehensive summary on the evaluation of online learning methods (Paper 6). Specifically, we propose meaningful properties for the evaluation of online classifiers, concept drift detection methods and online feature selection models. In addition, we introduce a novel Python evaluation framework called float. Float automatizes major parts of the evaluation workflow, thereby enabling quicker and more comparable experiments. Since float can be combined with custom code, as well as popular online learning libraries like scikit-multiflow [118] and river [120], it is a flexible and powerful solution for better online learning experiments. Float is open-sourced and can be accessed on Github or the Python packaging index (Pypi).

1.4 Scope and Structure

This work deals with a variety of different online learning activities and techniques. Therefore, we cannot provide a thorough analysis of all related methods. Instead, we focus on the approaches that are most closely related to our contributions and highlight their limitations in relation to the problems mentioned above. The thesis is based on six publications. In the main body of this thesis we provide an overview of the contributions in each paper. Further details can then be found in the full papers in the Appendix of this thesis.

To make reading more comprehensible, we structure the background information and contributions according to the four topics described in Section 1.3. After a brief introduction to general concepts and the setting for online machine learning (Section 1.5), the thesis is organized as follows: In Chapter 2, we focus on online feature selection and concept drift detection. In this context, we introduce existing methods and present the probabilistic framework that serves as the basis for our proposed approaches: FIRES and ERICS. In Chapter 3, we look at online classification techniques, summarize the limitations of common approaches and introduce the novel Dynamic Model Tree framework for online learning. In Chapter 4, we examine the behavior of popular local attribution methods in data streams and highlight the important role of baselines for representing missing features. We then introduce the CDLEEDS change detection framework, which is an intuitive extension to local attribution methods in data streams. Finally, we discuss the need for uniform evaluation standards and introduce the new float Python framework in Chapter 5.

1.5 Typical Setting for Online Machine Learning

Online machine learning differs from regular batch learning as data arrives sequentially and is subject to temporal changes. Accordingly, machine learning in data streams requires specialized solutions, e.g., for preprocessing, handling concept drift, and predictive modeling. In this section, we briefly introduce the definitions, important terms, and assumptions that are relevant to our work.

In the context of online machine learning, a data stream is typically represented by a potentially infinite series of discrete time steps $1,\dots,t,\dots,T$. At each time step t, we receive a batch of observations $x_t \in \mathbb{R}^{n\times m}$, where n is the batch size and m is the number of input features. In the supervised setting, which is the focus of this work, we also obtain a vector of labels $y_t \in \mathbb{R}^n$ corresponding to the observations in x_t.

The data generating process can be described in terms of probability distributions. In particular, we can represent the input observations and labels by corresponding random variables X and Y that follow a probability distribution at time step t, commonly denoted as

$$P_t(X,Y) = P_t(Y|X)P_t(X), \tag{1.1}$$

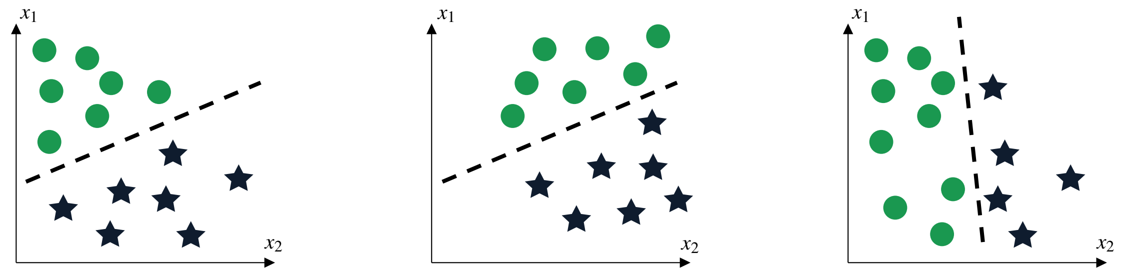
where $P_t(Y|X)$ is the conditional distribution of the target variable and $P_t(X)$ is the marginal distribution of the observations [108, 168, 64, 117, 45]. The probability distribution $P_t(X,Y)$ is also called the *active concept* at time step t [168]. Typically, a probability distribution is specified in terms of a probability mass function for discrete random variables and in terms of a probability density function for continuous random variables. However, the general notation of Eq. (1.1) is more commonly used in the literature.

1.5.1 Concept Drift

In offline batch learning, the data generating process is generally assumed to be stationary [117]. However, in data streams, the probability distribution $P_t(X,Y)$ may change over time. This phenomenon is called *concept drift*. Specifically, a concept drift between two time steps t_1 and t_2 is defined as a change in the joint probability distribution:

$$P_{t_1}(X,Y) \neq P_{t_2}(X,Y). \tag{1.2}$$

Based on this general definition, we can distinguish two fundamental types of concept drift: *real concept drift* and *virtual concept drift* [108, 168, 175, 64, 117, 45]. Real concept drift, sometimes also called concept shift, corresponds to a drift of the conditional distribution $P_t(Y|X)$, while the marginal distribution $P_t(X)$ remains stationary, i.e., for



(a) Original data concept at time step t_1: $P_{t_1}(X,Y)$.

(b) Virtual concept drift between the time steps t_1 (see subfigure (a)) and t_2: $P_{t_1}(X) \neq P_{t_2}(X)$.

(c) Real concept drift between the time steps t_1 (see subfigure (a)) and t_2: $P_{t_1}(Y|X) \neq P_{t_2}(Y|X)$.

Figure 1.1: **Virtual and Real Concept Drift.** We illustrate the two fundamental types of concept drift. This illustration is inspired by Gama et al. [64]. In the left subplot, we show the original data concept, i.e., the probability distribution $P_{t_1}(X,Y)$. Specifically, we depict two dimensional toy data with a binary target (indicated by green circles and blue stars). The decision boundary is highlighted by a dashed line. In the center subplot, we depict the active concept $P_{t_2}(X,Y)$ after a virtual drift. Virtual concept drift changes the marginal distribution of the data, i.e., $P_{t_1}(X) \neq P_{t_2}(X)$, but does not affect the conditional target distribution. Conversely, as the right hand subplot shows, real concept drift changes the conditional target distribution $P_{t_1}(Y|X) \neq P_{t_2}(Y|X)$.

two time steps t_1 and t_2 we write

$$P_{t_1}(Y|X) \neq P_{t_2}(Y|X). \tag{1.3}$$

On the other hand, virtual concept drift describes a shift of the marginal distribution $P_t(X)$, i.e.,

$$P_{t_1}(X) \neq P_{t_2}(X). \tag{1.4}$$

Virtual drift leaves the conditional distribution $P_t(Y|X)$ unaffected. An illustrative comparison of real and virtual concept drift is shown in Figure 1.1. In practice, we might also observe a mixture of both types of concept drift, leading to a change in both the conditional and marginal distribution [108].

We can further distinguish concept drift according to the characteristics of the transition between two concepts. While there exist extensive taxonomies of concept drift [168], the most commonly mentioned types are *sudden* (or abrupt), *incremental, gradual* and *reoccuring* concept drift [108, 168, 175, 64, 117]. A simple illustration of these concept drift types was introduced by Gama et al. [64] and is shown in Figure 1.2.

Sudden (or abrupt) concept drift corresponds to an immediate transition from the old to the new concept. Since the old concept becomes obsolete within a short period of time (often after a single time step), sudden drift is usually the easiest to detect. At the same

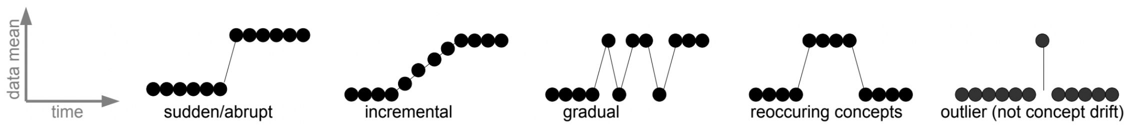


Figure 1.2: **Types of Concept Drift.** In addition to virtual and real concept drift, the literature often further distinguishes between different types of drift. The most common types are sudden, incremental, and gradual drift. We might also observe reoccuring concepts. Above, we show the illustration of the different concept drift types from Gama et al. [64], which is based on changes in the data mean. In practice, the true type of concept drift is usually unknown. Indeed, since concept drift manifests itself in atypical observations (compared to the ones seen so far), it is often difficult to distinguish concept drift from outliers.

time, sudden drift requires a short adaptation time to avoid a dramatic deterioration in predictive performance [117]. Incremental (or stepwise) concept drift describes a smooth transition between two concepts. Incremental drift usually occurs over a longer period of time, during which we observe several intermediate concepts [117, 64]. Incremental drift is thus often more difficult to detect than sudden drift. Similarly, during a gradual concept drift, the old and the new concept alternate within the transition period [117]. That is, both concepts are temporarily active until the new concept eventually replaces the old one [175]. Adjusting the predictive model during gradual drift can be difficult because we need to learn and maintain information about both concepts. Finally, we may also observe reocurring concepts. Such reocurrence can, but need not necessarily correspond to a periodic event [175, 117]. Ideally, to deal with reocurring concepts, one would need to memorize previous concepts and find an effective way to retrieve this information in the future. In practice, however, information on obsolete concepts is usually discarded for reasons of computational efficiency.

Webb et al. [168] proposed a more principled categorization of concept drift in terms of quantitative measures. However, since the data generating distribution is unknown in practice, the true type of concept drift usually remains unclear.

1.5.2 Common and Simplifying Assumptions

Real-world streaming processes are highly dynamic and therefore difficult to replicate in all details. To simplify the development and testing of online machine learning methods, one usually makes a number of assumptions. In the following, we briefly describe the most important assumptions for our work.

As mentioned above, we consider a supervised setting. In particular, if not stated otherwise, we assume a classification setting. That is, the random variable Y follows either a binary or categorical distribution, thereby representing the class label. In practice, labeling information might not be immediately available [62, 63, 8]. For example, in healthcare, the diagnosis corresponding to a vector of symptoms might not be available for a few days. However, for simplicity, we assume that there is no delay of labels.

In practice, the number or proportion of classes may change over time [24]. Similarly, the input feature space might be subject to temporal changes [141, 58]. For example, when processing streaming text data, new words may appear in the future while old words become irrelevant [114]. These phenomena are called concept evolution and feature evolution, respectively [85, 114]. However, in line with previous work, we assume that both the set of available classes and the set of input features are stable.

In contrast to machine learning for time series data, online machine learning is mainly concerned with heterogeneous feature sets and tabular data. In this context, we commonly assume the independence of observations drawn from the active concept [62, 8]. Although this independence assumption may be violated in practice, it has allowed for effective online learning in many applications.

Chapter 2

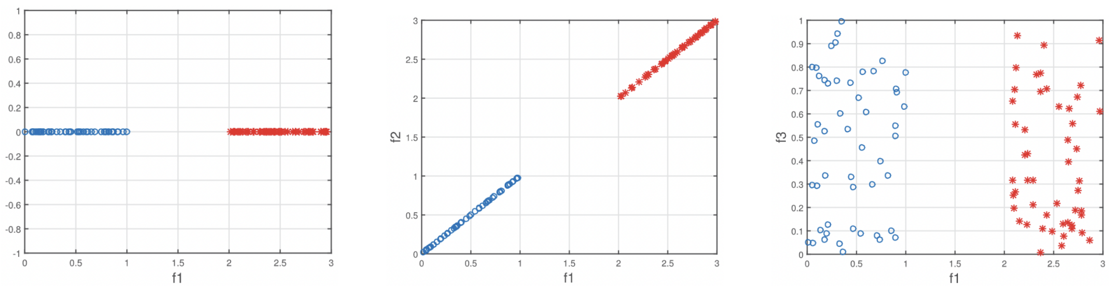
Uncertainty-Aware Online Feature Selection and Concept Drift Detection

Online feature selection and concept drift detection are two important tasks in online machine learning. While feature selection is a preprocessing technique used in many scenarios, concept drift detection is unique to online learning. Both tasks have been studied extensively over the years. However, there are still open questions, especially regarding the stability and reliability of existing methods in both contexts. In this chapter, we present the state of the art in online feature selection and concept drift detection. We also take a more general look at common approaches to handle concept drift during incremental training. We then introduce a novel probabilistic framework that enables improved online feature selection and concept drift detection by taking into account the uncertainty of the optimal model parameters. The contributions presented in this chapter are based on the Papers 1 and 2.

2.1 Background on Online Feature Selection

Many real-world applications deal with high-dimensional data. High-dimensionality typically entails considerable memory consumption and computational cost for training a machine learning model [26, 103]. Indeed, high-dimensional training data can lead to a series of phenomena and problems that do not occur in a low-dimensional setting and are called the *curse of dimensionality* [14]. Specifically, training data usually become sparse in high-dimensional space, which in turn can lead to poor generalization ability of the predictive model [26]. That is, predictive models trained on a high-dimensional feature space tend to overfit if the number of training observations does not grow proportionally [103]. To mitigate the effects of the curse of dimensionality, and thereby improve the generalization and computational efficiency of machine learning methods, we can apply dimensionality reduction techniques.

Feature selection and *feature extraction* are the most popular approaches for dimensionality reduction. Feature extraction methods generate low-dimensional representations by transforming the original high-dimensional feature space [141, 103]. Principal component analysis and matrix factorization techniques like singular value decomposi-



(a) Relevant feature f_1.

(b) Redundant feature f_2 given feature f_1.

(c) Irrelevant feature f_3.

Figure 2.1: **Important Feature Types During Feature Selection.** Feature selection methods aim to eliminate irrelevant and redundant features, while retaining all features that are relevant for the prediction of the target [38]. Above, we depict a simple example of the three general types of features that need to be considered for feature selection. The blue and red markers indicate a binary target class. This illustration was taken from Li et al. [103].

tion are widely used feature extraction methods [72]. Feature selection, on the other hand, selects a subset of the original features [141, 103]. In particular, feature selection aims to find a small subset of the original input features that contains enough information about the target class to achieve high predictive performance [26]. Since feature selection retains the original meaning of the input features, it generally provides better interpretability [103, 141]. Therefore, we focus on feature selection in this thesis.

In general, feature selection models aim to identify relevant, i.e., discriminative, input features and discard irrelevant and redundant ones. A feature is irrelevant for the prediction, if it is conditionally independent of the target class [38]. Similarly, a feature can be considered redundant if it correlates strongly with another input feature, thereby providing similar discriminative information about the target. Typically, feature selection algorithms aim to avoid redundancies in the selected feature set [103]. However, Guyon and Elisseeff [72] conducted a comprehensive analysis of the effects of feature correlation, which shows potential benefits of including redundant features. A simple illustration of the different types of features considered in feature selection was provided by Li et al. [103] and is shown in Figure 2.1.

Evaluating all possible feature sets is not tractable. As a result, feature selection models cannot generally guarantee that the optimal (i.e., most predictive) feature set will be selected [38]. Likewise, the optimal number of selected features is usually not known and must therefore be specified by the user via a hyperparameter [103]. In practice, however, it is often difficult to optimize the number of selected features. If the set of selected features is too small, relevant features could be wrongly discarded. If, on the other hand, the set of selected features is too large, irrelevant features may still be used, which can impair the predictive performance [103]. Most feature selection approaches comprise an auxiliary feature weighting or ranking procedure [72], which can help optimize the size of the selected feature set. For example, the number of selected features can be set dynamically by specifying a threshold for the feature weights obtained.

Feature selection has proven effective in mitigating the effects of high-dimensional data in the stationary setting [72]. However, *online feature selection* has not received the same level of attention in the past [26]. In data streams, feature selection is not a preprocessing technique that is applied once to the entire training data, but must be applied to every incoming streaming observation. In addition, the importance of the input features may change over time due to concept drift, so we need to update the selected features accordingly. In fact, changes in feature importance (also called feature drift) directly affect the decision boundary and can thus be considered a form of real concept drift [141]. If we fail to eliminate features that become irrelevant over time and include new relevant features in their place, predictive performance generally suffers.

In the literature on online feature selection, different assumptions are made about the streaming process [141]. In particular, we can distinguish between online feature selection methods for *data streams* and *feature streams*. As described in our definition from Section 1.5, a data stream consists of a series of time steps, where we receive a new observation (or batch) at each time step. Moreover, we generally assume that the feature space is stable, i.e., that the set of available input features is fixed. In contrast, in a feature stream setting, we assume that new features become available over time, while the sample of observations remains fixed [141, 103]. Based on this static sample, feature stream methods determine whether a newly observed feature should be included in the selected feature set and whether an old feature should be removed in its place [103]. Many online feature selection methods are designed for the feature stream setting [174, 169, 170, 102, 167, 134]. According to our definition (Section 1.5), however, a streaming process is characterized by the gradual appearance of new observations. Hence, we consider the more common data stream setting. Figure 2.2 illustrates the difference between both settings. Note that real-world applications may also comprise a combination of data stream and feature stream [141].

In both the offline and online setting, we distinguish between three categories of feature selection methods: *filters, wrappers,* and *embedded methods* [141, 26, 72]. Intuitively, these categories are based on the relationship and interaction between the feature selection model and the predictive model [26]. In particular, filter methods act independently from the prediction [38]. They are typically based on simple statistical measures (e.g., correlation). Wrapper methods use a predictive model to identify relevant features. In general, wrappers aim to select the feature set that offers the highest improvement in predictive performance. Embedded feature selection methods are most strongly entangled with the predictive model. Here, feature selection is an inherent part of the training procedure. In Section 2.1.1, we give a brief overview of popular online feature selection methods in each category.

Aside from the discriminative power of the selected features, the stability of feature sets has received attention in the past [72, 38, 93, 128]. In data streams, where the

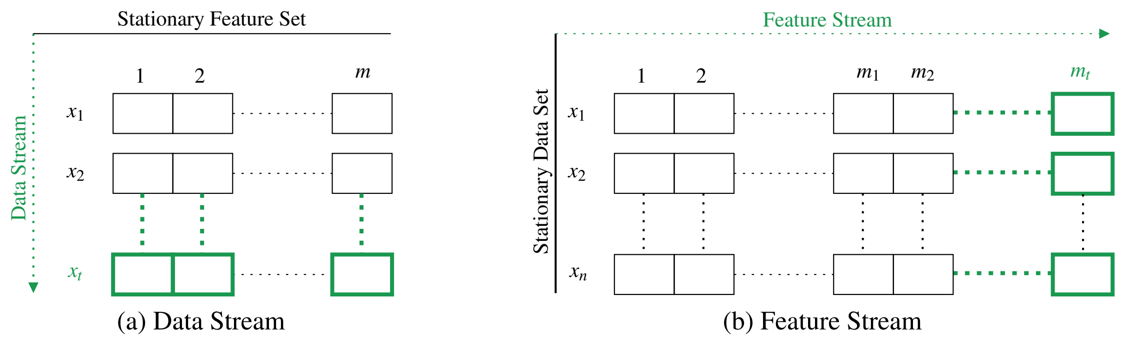


Figure 2.2: **Online Feature Selection Settings.** Online feature selection methods usually adopt one of two stream processing settings [103, 141]. In the traditional data stream setting (left subplot), new observations appear over time, while the set of features is assumed to be stable. Conversely, in the feature stream setting (right subplot), novel features may appear over time, while the data sample is assumed to be fixed. Above, the green color indicates the dimension that is subject to a temporal change. In this thesis, we focus on the more common data stream setting.

online predictive model is subject to change and uncertainty, the stability of the generated feature set can have a strong impact on the interpretability and overall reliability of the underlying model. In Section 2.1.2, we go into more detail about the importance of stable feature sets.

Finally, more information about feature selection in general, and online feature selection in particular can be found in several surveys [38, 72, 103, 141, 103].

2.1.1 Popular Methods for Online Feature Selection

As mentioned above, feature selection models are usually divided into three categories based on the degree of entanglement between feature selection and the training process [26]. Accordingly, we introduce popular online filter, wrapper and embedded feature selection models below.

Online Filter Methods

Filter methods for online feature selection are independent of the training process [141]. Instead, filters typically apply simple and efficient weighting strategies based on an information-theoretic or statistical measure.

Katakis et al. [97] proposed an online feature selection filter for text processing. They argued that each word (feature) can be ranked according to a cumulative statistic of the frequency of occurrence of the word in each class. Specifically, they applied the χ^2 statistic to rank words. This approach can be transferred to a more general setting with arbitrary feature types. Accordingly, we can use the χ^2 statistic to measure the co-occurrence of a feature value and class, and thus quantify the relative importance of each input feature.

The Fast Correlation-Based Filter (FCBF) is based on symmetrical uncertainty, a coefficient that quantifies the dependence of features using the information gain normalized by the entropy [171]. FCBF is applied in a sequential procedure to remove irrelevant and redundant features. The original FCBF algorithm can also be used in a sliding window approach (see Section 2.2.1), which was shown by Nguyen et al. [125]. In particular, the authors proposed a Heterogeneous Ensemble with Feature Drift for Data Streams (HEFT) that integrates ensemble learning and online feature selection based on FCBF.

The DXMiner is a holistic classification framework that includes a feature selection component [114]. In particular, the authors proposed a predictive and an informative feature selection approach. The informative feature selection is an unsupervised approach where features are selected according to their frequency in the current training batch. The predictive feature selection, on the other hand, is similar to the filter method of Katakis et al. [97], but uses a new selection criterion called deviation weight. Similar to the χ^2 statistic, the deviation weight scales with the conditional frequency of a feature given the class.

For most online filters, including the three methods presented above, we need to maintain frequency statistics over time, e.g., in a sliding window (see Section 2.2.1). The performance of a filter depends heavily on whether these frequency statistics are representative of the active data concept, which can be difficult to achieve. Filters also do not take into account the model's performance and uncertainty, which can reduce the robustness and expressiveness of the selected features.

Online Wrapper Methods

Many online learning models maintain an internal representation of the importance of each input feature. For example, linear models represent the input features by a vector of weights. Wrapper methods use this model-inherent information to select features [141].

The online feature selection (OFS) algorithm is one of the best known wrapper methods [166]. OFS uses the weights of a Perceptron model trained by stochastic gradient descent. Since the raw Perceptron weights are too unreliable for online feature selection, Wang et al. [166] proposed a sparse projection approach. Specifically, OFS projects the weight vector to an L2 ball to ensure the boundedness of the obtained feature weights. OFS then selects the features with highest absolute weight.

Similarly, the Extremal Feature Selection (EFS) model uses the weights of a modified balanced Winnow classifier [37]. EFS ranks the input features according to the absolute difference between the positive and negative weights of the Winnow model. Surprisingly, Carvalho and Cohen [37] found that the discriminative power of the obtained feature sets can be improved by including also a small number of low-ranked features.

Linear models such as the Perceptron or Winnow algorithm are particularly adept for online feature selection due to their efficiency. In general, however, a wrapper could also be implemented with a more complex model type, such as a neural network. For example, the Cancelout approach introduces an additional feature selection layer after the input layer of a neural network [28]. The nodes of the Cancelout layer are mapped one-to-one to the nodes of the input layer. The network then automatically learns the weights of input features, such that the activation of irrelevant features in the Cancelout layer approaches zero. Although Cancelout is an offline approach, the algorithm could generally be used in an online setting where the neural network is trained incrementally (e.g., via stochastic gradient descent).

While many wrapper methods exploit the first order information of a predictive model, Liu et al. [104] proposed an online feature selection method based on second order information. Their Adaptive Sparse Confidence Weighted (ASCW) algorithm uses an ensemble of confidence-weighted linear learners [49] that consider feature correlations by modeling a Gaussian distribution over the weights. According to Liu et al. [104], the ASCW algorithm is particularly well-suited for online feature selection on imbalanced targets, as it employs a cost-sensitive loss function.

Finally, there are also unsupervised wrapper methods, such as the Feature Selection on Data Streams (FSDS) [87]. FSDS applies matrix sketching techniques to obtain a low-rank approximation of incoming data. The low-rank matrix is continuously updated as new observations arrive. Using a regularized linear model (e.g., Lasso or Ridge), FSDS is able to generate feature weights from the current low-rank approximation at each time step. Due to the use of matrix sketching techniques, the FSDS algorithm can be more time-consuming than related wrapper methods.

Wrappers are usually associated with higher computational costs than filters, as they require an incrementally updated predictive model. However, if the model is robust to noise and adapts well to concept drift, the selected features of a wrapper can be much more discriminative [141]. In general, the performance of a wrapper depends strongly on the underlying predictive model. For example, if the weights of a linear model vary greatly between time steps, the sets of selected features will be unstable. Yet, none of the above methods except ASCW [104] takes into account the uncertainty of the model, which could help to extract more stable and meaningful feature sets. While ASCW only works for a specific linear model type, we generally aim for an online feature selection framework that offers the same level of expressiveness but can be flexibly applied to different (non-linear) model types depending on the application at hand.

Online Embedded Methods

Feature selection does not always have to be an isolated activity but can be embedded in the training process [141]. Lasso and Ridge regression models are popular examples of offline predictive models with embedded feature selection. In online learning, incremental decision trees are the most popular models with embedded feature selection. In an incremental decision tree, the set of selected features implicitly grows whenever a new feature is used for splitting. We discuss incremental decision trees in more detail in Section 3.1.4.

The distinction between wrapper and embedded methods can sometimes be confusing. In general, a wrapper exploits a predictive model for feature selection without using it for the actual prediction. A separate predictive model is trained on the reduced feature set in a second step. Embedded methods, on the other hand, use the same model for feature selection and prediction.

One advantage of embedded feature selection is that the number of features is usually determined automatically. Accordingly, there are fewer hyperparameters to optimize. On the other hand, this gives us less control over the type of features selected.

2.1.2 Importance of Stable Feature Sets

As mentioned earlier, it is typically intractable to find the optimal set of features from all possible combinations. Therefore, feature selection methods generally optimize the selected features with respect to a heuristic, e.g., the empirical error or accuracy [38]. As a result, there may be many possible combinations of features that all achieve the same or similar performance – especially if the data set contains redundant features [72]. The abundance of possible feature sets may lead to undesirable variations. However, high variability in the selected features for samples drawn from the same data distribution can lead to practical concerns about robustness, trustworthiness and overall reliability [72, 38]. Feature selection methods should therefore aim to produce stable feature sets. Stable feature sets reduce the computational cost of (re)training the model (especially if feature selection is not embedded in the training process) and can help increase user confidence in the prediction [93].

Algorithmic stability has been extensively studied in the context of predictive machine learning [161, 31, 70, 74]. Here, stability describes the sensitivity of the model outcome to variations in the input data. However, the stability of feature selection methods has not received the same attention [38]. Kalousis et al. [93] describe the stability of a feature selection algorithm as the sensitivity of the generated feature preferences to differences in the training set drawn from the same data generating distribution. The term "feature preferences" serves as a generalization of the possible outcomes of a feature selection model, i.e., feature weights, a ranking, or a selected subset. In this thesis, we are mainly concerned with subset selection, where feature weighting or ranking is often used as an intermediate step. Accordingly, we use the term *feature set stability* (also known as subset stability [128]) throughout our exposition.

The stability of feature sets is particularly important in data streams, which often have a high degree of variation and noise. Since the selected features are updated at each time step, high variability of the selected subset between time steps can lead to considerable model retraining costs and reliability issues. Fortunately, the notion of feature set stability can be directly applied to the online case. Specifically, we say that an online feature selection model is stable if the feature subset obtained has low variability over time under a stationary data concept.

To quantify feature set stability, we require a similarity measure [93]. For example, Kalousis et al. [93] proposed different correlation measures to quantify the similarity of two sets of feature preferences. Later, Nogueira et al. [128] introduced a novel stability measure for feature sets that is a generalization of many existing measures and has a number of desirable properties. Although the stability measure of Nogueira et al. [128] is specified for offline methods, it can be readily adopted for data streams. In Paper 1 (and again in Paper 6), we describe a corresponding implementation, which we repeat below for the sake of convenience.

Based on the exposition of Nogueira et al. [128], let $a_t \in \{0,1\}^m$ be the active feature vector at time step t, where m is the total number of features. Specifically, selected features are represented by ones and unselected features are represented by zeros. We can measure feature set stability at time step t for a sliding window of size w as:

$$FSS_{t,w} = 1 - \frac{\frac{1}{m}\sum_{j=1}^{m} s_j^2}{\frac{k}{m}\left(1 - \frac{k}{m}\right)}, \tag{2.1}$$

where k is the number of selected features specified by the user and $s_j^2 = \frac{w}{w-1}\hat{p}_j(1-\hat{p}_j)$ is the unbiased sample variance of the selection of feature j, with $\hat{p}_j = \frac{1}{w}\sum_{i=0}^{w-1} a_{t-i,j}$. Here, $a_{t-i,j}$ denotes the j'th element of the active feature vector at time step $t-i$, i.e., it denotes whether feature j was selected at time step $t-i$. Note that the feature set stability measure decreases, if the total variability of the selected features $\sum_{j=1}^{m} s_j^2$ increases. The stability Eq. (2.1) is maximized if $s_j^2 = 0$ for all features j, i.e., if the selected feature set remains stationary over the full length of the sliding window. Unfortunately, this measure does not explicitly account for concept drift, where we would tolerate some degree of variation in the feature set. However, as long as we lack a dedicated measure, Eq. (2.1) provides a valuable indication of the stability of feature sets over time.

In general, online feature selection models should aim at providing both discriminative and stable feature sets [93]. However, the trade-off between stability and sufficient flexibility to adapt to concept drift can be challenging and should receive more attention [141].

2.2 Background on Concept Drift Handling

Concept drift (see Section 1.5) is a fundamental characteristic of data streams that distinguishes online machine learning from batch learning. In particular, concept drift can render previously learned data distributions and model parameters obsolete. As a result, concept drift can have a large impact on the predictive quality and interpretability of online learning approaches.

Concept drift requires us to "forget" outdated information while we continue to learn new information about the current data generating concept [64]. In general, we can deal with concept drift in a passive way, by continuously updating the model, or in an active way, by explicitly detecting concept drift and making appropriate model adjustments [45]. In the following, we outline common mechanisms for passive model adaptation and present popular methods for active concept drift detection.

2.2.1 Mechanisms for Passive Model Adaptation

To some extent, any online learning model that is incrementally updated will naturally adapt to new data generating concepts over time [64]. However, predictive performance in the presence of concept drift – and in particular sudden concept drift – can usually be improved by including an explicit forgetting mechanism in the model design. Most passive forgetting mechanisms either apply performance-based re-evaluation of old model components or use a windowing scheme to keep up with recent training observations. Other passive adaptation strategies include instance selection and feature engineering or manipulation [175]. However, these strategies are out of the scope of this work.

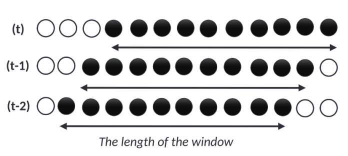
Dynamic Re-Evaluation of Model Components

Dynamic re-evaluation of model components is an effective mechanism to adapt to concept drift. In general, the goal is to identify parts of the online learning model that no longer contribute to the overall performance and dynamically delete or replace them. Re-evaluation mechanisms are usually integrated into the incremental update procedure and use either the historical predictive performance [176] or a heuristic gain measure (e.g., as in the incremental decision trees [111] introduced in Section 3.1.4).

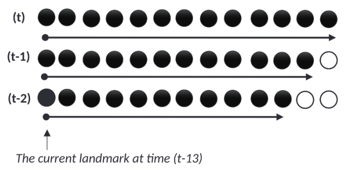
Ensemble models (see Section 3.1.5) are particularly well suited for such re-evaluations [45]. Indeed, ensembles allow the isolated addition and deletion of a weak (base) learner without affecting the remaining components. If the weak learners are diverse, this can be a practical advantage as concept drift often only affects parts of the input space. The adaptability of an ensemble is usually achieved through dynamic weighting (fusion) schemes [175]. In this context, the weights represent the confidence in the weak learners and control their influence on the prediction of future observations. In Section 3.1, we introduce popular adaptive online learning models and ensembles.

Windowing Schemes

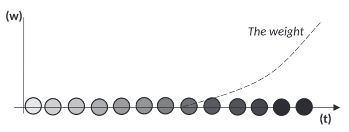
Time windows are a simple and popular approach to obtain a representation of the active concept at any point in time. Many online learning techniques integrate time windows to use current observations more effectively [88, 18]. Time windows are also a key component of a large number of active concept drift detection approaches (see Section 2.2.2). Although windowing schemes have proven themselves in practice, they introduce additional hyperparameters and increase memory consumption. A simple illustration of



(a) Sliding window, i.e., a window of recent observations with fixed or adaptive [17] size.



(b) Landmark window, i.e., a window of all observations starting from a landmark point.



(c) Decayed window, i.e., a gradual forgetting of old observations by means of decaying weights.

Figure 2.3: **Popular Windowing Schemes.** Windowing schemes are a popular mechanism for passive model adaptation in the presence of concept drift. Indeed, they are a straight-forward approach to "forgetting" outdated information in online learning. Above, we depict illustrations by Bahri et al. [8] for three of the most common windowing schemes.

common windowing schemes was introduced by Bahri et al. [8] and is shown in Figure 2.3. We summarize the different windowing schemes below.

Fixed-size *sliding windows* are one of the most frequently used schemes. A sliding window stores recent streaming observations according to the first-in-first-out (FIFO) principle [64, 8]. In this way, old observations are gradually replaced by new ones. Choosing an appropriate size for the sliding window can be difficult in practice. If the window size is too large, the window might contain outdated information long after a concept drift, which in turn can lead to performance deterioration [126]. Conversely, if the window size is too small, one might discard information that is still valid. Indeed, a small window size may lead to models with high variance and overfitting [125].

To overcome the difficulty of selecting an appropriate window size, Bifet and Gavaldà [17] proposed the Adaptive Windowing (ADWIN) method. ADWIN manages a variable-length window of recent observations. In a first step, new streaming observations are added to the window as they become available. The window size is then automatically adjusted if the window contains two sub-windows that exhibit distinct enough averages. Specifically, if the average value of the older sub-window differs significantly from the average of the newer sub-window, the old sub-window can be removed, reducing the overall size of the sliding window. Accordingly, ADWIN grows the window as long as the active concept is stationary and shrinks the window by removing outdated observations as soon as a change becomes apparent. In other words, ADWIN automatically adjusts the sliding window size to the current rate of change in the data. A shrinking ADWIN window can therefore also be considered as an indicator of concept drift [17, 58].

An alternative windowing scheme is the *landmark window* [8]. A landmark window stores all streaming observations after a specified time step (the landmark). Indeed, the landmark window continues to grow until the landmark is reset. Resetting the landmark deletes all previously stored observations. Consequently, resetting the landmark means that we are temporarily relying on a relatively small set of recent observations, which could affect the robustness of our model. For this reason, the landmark is sometimes positioned at the first time step and never reset, creating a window that contains the entire data stream [126]. Obviously, this approach is infeasible for realistic, large-scale streaming applications.

Both the sliding window and the landmark window correspond to a mechanism of abrupt forgetting [64]. In particular, each observation within the window has the same importance. That is, the weight of old observations in the window does not diminish over time. However, as soon as an observation falls out of the window, it abruptly loses its importance. This abrupt behavior might affect the performance of the online learning model – in particular if the window is small.

Decayed windows, also called damped or fading windows, provide a more gradual decay of importance [8, 126, 64]. Decayed windows assign weights to each streaming observation that scale proportionally to their age. Specifically, the weight of old and possibly outdated observations is automatically reduced over time by a fading function. If the weight of an observation falls below a threshold, the observation may be removed altogether [8]. The choice of the fading factor, i.e., the degree by which the weight of old observations deteriorates, can be difficult in practice. Note that the decayed windowing scheme is also closely related to the well-known Exponentially Weighted Moving Average (EWMA).

The *tilted time window* is a another, but less common windowing scheme [126, 58]. A tilted time window stores observations at different granularities depending on their age. Methodologically, this approach lies somewhere between a sliding window and a decayed window [126]. Similar to these schemes, the tilted time window follows the intuition that old observations can be stored with coarse granularity, while more recent observations should be stored with finer granularity. In this way, the tilted time window aims to provide a memory efficient representation of the entire data stream.

Online sampling approaches are an alternative to the classical windowing schemes presented above [64]. *Reservoir sampling* [164] is the most common approach to obtain a representative sample from a data stream. In contrast to windowing schemes, reservoir sampling produces an unbiased sample. Each streaming observation is randomly included in the reservoir with uniform probability. In turn, a randomly chosen observation is removed from the reservoir to maintain the specified sample size. Sampling techniques, while providing an effective way to summarize statistics from data streams, may not be suitable for certain applications such as anomaly or extreme value detection [58, 64]. Since sampling methods are not relevant to our work, we do not go into further detail.

2.2.2 Popular Methods for Active Concept Drift Detection

Passive adaptation strategies (see Section 2.2.1) allow us to adjust the online learning model continuously or periodically. Alternatively, many online learning frameworks use dedicated methods to actively detect concept drift. Generally, these strategies involve re-training the online learning model (or parts of it) when a concept drift is explicitly detected [64, 175]. Active concept drift detection often works well in the presence of sudden concept drift, which requires an immediate – and typically extensive – adjustment of the online learning model [45].

Reliable concept drift detection is usually difficult to achieve. In particular, we need to be able to distinguish concept drift from noise, which manifests itself in similar ways in the data and in the model. Therefore, most drift detection models are very sensitive to the configuration of their hyperparameters. However, active drift detection can provide additional insights into the latent dynamics of the data stream and the behavior of the model. It can therefore also be a tool to improve the overall interpretability of online learning.

Lu et al. [108] introduced a comprehensive taxonomy of concept drift detection methods. Accordingly, concept drift detection can be distinguished according to whether it is based on the error rate, the data distribution or a combination of different criteria using multiple hypothesis testing. Below, we briefly outline each category and introduce popular implementations.

Error Rate-Based Drift Detection

Error rate-based drift detection is the largest category of active concept drift detection methods [108]. These methods follow a simple intuition: In general, the predictive error of an online learning model can be expected to decrease as long as the model is optimized with respect to a stationary data concept [61, 162]. However, as soon as concept drift changes the data generating distribution, the error usually increases because the model has not yet been optimized with respect to the new data concept. A (significant) increase in the error is therefore an indication of concept drift. This principle can also be applied to other performance estimates such as accuracy [69].

The adaptive windowing method ADWIN [17], described in Section 2.2.1, can be used as an error rate-based drift detector. ADWIN automatically reduces its size if it identifies subwindows with significantly different means. Accordingly, the length of an ADWIN window over the observed errors would decrease if the mean error changes significantly. This change in window size can in turn be used to detect a concept drift.

The Cumulative Sum (CUSUM) and the Page-Hinkley test are two memory-less error rate-based drift detection methods [23]. Both tests are originally based on the expositions of Page [131] and have been adopted for online learning [23, 118, 21, 69]. Intuitively, these tests attempt to detect concept drift through deviations from the mean error over time.

Let e_t be the error at time step t and let $\bar{e}_t = \frac{1}{t}\sum_{i=0}^{t} e_i$ be the mean error up to the given time step. Moreover, let $z_t = e_t - \bar{e}_t$ be the difference between the current error and the mean error. To identify concept drift, CUSUM calculates the following test statistic at time step t:

$$g_t = \max(0, g_{t-1} + z_t - \delta), \tag{2.2}$$

where δ is a hyperparameter that controls the sensitivity of the test [23]. The test statistic is initialized with zero at time step $t = 0$. If $g_t > h$ for a user-defined threshold h, CUSUM issues a drift alert and resets the test statistic to $g_t = 0$. In other words, if concept drift increases the error e_t, it temporarily increases z_t (as the mean error $\bar{e}_t$ does not change as quickly), which ultimately increases the test statistic g_t of the CUSUM method. The Page-Hinkley test operates in a similar fashion, using the following test statistic [23]:

$$g_t = g_{t-1} + z_t - \delta \tag{2.3}$$

Given the current minimal statistic, i.e., $g_t^{\min} = \min(g_{t-1}^{\min}, g_t)$, the Page-Hinkley method issues an alert if $g_t - g_t^{\min} > h$. Note that there exist alternative implementations of the Page-Hinkley test that use a decay hyperparameter $\alpha \in [0,1]$ to calculate the test statistic $g_t = \alpha g_{t-1} + z_t - \delta$ [118]. In this case, the corresponding test is $g_t > h$.

The Drift Detection Method (DDM) proposed by Gama et al. [61] is another well-known model. In DDM, the error rate is described by the probability of misclassifying an observation at time step t, denoted p_t. The corresponding standard deviation is given by $s_t = \sqrt{p_t(1-p_t)/t}$. Similar to the Page-Hinkley test, DDM stores the minimal statistics p_{min} and s_{min} over time, which are updated whenever $p_t + s_t$ reaches a new minimum. DDM issues an intermediate warning at $p_t + s_t \geq p_{min} + 2s_{min}$ (corresponding to a confidence level of 95%) and a drift alert at $p_t + s_t \geq p_{min} + 3s_{min}$ (corresponding to a confidence level of 99%) [61]. The warning can be used to prepare a retraining of the predictive model. Specifically, once the warning level is reached, we may start saving all future observations, which corresponds to a landmark windowing scheme (see Section 2.2.1) [108]. If a drift is detected, we can use these recent observations to retrain the model [61]. If the error decreases after the warning without reaching the drift threshold first, we can assume a false alarm. In this case, we would discard the stored observations and continue training the old model.

There are many other and more advanced methods for detecting concept drifts based on error rates. In general, these methods are very similar to the ones described above, but differ in the way the error is accumulated over time, what test statistics are calculated, or what threshold is applied for issuing a warning and drift alarm. Below is a concise summary of some of the most popular methods.

The DDM algorithm [61] has inspired a variety of follow-up work. The Early Drift Detection Method (EDDM) is an early adaptation of DDM [6]. EDDM is based on the distance between errors, i.e., the number of streaming observations between two errors. Another adaptation is the Reactive Drift Detection Method (RDDM) [12]. RDDM resets the statistics of the DDM method, if the data concept is stationary for a long time, i.e., if we observed many observations without issuing an alert. In this way, RDDM aims to obtain more recent error statistics and thus react to concept drift faster. DDM was also used for local drift detection, by training a separate drift detector at every inner node of an incremental decision tree [59]. Another method similar to DDM was proposed by Ross et al. [145]. The authors used exponentially weighted moving averaging (EWMA) to obtain a more recent estimate of the error rate. The corresponding algorithm is called ECDD (EWMA for Concept Drift Detection).

Drift detection methods also often use probabilistic inequalities, which allow more principled statistical tests and performance guarantees. For example, Hoeffding's inequality, which we discuss in more detail in Section 3.1.4, is used by the Hoeffding Drift Detection Method (HDDM) [56] and the Fast Hoeffding Drift Detection Method (FHDDM) [135]. The Adaptive Sliding Window Based Drift Detection Method (ADDM) also applies Hoeffding's inequality to dynamically determine the size of a sliding window [50]. ADDM then monitors the entropy in the window to detect concept drift. In contrast to the above methods, the McDiarmid Drift Detection Method (MDDM) uses the more general McDiarmid's inequality [137].

Instead of using a single time window, one may also compare error rates across two time windows. The Statistical Test of Equal Proportions Detection (STEPD) compares the accuracy in a window of recent observations with the accuracy in the overall landmark window from the beginning of the data stream [127]. Cabral and Barros [35] proposed a variation of the STEPD method using Fisher's exact test. Similarly, the Paired Learners (PL) method uses two models, a "stable" learning model that is trained on all observations and a "reactive" learning model that is trained on a sliding window of recent observations [5]. PL uses the difference between both models to detect concept drift.

A different approach was proposed by Harel et al. [75], who apply permutation tests to estimate the distribution of the loss, which in turn is used for concept drift detection. The method of Sobhani and Beigy [153] processes streaming observations batch-wise and detects concept drift by comparing the labels of close data points in successive batches. And finally, How Tan et al. [86] presented a semi-supervised approach to concept drift detection that measures the diffusion of density estimates of the posterior class probabilities.

Error rate-based drift detection methods are usually very efficient. They are also model-agnostic, i.e., they can be applied to any type of model. However, for reliable error rate-based drift detection, label (target) information must be continuously available and the online learning model must be robust. In particular, if the data concept is stationary, the predictive model must provide stable error scores to avoid false alarms.

The above summary of error rate-based concept drift detection methods is not exhaustive. For more information on related methods, we refer to the surveys of Gonçalves et al. [69], Lu et al. [108] and Gama et al. [64].

Data Distribution-Based Drift Detection

One may detect concept drift directly from changes in the (estimated) distribution of streaming observations. For this purpose, data distribution-based methods typically use distance measures to quantify the dissimilarity between historical and current observations [108].

An early distribution-based drift detection approach was introduced by Kifer et al.

[98]. The proposed method compares the observations in a sliding window of recent observations with a reference (landmark) window of historical observations. Specifically, concept drift is detected by testing whether the samples in the two windows were drawn from different distributions. The reference window is reset when a concept drift is detected. Kifer et al. [98] introduced a new family of distance functions that provide sensible guarantees. To detect different types of concept drift, they used several pairs of windows with different sizes.

A similar window-based strategy was used by Dasu et al. [42]. Here, the authors proposed an information-theoretic approach that uses the Kullback-Leibler divergence to measure the distance between empirical distributions in two sliding windows. In particular, they apply bootstrapping to obtain a set of distance estimates and thereby determine statistical significance. Similar to Kifer et al. [98], Dasu et al. [42] argued that we can account for different types of concept drift by using multiple windows of different sizes.

Data distribution-based methods can often provide information about the location of concept drift in the input space. However, they generally involve higher computational costs than the error rate-based models presented above. Additional methods and information on data distribution-based concept drift detection can be found in the survey of Lu et al. [108].

Drift Detection With Multiple Hypothesis Tests

Concept drift detection methods typically use a single test statistic, e.g., based on the error rate. In practice, however, this can lead to problems with reliability. For example, if the target class is imbalanced, the error rate is not a meaningful indicator of changes in model performance [165]. Concept drift detection methods based on multiple hypotheses aim to compensate for such weaknesses. In these approaches, concept drift is detected by a structured examination of multiple characteristics, trading computational efficiency for more reliable results. In general, one can distinguish between approaches with parallel and hierarchical multiple hypothesis testing [108].

Alippi and Roveri [3] proposed an extension of the CUSUM model [131, 23] for parallel multiple hypothesis testing. Assuming that the streaming observations are identically and independently distributed, the authors apply the central limit theorem to model the data stream by a set of statistics (i.e., the mean and variance of a normal distribution). These statistics are then extended to a larger feature set, including new features generated by the pairwise correlation of existing ones. This high-dimensional feature vector is again reduced by a Principal Component Analysis. Finally, Alippi and Roveri [3] use the resulting features in an extended multidimensional CUSUM test.

The Linear Four Rates (LFR) algorithm is another concept drift detection model based on parallel multiple hypothesis testing [165]. LFR detects concept drift based on four independent tests on the characteristic rates (TP, TN, FP, FN) of a confusion matrix. Compared to simpler error rate-based drift detection, the performance of LFR is not affected by class imbalances.

The idea of hierarchical multiple hypothesis testing is to validate the original drift detection in an additional layer [108]. Specifically, the hierarchical approach aims to make concept drift detection more reliable by filtering out false alarms of the first layer model. Yu et al. [172] proposed a hierarchical hypothesis testing framework, which they combined with the LFR model [165] to detect concept drift. At the first layer, they use LFR to detect a potential concept drift. The second layer confirms or rejects the potential concept drift based on a more elaborate permutation test. If the second layer test declares the drift detection a false alarm, the LFR model statistics are reset. If, on the other hand, the second layer confirms the concept drift, the predictive model can be updated or retrained accordingly.

As before, we refer to the overview of Lu et al. [108] for more information on concept drift detection based on multiple hypothesis testing.

2.3 An Incremental Framework for Modeling Parameter Uncertainty

Online learning models often have to deal with noisy or erroneous inputs, as new observations are generated in real time (e.g., social media posts or sensor signals) and thorough data cleansing is not possible. In addition, the model can only access a small number of observations at each time step and must account for concept drifts. As a result, updates to the model can vary greatly over time, which in turn can lead to considerable uncertainty in the optimized parameters.

As shown in Section 2.1 and 2.2, a large number of existing methods for online feature selection and concept drift detection use an underlying online learning model. In particular, this involves online wrapper methods (Section 2.1.1) and error rate-based drift detection models (Section 2.2.2). To obtain stable feature sets and to avoid false drift alarms, we require robust model behavior. Accordingly, uncertain model parameters can cause problems for both online feature selection and concept drift detection if they are not taken into account. On the other hand, if we were able to quantify the uncertainty of the model, we might use this information to develop more powerful and reliable algorithms.

Based on this intuition, we propose a simple framework for estimating parameter uncertainty over time (Paper 1). Let $\theta = [\theta_1, \dots, \theta_k, \dots, \theta_K]$ be the parameter vector of an online learning model. Variations of the parameters θ are our best estimate of the uncertainty of the model. In order to quantify this uncertainty, we can treat the model parameters θ as random variables, which are parameterized by ψ (sufficient statistics). Given a data stream up until time step T, we aim to find the set of distribution parameters Ψ_T^* that maximize the log-likelihood $\mathcal{L}$:

$$\Psi_T^* = \arg\max_{\Psi_T} \mathcal{L}(\Psi_T, X_T, Y_T) = \arg\max_{\Psi_T} \sum_{t=1}^{T} \log P(y_t | x_t, \psi_t), \tag{2.4}$$

where

$$P(y_t|x_t, \psi_t) = \int P(y_t|x_t, \theta_t) P(\theta_t|\psi_t) \, d\theta_t, \tag{2.5}$$

is the marginal likelihood at time step t. Generally, we cannot solve Eq. (2.4) analytically. Instead, we may incrementally update the distribution parameters ψ over time. To this end, we can use a gradient-based optimization strategy such as stochastic gradient descent (SGD). SGD is an efficient and popular incremental optimization algorithm that is well suited for evolving data streams. In particular, if the data distribution is stationary, SGD converges almost surely to a local optimum. At the same time, if concept drift makes a previously learned parameter distribution obsolete, the gradient step at time step t resembles, in the worst case, an optimization with random prior parameters ψ_t. Since we aim to maximize the log-likelihood, we ascent on the gradient. Accordingly, the distribution parameters at time step t can be updated as

$$\psi_{t+1} = \psi_t + \frac{\tau}{P(y_t|x_t, \psi_t)} \nabla_{\psi_t} P(y_t|x_t, \psi_t), \tag{2.6}$$

where the hyperparameter τ specifies the learning rate. In practice, we may also perform multiple gradient steps per time step, which can lead to faster convergence.

The above framework leaves the predictive model and the distribution of its parameters unspecified. It is therefore a very flexible solution. For an effective implementation of the framework, we may use independent, normally distributed model parameters (Paper 1). Accordingly, let $\mathcal{D}(\psi_k) = \mathcal{N}(\mu_k, \sigma_k)$ be the distribution of the parameter θ_k, where μ_k is the mean and σ_k is the standard deviation. Treating the parameters as independent has computational advantages and often leads to good results, as we show in Paper 1 and 2 in the context of online feature selection and concept drift detection. Besides, the normal distribution is found in various natural phenomena and has valuable properties that simplify calculations. In fact, for Generalized Linear Models with normally distributed weights, we can obtain a closed form solution for the gradients of μ_t and σ_t in Eq. (2.6) (see Paper 1). Similarly, we can obtain effective implementations of the proposed framework for deep neural networks and Soft Decision Trees [57, 91], using Monte Carlo sampling and the reparameterization trick (see Paper 1). If the parameter distributions are treated as independent, the framework can generally also easily handle changing parameterizations of the model. That is, we may delete and add parameters over time, allowing the framework to adapt to changing model complexity.

Although it is a straightforward approach, the framework presented above can be very powerful in practice. Indeed, it enables two novel approaches to online feature selection and concept drift detection.

2.3.1 FIRES - Stable Online Feature Selection

Online feature selection is an effective mechanism for improving predictive quality and interpretability in high-dimensional data streams. To learn from new observations and adapt to concept drifts, we need to update the set of selected features over time. Ideally, however, we only want to alter the selected features when the importance of the input features for the prediction has actually changed. At all other times, e.g., after marginal (local) updates of the model parameters, the selected features should remain stable (see Section 2.1.2). Therefore, in order to obtain both discriminative and stable feature sets, we aim to select the input features that offer the best trade-off between high importance and low uncertainty.

As mentioned earlier, the parameters of an online learning model can provide a straightforward quantification of the importance and uncertainty of each input feature. The close link between input features and model parameters is most evident in linear models where there is a one-to-one mapping between features and parameters. For the sake of illustration, we thus consider a linear model below. However, in Paper 1 we describe how our approach can also be applied to neural networks and Soft Decision Trees [57, 91], where there is a one-to-many relationship between features and parameters.

According to the framework presented in Section 2.3, the distribution of a model parameter θ_j contains information about the importance and uncertainty of the corresponding input feature. In particular, for independent normally distributed model parameters, the mean μ_j indicates importance and the standard deviation σ_j indicates uncertainty. On this basis, we present the *Fast, Interpretable and Robust Feature Evaluation and Selection (FIRES)*. FIRES weights input features in a trade-off between importance and uncertainty according to the following objective:

$$\begin{aligned}
&\arg\max_{\omega_T} \sum_{t=1}^{T} \left(\underbrace{\sum_{j=1}^{m} \omega_{tj}\mu_{tj}^2}_{\text{importance}} - \lambda_s \underbrace{\sum_{j=1}^{m} \omega_{tj}\sigma_{tj}^2}_{\text{uncertainty}} - \lambda_r \underbrace{\sum_{j=1}^{m} \omega_{tj}^2}_{\text{regularizer}} \right) \\
&= \arg\max_{\omega_T} \sum_{t=1}^{T} \sum_{j=1}^{m} \omega_{tj}(\mu_{tj}^2 - \lambda_s \sigma_{tj}^2 - \lambda_r \omega_{tj}) \qquad (2.7)
\end{aligned}$$

The additional regularization term prevents arbitrarily large feature weights. With the hyperparameters λ_s and λ_r we can control the influence of the uncertainty penalty and the regularization term. In particular, by increasing λ_s, we can reduce the optimal weight of features with high uncertainty. To calculate the weight of a particular feature j at time step t, we evaluate the corresponding derivative with respect to Eq. (2.7):

$$\begin{aligned}
\frac{\partial}{\partial \omega_{tj}} &= \mu_{tj}^2 - \lambda_s \sigma_{tj}^2 - 2\lambda_r \omega_{tj} \overset{!}{=} 0 \\
\Leftrightarrow\ -2\lambda_r \omega_{tj} &= -\mu_{tj}^2 + \lambda_s \sigma_{tj}^2 \\
\Leftrightarrow\ \omega_{tj}^* &= \frac{1}{2\lambda_r}\left(\mu_{tj}^2 - \lambda_s \sigma_{tj}^2\right) \qquad (2.8)
\end{aligned}$$

According to Eq. (2.8), features with low importance and high uncertainty receive the smallest weights for the final feature selection, and vice versa. In this way, we can jointly increase the discriminative power and stability of the selected features over time.

FIRES belongs to the group of wrapper methods (Section 2.1.1), and is one of the first approaches whose feature weights satisfy three meaningful properties, such as a consistent ranking of features for stable data concepts (see Paper 1). As a result, FIRES is able to generate stable and discriminative feature sets in a variety of applications. In our experiments, FIRES implemented on a Probit model outperformed existing approaches in terms of predictive performance, computation time and feature set stability (Eq. (2.1)).

2.3.2 ERICS - Global and Partial Concept Drift Detection

To achieve high discriminative power over time, online learning models need to adapt to concept drift. In this context, active concept drift detection methods (Section 2.2.2) can enable more targeted adjustments while providing additional information about the data generating process. However, existing drift detection methods often only consider the predictive outcome (e.g., the error), but not the uncertainty of the model.

As before, we argue that concept drift detection can benefit from taking model uncertainty into account. Accordingly, let θ_t be the parameter vector of the model at time step t. Since the parameters are updated incrementally with new observations, the likelihood of the parameters θ_t can usually be considered a good approximation to the true conditional target distribution $P_t(Y|X)$, i.e., $P_t(Y|X) \approx P(Y|X, \theta_t)$. In this sense, the model parameters θ_t represent our most current information about the data generating concept at time step t. Moreover, (drastic) changes of the optimal model parameters are a clear indication of concept drift. If we treat the model parameters as random variables according to the framework presented in Section 2.3, we can associate concept drift with a change of the optimal parameter distributions. Specifically, we may represent real concept drift between two time steps t_1 and t_2 by a difference in the marginal likelihood (Paper 2):

$$\begin{aligned}
& P(Y|X, \psi_{t_1}) \neq P(Y|X, \psi_{t_2}) \\
\Leftrightarrow\ & |P(Y|X, \psi_{t_1}) - P(Y|X, \psi_{t_2})| > 0 \\
\Leftrightarrow\ & \left| \int P(Y|X, \theta) \left[P(\theta|\psi_{t_1}) - P(\theta|\psi_{t_2}) \right] \, d\theta \right| > 0. \qquad (2.9)
\end{aligned}$$

Accordingly, concept drift is reflected in different distribution parameters ψ_{t_1} and ψ_{t_2}, e.g., in different means or variances for normally distributed model parameters.

Based on this intuition, we present a novel approach for the *Effective and Robust Identification of Concept Shift (ERICS)* (Paper 2). ERICS uses information-theoretic measures to detect concept drift based on changes in the distributional uncertainty of the optimal model parameters. Specifically, we may replace the integral in Eq. (2.9) with a proportional expression based on the differential entropy h and the Kullbach-Leibler (KL) divergence D_{KL} to rephrase the basic drift detection scheme as follows (see Paper 2):

$$\Big|\underbrace{h[P(\theta|\psi_{t_2})] - h[P(\theta|\psi_{t_1})]}_{\Delta\text{Uncertainty}} + \underbrace{D_{KL}[P(\theta|\psi_{t_2})\|P(\theta|\psi_{t_1})]}_{\Delta\text{Distribution}}\Big| > 0 \tag{2.10}$$

Accordingly, concept drift manifests itself in two intuitive ways: as a change in entropy (i.e., distributional uncertainty) and as a divergence between the optimal parameter distributions at the respective time steps. As mentioned in Section 2.3, when optimizing parameters with SGD, we know that the algorithm almost surely converges to a local optimum if the data generating distribution is stationary. Thus, as long as there is no concept drift, the difference in entropy and the KL divergence should decrease over time. Accordingly, for concept drift detection, we are generally only interested in an increase of both measures.

We may use Eq. (2.10) to continuously check for concept drift at all successive time steps. However, attempting to detect concept drift based on the parameter distributions in two isolated time steps may not be robust in practice. To mitigate noise and achieve more reliable drift detection, we can instead investigate the moving average of Eq. (2.10) in a fixed-size sliding window (Paper 2). Moreover, like FIRES (Section 2.3.1), ERICS can be implemented with different parameter distributions and predictive models. For example, as before, we can use a linear model with normally distributed parameters, which allows us to calculate the differential entropy and KL divergence in closed form.

If we treat the model parameters as independent random variables, we can also apply ERICS to each parameter individually. This allows us to detect partial (i.e., feature-wise) concept drift. In particular, for linear models, concept drift at a parameter directly indicates partial drift of the corresponding input feature. For larger models such as neural networks or Soft Decision Trees, we can apply the simple parameter aggregation techniques proposed in Paper 1 to achieve partial drift detection.

ERICS cannot be clearly assigned to one of the existing categories of concept drift detection methods (Section 2.2.2). It is most closely related to the error rate-based drift detectors. However, ERICS comes with the advantage that it does not need to use the predictive outcome, but directly examines the parameterization of the model, which is much more sensitive to subtle and early changes in the data generating distribution. In this way, we can detect concept drift earlier and more reliably, as we have shown in experiments (Paper 2).

2.4 Discussion and Future Work

In this chapter, we presented new approaches for online feature selection and concept drift detection – two of the most challenging tasks in online learning. In the corresponding papers (Paper 1 and 2), we studied the limitations of previous work and highlighted the practical advantages of FIRES and ERICS. In the following, we briefly summarize our main findings and open questions.

The simple probabilistic framework underlying FIRES and ERICS is an effective approach to quantifying parameter uncertainty over time. Apart from its use in FIRES and ERICS, the framework provides an intuitive mechanism for detecting uncertain parts of the model. In this way, we can enable more informed training or debugging and contribute to better overall interpretability.

However, the performance of both FIRES and ERICS is closely linked to the predictive model via the proposed framework. If the model is not able to represent the data generating concept well, the parameter distributions are usually not very meaningful. Therefore, we need to ensure that our framework is used in combination with reliable online learning methods. The implementations of FIRES and ERICS with a linear model have performed well in experiments (see Paper 1 and 2). In fact, we did not find a considerable advantage in extracting importance and uncertainty estimates from the parameters of more complex models. In general, the predictive power and flexibility of complex nonlinear models could be better exploited, if we used more advanced strategies for aggregating parameter statistics. However, this flexibility comes at a drastically increased computational cost. Thus, given the strong performance and high efficiency of the simple linear implementations of FIRES and ERICS in a variety of applications, we argue that a simple scheme should usually be sufficient.

Like most online feature selection approaches, FIRES requires the user to specify the number of selected features. However, as mentioned earlier, it is usually difficult to choose an appropriate feature set size. In the future, one could therefore experiment with thresholds applied to the obtained feature weight vector to determine the number of features more dynamically. Alternatively, one might replace the squared ℓ_2 regularization term in Eq. (2.7) with a different penalty (e.g., the ℓ_1 norm) to enforce sparser feature weights.

Although FIRES is a wrapper method, it could also be embedded in an online learning framework because of its efficiency and flexibility. In particular, we could use FIRES to reduce the set of eligible split candidates at each node of an incremental decision tree or to reduce the dimensionality of linear models learned at the leaf nodes (see Section 3.1.4). Besides, in Paper 1, we showed that FIRES is able to identify salient areas in the images of the MNIST data set. In the future, one might extend this analysis and exploit the proposed framework as an effective and principled global explanation mechanism for evolving data streams.

Concept drift detection methods like ERICS are often sensitive to the configuration of the hyperparameters. In general, there is no particular configuration that works well for all applications. Therefore, it can often make sense to re-evaluate the hyperparameter configuration of a drift detector at frequent intervals. In the future, one could try reduce the number of adjustable hyperparameters in ERICS, e.g., by replacing the fixed size sliding windows with an adaptive parameter-free windowing scheme like ADWIN [17] (see Section 2.2.1).

While we demonstrated the ability of ERICS to detect concept drifts in experiments, we have not yet embedded ERICS in an online learning model. Therefore, future work should investigate how ERICS changes the predictive performance of popular online learning methods that rely on active concept drift detection (e.g., adaptive versions of the Hoeffding Tree [18] presented in Section 3.1.4).

Chapter 3

Reliable and Interpretable Online Classification

Classification is one of the most common applications of offline and online machine learning. The goal of a classifier is to predict the correct label among two or more possible target classes for a given observation. For example, in spam detection, we aim to classify an email as either genuine or spam. In general, a classifier should offer high discriminative power, while trying to reduce time and memory requirements. In addition, classifiers should be interpretable – especially when they are used for high stakes decisions, e.g., in healthcare or credit scoring. Interpretability is not only required by regulations such as the European Union's General Data Protection Regulation (GDPR), but also helps build confidence in the model and predictions. In this chapter, we give an overview of the current state of the art in online classification. In this context, we also briefly address the interpretability of online learning models, which has been largely neglected in the past. Finally, we introduce the Dynamic Model Tree, a novel incremental decision tree framework that has valuable properties to achieve a more interpretable and thus reliable classification in data streams. The contributions of this chapter are based on Paper 4.

3.1 Background on Online Classification

Similar to concept drift detection and online feature selection methods (Chapter 2), online classification models are trained incrementally to learn from new observations and adjust to concept drift [106]. In particular, an online classifier should be able to learn new data concepts while retaining previously learned information that is still relevant [67, 106]. This trade-off can be difficult, as (sudden) concept drift may require a quick adjustment of the classifier, while wrong and extensive retraining often affects the predictive performance [67, 106].

Online classification models should adhere to the general requirements mentioned in the introduction (Section 1.1). Similarly, previous work has established a set of important properties for online classifiers [47, 48, 8], the most relevant of which we repeat below:

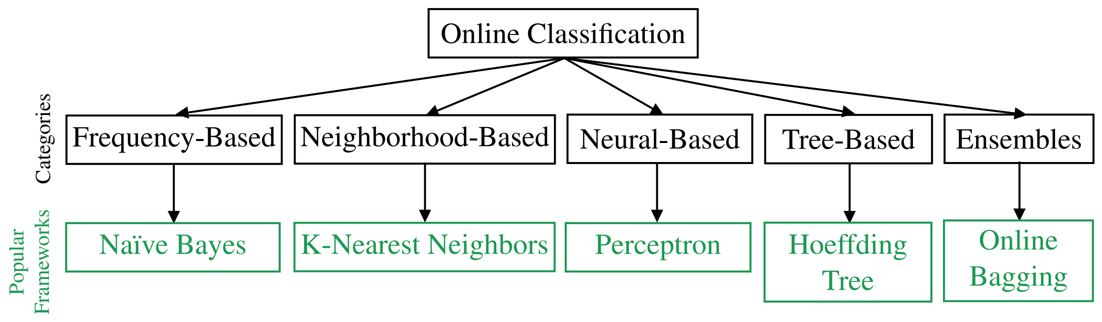


Figure 3.1: **Taxonomy of Online Classifiers.** Bahri et al. [8] distinguished five categories of online classifiers. Above, we show a corresponding illustration that closely follows the original work [8]. We also name a popular framework for each category (green), which is described below. The focus of this thesis is on tree-based online classifiers such as the Hoeffding Tree [46], as this is the most popular group of online classifiers with strong benefits in terms of flexibility and efficiency.

- Observations should be processed in small constant time, using a limited amount of memory.
- As a consequence of the above, an online classifier should deal with a single pass over incoming observations.
- Besides, an online classifier should be able to perform predictions at any time.
- Finally, an online classifier should be able to handle both periods of stationary concept and concept drift.

In general, there are two alternative approaches for processing streaming data [8, 142]. In the batch-incremental approach, the classifier is updated as soon as a new batch of observations (of a given size) is available. In the instance-incremental approach, the classifier learns from each observation as it arrives. Since instance-incremental processing is more time and memory efficient, it is more in line with the desiderata mentioned above. Indeed, most of the models discussed in this chapter use instance-incremental processing.

Bahri et al. [8] proposed a simple taxonomy of online classifiers that we adopt in this thesis (see Figure 3.1). The focus of our work is on tree-based online classifiers. This well-explored group of online learning models has been shown to offer a good trade-off between efficient computation, effective adaptation to concept drift, and strong predictive performance. Tree-based classifiers are also very flexible, e.g., they can be easily combined into an ensemble. Nevertheless, we also briefly present popular models from the remaining categories. A more detailed summary of online classification methods and the state of the art can be found in recent surveys [126, 45, 106].

3.1.1 Naïve Bayes

Naïve Bayes is a well-known probabilistic classifier that applies Bayes' theorem to infer the posterior class probabilities from given observations. To do this efficiently, the classifier "naïvely" assumes that the input features are independent given the target class.

A Naïve Bayes classifier is trained incrementally and can therefore be applied to data streams out of the box. The original algorithm can also be extended by sketching techniques to maintain approximate class frequencies for more efficiency in the online environment [7]. However, Naïve Bayes does not incorporate an explicit mechanism to handle concept drift. To be able to adjust to (sudden) concept drift, the Naïve Bayes classifier should thus be combined with an active drift detection model (see Section 2.2.2) [7].

3.1.2 K-Nearest Neighbors

The *k*-nearest neighbors (kNN) model is one of the best-known and most intuitive classifiers. Given a test observation, kNN identifies the *k* closest observations according to some distance metric (e.g., the Euclidean distance). The classifier then predicts the label of the test observation by majority voting among the labels of these *k* neighboring observations.

The kNN classifier is a "lazy learner", because it does not need to train parameters. Instead, kNN requires access to a set of observations that serve as possible neighbors. Since it would be infeasible to store all streaming observations over time, online adaptations of kNN typically use sliding windows (Section 2.2.1), which also helps to adjust to concept drift [8]. Indeed, simple implementations of the kNN classifier show surprisingly good performance on common benchmark streaming data sets [142]. However, kNN does not provide information about the importance of each input feature. Besides, as already mentioned, choosing the right size of sliding window can be difficult. Bifet et al. [22] showed that there is typically a trade-off between small windows with high efficiency and large windows with high predictive power. As a compromise, we might use a dual-memory strategy, where a short-term and a long-term memory represent current and past data generating concepts respectively [105].

3.1.3 Perceptron and Deep Neural Networks

Neural networks are a powerful solution for various classification tasks. However, modern deep learning techniques tend to be computationally intensive. For example, while recurrent neural networks have been successfully used in other dynamic applications like speech processing or time series forecasting, they are generally too complex to meet the strict memory and real-time requirements of a data stream [8].

In the online learning domain, we therefore usually only find very simple neural network architectures like the Perceptron. The Perceptron algorithm is a single-layer neural network that can be used for online classification. Compared to the traditional implementation of the Perceptron, the online version does not use multiple training iterations (epochs) over the data. Instead, the weights are updated once for each incoming observation with stochastic gradient descent [8].

Still, there are also approaches to using advanced neural networks for machine learning in evolving data streams. For example, Sahoo et al. [149] introduced a scalable deep neural network architecture. Specifically, they proposed to add a classifier to every hidden layer and combine the individual outputs for the final prediction. The weights of the individual classifiers are learned incrementally, so that the weighting of the lower-performing classifiers decreases over time. In this way, the influence of each hidden layer is dynamically adjusted. Likewise, Pratama et al. [139] presented a randomized neural network that can scale to both instance-incremental and batch-incremental online learning. Finally, there are also approaches to combine deep learning with online learning techniques such as concept drift detection to improve training efficiency [53].

Apart from efficiency problems, however, neural networks have other weaknesses in the context of data streams [8]. Specifically, neural networks are often sensitive to the configuration of hyperparameters and therefore usually require extensive tuning, which is infeasible in an online application. In addition, neural networks often have disadvantages on heterogeneous tabular data [29], which is the most common data type in online learning. Therefore, neural networks are rarely used for online classification [8].

3.1.4 Tree-Based Online Classifiers

Most state of the art online learning frameworks are based on incrementally trained decision trees. Decision trees are efficient, intuitive, and usually deliver good predictive performance. Moreover, through the effective use of pruning, decision trees allow to discard outdated information more easily than linear models, neighborhood-based approaches or neural networks. In the following, we introduce the two incremental decision tree frameworks that are relevant to our work.

Hoeffding Tree

The Hoeffding Tree, first introduced by Domingos and Hulten [46], is the most popular incremental decision tree framework. Like all online learning methods, Hoeffding Trees do not have access to a full training data set at any point in time. Instead, Hoeffding Trees apply Hoeffding's inequality to determine at which point there is sufficient evidence (i.e., a sufficiently large number of streaming observations) to split a leaf node and thereby grow the tree.

In line with Domingos and Hulten [46], let r be a random variable in the range R and let $\bar{r}$ be the empirical mean of that random variable over n independent observations. According to Hoeffding's inequality, the true mean of the variable is at least $\bar{r} - \varepsilon$ with probability $1-\delta$, where δ is a hyperparameter and

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}. \tag{3.1}$$

Let then G be a heuristic gain measure (e.g., the Information Gain) used to compare split candidates (i.e., feature-value pairs that may be used to separate observations at a node). According to Domingos and Hulten [46], Hoeffding's inequality allows us to determine the time step at which the gain of a split candidate a is significantly greater than the gain of a second candidate b. Indeed, Hoeffding's inequality ensures that with probability $1-\delta$ the candidate a chosen after n observations would also be optimal after infinitely many observations. Formally, if $\bar{G}_a - \bar{G}_b > \varepsilon$ after n observations, then a is the best split candidate with probability $1-\delta$ [46]. Therefore, we would split the node on the candidate a and continue to grow the tree. Domingos and Hulten [46] show that a Hoeffding Tree trained in this way guarantees a high asymptotic similarity with the corresponding batch tree trained on the full data set.

The Very Fast Decision Tree (VFDT) is an early implementation of the Hoeffding Tree [46]. In addition to the core algorithm outlined above, VFDT applies a number of extensions to improve efficiency and predictive power. For example, VFDT introduces a threshold to break ties between split candidates with similar gains. Moreover, the memory usage of the algorithm is reduced by deleting split candidates whose gain diverges too far from the currently optimal gain. More details about the VFDT algorithm can be found in the corresponding paper.

A major disadvantage of the VFDT algorithm is that it does not take concept drift into account. In fact, Domingos and Hulten [46] assumed that the data generating process is stationary. Accordingly, split decisions made by the VFDT may no longer be optimal in the future due to concept drift, making old nodes and branches of the tree obsolete. To overcome this problem, Hulten et al. [88] introduced an adaptive version of the VFDT, called CVFDT, which revisits old split decisions. CVFDT keeps node statistics consistent with a sliding window of recent observations (Section 2.2.1). In this way, CVFDT is able to periodically revisit and replace outdated splits. Specifically, CVFDT begins to grow an alternate subtree at a node, once there is sufficient evidence that another split candidate might be more suitable. If the new subtree outperforms the old one, CVFDT replaces the obsolete branch.

Later, Bifet and Gavaldà [18] generalized this idea to the Hoeffding Window Tree (HWT). A HWT is any incremental decision tree that uses Hoeffding's inequality and a sliding window. For example, a HWT can be implemented with ADWIN [17] (see Section 2.2.1). If ADWIN detects concept drift at a node, the HWT starts growing an alternate subtree, which may ultimately replace the old branch [18]. According to Bifet and Gavaldà [18], the HWT constructs and replaces subtress faster than the CVFDT, using fewer hyperparameters. The Hoeffding Adaptive Tree (HAT) is an extension of the general HWT framework [18]. In HAT, the frequency statistics at the nodes are replaced by simple estimators (e.g., an exponentially weighted moving average or ADWIN). As a result, HAT can work without a fixed window size and is therefore more memory-efficient than HWT. Both HAT and HWT guarantee that the resulting tree is equivalent to the VFDT, if no concept drift is detected.

The Hoeffding Tree and its early implementations have been improved over the years. For the sake of brevity, we do not discuss each paper in full detail, but give a brief summary of some of the most important work.

The Hoeffding Anytime Tree (HATT) is less rigorous than earlier Hoeffding Trees regarding initial split decisions [111]. In particular, HATT applies a split as soon as it is beneficial in terms of predictive performance, even if it is not significantly better than the second-best split candidate. This initial split decision will be revisited and possibly replaced in the future. The Extremely Fast Decision Tree (EFDT) is a popular implementation of the HATT.

The basic VFDT algorithm may grow indefinitely, which can compromise its efficiency and interpretability. As mentioned above, sliding window schemes and concept drift detection methods are the most popular approaches to avoid an increasingly complex tree. However, there are other attempts to control the complexity of a Hoeffding Tree. For example, the approach of Losing et al. [107] monitors the class distribution of previous splits to approximate the minimum time to reach the Hoeffding bound. In this way, certain (unlikely) splits can be avoided in the first place. Alternatively, one can introduce regularization to reduce the complexity of the Hoeffding Tree [9, 10]. In particular, Barddal and Enembreck [10] proposed to reduce the tree depth by penalizing splits on features that had not been used before. At the same time, their method allows the reuse of a previously used feature only if the gain exceeds the maximum previous gain for that feature in the current branch [10].

In general, a Hoeffding Tree uses majority voting at the leaf nodes to obtain predictions. However, the leaves of Hoeffding Trees have also been augmented with simple predictive models to improve classification performance. For example, the VFDTc algorithm uses Naïve Bayes models (Section 3.1.1) [60]. Similarly, Bifet et al. [20] extended the Hoeffding Tree with Perceptron models (Section 3.1.3). Although such extensions usually improve predictive performance, Holmes et al. [83] also discovered situations in which a majority voting scheme proved more discriminative. In particular, they showed that noisy inputs lower the performance of a Naïve Bayes classifier. As a solution, Holmes et al. [83] suggested switching between majority voting and prediction by a simple model, depending on the current performance of the two approaches.

Limitations and Modifications of the Hoeffding Tree. Hoeffding Trees are intuitive, efficient and transparent. Besides, they are supported by a variety of online learning packages (see Section 5.1) and are thus easily accessible to a broad community. Still, the general Hoeffding Tree framework has been repeatedly questioned. In the following, we summarize some of its fundamental limitations.

Rutkowski et al. [147] claimed that Hoeffding's inequality is often too restrictive in practice, as it requires inputs that are numeric and can be expressed as a sum of independent variables. As a solution, the Hoeffing bound may be replaced with a different criterion, e.g., based on the more general McDiarmid's inequality [147]. The same authors also proposed a specific split criterion for the Gini gain function [148].

However, Matuszyk et al. [115] showed that existing implementations often also violate the independence assumption that is shared by both Hoeffding's and McDiarmid's inequality. Specifically, since the gain measurements are usually averaged over a sliding window, the gains in successive time steps relate to an overlapping set of observations. Therefore, the obtained gains are not independent. Matuszyk et al. [115] presented a simple workaround by adjusting the threshold according to Eq. (3.1). In addition, they proposed the Quality Gain function, which measures the possible improvement of a new leaf node over its parent in terms of accuracy. Unlike existing gain functions such as the Information Gain, Quality Gain can be safely combined with Hoeffding's inequality [115].

Although modern implementations of the Hoeffding Tree perform well in a variety of online applications, they need to overcome a number of limitations of the basic algorithm. Indeed, Manapragada et al. [112] found that most implementations of the Hoeffding Tree make a number of (unspecified) design decisions to improve performance. As mentioned above, these include mechanisms to adjust for concept drift, achieve theoretically sound split decisions, and improve predictive performance at the leaf nodes. However, many of these modifications are costly and may ultimately affect the interpretability and reliability of the Hoeffding Tree in practical applications.

Model Tree

As mentioned before, Hoeffding Trees have been augmented with simple models like Naïve Bayes [60] or Perceptron [20] to improve predictive performance. The Model Tree framework generalizes this idea. A Model Tree is a decision tree that maintains simple – typically linear – models at the leaf (and sometimes also inner) nodes. Through the hierarchical combination of these weak learners, Model Trees are able to approximate complex and non-linear functions [138], while remaining highly transparent. Model Trees have been successfully used in offline batch learning scenarios [140, 33]. However, Model Trees for online learning have received less attention.

Potts and Sammut [138] were one of the first to present an incremental training and pruning procedure for Model Trees. The authors introduced a new test statistic to evaluate whether all observations at a given node were generated by a single linear model (null hypothesis). Accordingly, in their approach a node is split if the null hypothesis can be rejected. Likewise, if the null hypothesis at an inner node can no longer be rejected after obtaining new training observations, the corresponding subtree is considered obsolete and pruned. At test time, each leaf node of the Model Tree generates a prediction. The predictions are then smoothly aggregated in a bottom-up approach, so that predictions that reach an inner node are weighted according to the distance of the predicted observation from the split value [138].

Compared to the Hoeffding Tree, the incremental Model Tree proposed by Potts and Sammut [138] is much more costly. Ikonomovska et al. [89] presented a more efficient alternative called Fast Incremental Model Tree with Concept Drift Detection (FIMT-DD). FIMT-DD uses the Standard Deviation Reduction measure as a gain criterion. Unlike the approach of Potts and Sammut [138], FIMT-DD does not exploit the simple models at the leaves (they used Perceptrons) to identify good split candidates. Rather, FIMT-DD adopts many ideas from the Hoeffding Tree framework. In particular, FIMT-DD applies Hoeffding's inequality to determine the right time for splitting. Moreover, FIMT-DD uses the Page-Hinkley test [131] (see Section 2.2) to detect concept drift and replace obsolete nodes and branches. Consequently, we argue that FIMT-DD could be considered both a Model Tree and a Hoeffding Tree.

The incremental Model Trees of Potts and Sammut [138] and Ikonomovska et al. [89] were introduced as a solution for online regression problems. However, because of the flexibility added by the simple models, Model Trees may often adapt better to the active data concept than standard Hoeffding Trees. Therefore, Model Trees should be investigated as an alternative for online classification.

3.1.5 Ensembles for Online Classification

Ensembles are an effective approach to combine the strengths of multiple weak learners for better predictions [8]. In practice, ensembles are often more powerful and versatile than stand-alone classifiers. Indeed, tree-based ensemble techniques such as XGBoost [40] still dominate machine learning on tabular data in the stationary setting [29].

There are several challenges when building an ensemble. In particular, one needs to select appropriate weak models. In general, the weak models should complement each other to form a diverse set of classifiers [100, 67]. In addition, one must decide how to combine the results of the individual weak learners, e.g., through a meaningful voting rule [67].

Ensembles offer distinct benefits for online machine learning. They are easily scalable by adding or deleting weak learners. Moreover, ensembles provide straightforward mechanisms for dealing with concept drift and outdated information. Specifically, we can dynamically change the voting rule over time to give less weight to low-performing weak learners [101]. Alternatively, we can periodically update or remove old weak learners and add new ones based on recent observations [8, 101]. In this context, many online ensembles apply passive adaptation strategies [100]. Yet, some ensembles also use active drift detection (Section 2.2.2) to identify outdated weak learners [66, 68, 173].

In the following, we briefly summarize popular online ensemble models. Most of these ensembles are implemented with tree-based online classifiers such as the Hoeffding Tree (see Section 3.1.4).

Oza [130] adapted the popular bagging and boosting techniques for machine learning on data streams. Traditional bagging approaches aim to construct a diverse ensemble by training the weak learners on different samples drawn with replacement from the training data. Oza [130] showed that the probability distribution of the number of times an observation is drawn converges to a $Poisson(\lambda = 1)$ distribution. Accordingly, in online bagging, each streaming observation should be used $k \sim Poisson(\lambda = 1)$ times to update a weak learner. This simulation of sampling with replacement using the Poisson distribution can also be used for online boosting. Generally, boosting ensembles combine weak learners in a sequential manner, so that observations misclassified in one weak learner are given higher weight in the subsequent learner. In the online boosting approach, the weight of misclassified observations can be adjusted by increasing the λ-parameter of the Poisson distribution. In this way, Oza [130] translated the popular AdaBoost [55] algorithm to the online case. A possible disadvantage of online boosting is that the training weights of new observations depend on the current performance of the ensemble. Therefore, depending on the order of streaming observations and the behavior of the underlying distribution, the training process and overall performance of an online boosting ensemble may change.

The basic online bagging and boosting approaches of Oza [130] have been improved and extended several times. The Leveraging Bagging algorithm aims to increase predictive performance of online bagging by using stronger randomization [19]. Specifically, Leveraging Bagging increases the λ-parameter of the Poisson distribution to further randomize the training data of each weak learner. In addition, the algorithm randomizes the output using error-correcting output codes, a technique that converts multi-class classification problems into a set of binary classification problems. Similarly, Gomes et al. [68] combined online bagging with streaming random patches to add randomization and thus increase the diversity of the weak learners. Moreover, Wang et al. [166] applied cost-sensitive ensemble techniques to improve the performance of online bagging and boosting in the presence of imbalances. Finally, Montiel et al. [119] introduced an online version of the eXtreme Gradient Boosting (XGBoost) algorithm [40], which is one of the most powerful and widely used machine learning models.

Random forests are another popular ensemble technique. Like bagging, random forests combine multiple weak learners, in this case decision trees, trained on different samples of the original data. In addition to standard bagging, a random forest allows only a small random subset of features to be considered for splitting at each node. In this way, the diversity of the weak learners is further increased. The Adaptive Random Forest (ARF) is an online adaptation of this general approach [66]. The ARF uses the same sampling strategy as online bagging [130]. In addition, the ARF incorporates an active drift detection method such as ADWIN [17] or Page-Hinkley [131] (Section 2.2.2). Specifically, when the drift detector of a weak learner issues a warning, the ARF starts training an alternate decision tree. In this way, the ARF can immediately replace an outdated weak learner as soon as a concept drift is detected.

The Hoeffding Tree (Section 3.1.4) is often used as a weak model type in online ensembles. However, its asymptotic convergence guarantees actually restrict the usefulness of the Hoeffding Tree as a diverse (unstable) weak learner [113, 68]. In this context, dynamic versions of the Hoeffding Tree with lower short-term stability such as the EFDT [111] are more suitable. Indeed, using EFDT as a weak learner can improve the performance of many of the ensemble techniques presented above [113].

Likewise, online ensembles can be constructed with weak model types that are not based on decision trees. For example, the DXMiner [114] that we discussed in the context of online feature selection (Section 2.1.1) uses nearest neighbor models. Similarly, the HEFT approach [125] (also introduced in Section 2.1.1) was combined with online Naïve Bayes as weak learners. Even neural networks have already been used as weak learners in an online ensemble [65].

Ensembles have become an extremely popular solution for online classification. However, ensemble techniques are inherently more complex than stand-alone classifiers. The resource consumption can be reduced through distributed computing and parallelization [8, 67], but the high complexity mitigates the interpretability of ensembles. That is, by combining the outputs of a dynamic set of weak learners, predictions of an online ensemble become more difficult to understand [67]. Krawczyk et al. [100] summarized and categorized online ensemble approaches according to different criteria. Similarly, Gomes et al. [67] proposed an extensive taxonomy of online ensembles, which we recommend for further reference.

3.2 On the Interpretability of Online Learning Methods

The term "interpretability" describes how well the inner workings of a model can be understood by a human. Accordingly, interpretability is highly subjective [51, 146]. To assess the interpretability of a machine learning model, we therefore usually resort to heuristic measures of complexity [15]. For example, the interpretability of a decision tree is often represented by the tree depth (local interpretability) or the number of nodes and leaves (global interpretability) [122]. Yet, it can be challenging to compare the complexity of different types of models [15]. In general, the fewer parameters a model uses in the prediction, the more interpretable it can be considered.

Surprisingly, interpretability in online learning has not received much attention in the past. In general, however, we can adopt the heuristic understanding of interpretability described above. Accordingly, we may say that an online learning model is interpretable if it uses a small constant number of parameters over time. This applies, for example, to linear models such as Naïve Bayes or the Perceptron (Section 3.1). However, most modern online learning models do not have a fixed number of parameters, but change their complexity over time.

The popular Hoeffding Tree framework is subject to considerable limitations that restrict its interpretability (see Section 3.1.4). In particular, the basic Hoeffding Tree contains no inherent mechanism for adjusting to concept drift. Accordingly, Hoeffding Trees would grow indefinitely (see early implementations, e.g., VFDT [46]) without using explicit mechanisms to prune obsolete branches. However, most extensions of the Hoeffding Tree, such as active drift detection methods (see Section 2.2), are chosen heuristically and typically increase complexity again.

In general, the complexity of online learning methods is dynamic. Indeed, complexity can often change suddenly as the model adapts to concept drift. Unfortunately, it is still unclear how changes in complexity, e.g., due to pruning, additional ensemble components or adaptive window sizes, affect the interpretability of online learning models. There is no common understanding of good interpretability in evolving data streams. However, given the public's increasing sensitivity to interpretable and fair machine learning, interpretability should play a greater role in the development of online learning methods. Specifically, we argue that there is a strong demand for a new and inherently interpretable online classification framework to replace the state of the art Hoeffding Trees.

3.3 Dynamic Model Tree

Existing and powerful online classifiers can exhibit low interpretability in practice. For example, Hoeffding Trees and tree-based ensembles (Section 3.1) often rely on heuristic gain functions or increase in complexity over time. As a result, the inner workings of these models can become less comprehensible. In this section, we present the *Dynamic Model Tree* (Paper 4), a novel online learning framework that combines the efficiency and extensibility of existing methods with greater flexibility regarding concept drift and stronger overall interpretability.

The Dynamic Model Tree (DMT) is based on the Model Tree framework presented in Section 3.1.4. Unlike previous Model Trees for data streams [138, 89], the DMT maintains simple predictive models at both the inner and leaf nodes. These simple models allow us to find new split candidates, identify outdated splits, and make predictions. Therefore, we need to update the simple models over time. We can represent each node of the DMT by a set of time indices corresponding to observations that passed through that node. Let $S_t \subseteq \{1,\dots,t\}$ be such a set of time indices. Accordingly, given a node represented by S_t, X_{S_t} and Y_{S_t} denote the corresponding observations and labels, and Θ_{S_t} the parameters of the simple model. Our goal is to optimize the model parameters with respect to a loss function $L(\cdot) \geq 0$:

$$\begin{aligned} \Theta^*_{S_t} &= \underset{\Theta_{S_t}}{\arg\min}\; L(\Theta_{S_t}, Y_{S_t}, X_{S_t}) \\ &= \underset{\Theta_{S_t}}{\arg\min} \sum_{t \in S_t} L(\theta_t, Y_t, X_t) \end{aligned} \tag{3.2}$$

We train the model parameters incrementally over time. In particular, we may use stochastic gradient descent, where θ_{t-1} serves as the prior parameter for the optimization at time step t. Based on Eq. (3.2), we update the simple model of a node whenever it receives a new observation.

Now, instead of using heuristic gain functions to identify split candidates, the simple models allow us to select candidates based on the estimated loss. To this end, we introduce new and node-specific gain functions. Suppose we are at a leaf node of the DMT represented by the set S_t. We look for the split candidate that would offer the greatest improvement in terms of the current loss. Let $C_t \subseteq S_t$ be the set of time steps representing the left child of a potential split and let $\bar{C}_t = S_t \backslash C_t$ be the set representing the right child. In other words, C_t and $\bar{C}_t$ correspond to the observations in S_t that would have been assigned to a potential left and right child node, respectively. On this basis, we can express the ideal split candidate at a leaf node as follows:

$$\begin{aligned} C_t^* &= \arg\max_{C_t} \; G_{S_t,C_t}, \text{ with} \\ G_{S_t,C_t} &= L(\Theta_{S_t}, Y_{S_t}, X_{S_t}) - L(\Theta_{C_t}, Y_{C_t}, X_{C_t}) - L(\Theta_{\bar{C}_t}, Y_{\bar{C}_t}, X_{\bar{C}_t}), \end{aligned} \tag{3.3}$$

where $L(\Theta_{C_t}, Y_{C_t}, X_{C_t})$ is the loss of the possible left child and $L(\Theta_{\bar{C}_t}, Y_{\bar{C}_t}, X_{\bar{C}_t})$ is the loss of the possible right child. According to Eq. (3.3), we would split the leaf node whenever there is a candidate that allows us to reduce the current loss.

We can define similar gain functions to re-evaluate the existing splits at the inner nodes of the DMT. As before, let I_t be a set of time indices representing an inner node. Each inner node is the root of a subtree (branch) of the DMT. Hence, to re-evaluate an existing split, we must check whether the current loss can be reduced by removing the corresponding subtree. Specifically, we can either make the inner node a leaf or replace it with a new inner node and two corresponding children. We propose the following two gain functions:

$$G_{I_t,C_t} = \sum_{J_t \subseteq I_t} L(\Theta_{J_t}, Y_{J_t}, X_{J_t}) - L(\Theta_{C_t}, Y_{C_t}, X_{C_t}) - L(\Theta_{\bar{C}_t}, Y_{\bar{C}_t}, X_{\bar{C}_t}), \tag{3.4}$$

$$G_{I_t} = \sum_{J_t \subseteq I_t} L(\Theta_{J_t}, Y_{J_t}, X_{J_t}) - L(\Theta_{I_t}, Y_{I_t}, X_{I_t}), \tag{3.5}$$

where each $L(\Theta_{J_t}, Y_{J_t}, X_{J_t})$ denotes the loss at one leaf of the subtree and $L(\Theta_{I_t}, Y_{I_t}, X_{I_t})$ is the loss at the current inner node. Accordingly, Eq. (3.4) measures the gain when we replace the current split with a new split (and thus replace the current subtree with a new subtree containing only two leaf nodes). Conversely, Eq. (3.5) corresponds to the gain when we make the current inner node a leaf node. In practice, we calculate both gains and choose the adjustment that brings the greater loss reduction (or larger gain).

To improve the training efficiency of the DMT, we incorporate the gradient-based algorithm of Broelemann and Kasneci [33], which allows us to approximate the candidate loss $L(\Theta_{C_t}, Y_{C_t}, X_{C_t})$ and $L(\Theta_{\bar{C}_t}, Y_{\bar{C}_t}, X_{\bar{C}_t})$ without training corresponding simple models. In this way, we can consider a large number of split candidates without risking computational overload. In addition, to improve the robustness of split and prune decisions, we introduce a threshold for the gain functions. Our threshold is based on the Akaike Information Criterion and can be dynamically adjusted to control the sensitivity and response time of the DMT.

The proposed framework can be used with different simple model types and loss functions, making it a flexible solution for various online applications. The DMT also adheres to two sensible properties that enable more meaningful and interpretable updates (see Paper 4). In particular, unlike previous work, the DMT guarantees that nodes and branches that no longer contribute to the overall loss are pruned. Consequently, the DMT achieves low complexity and adjusts to concept drift without using heuristic measures or separate drift detection models. In experiments, the DMT was able to outperform existing classifiers – especially in the presence of concept drift – while remaining much shallower than state of the art Hoeffding Trees (Paper 4).

3.4 Discussion and Future Work

In this chapter, we presented some of the most well-known frameworks for online classification and discussed their limitations – especially in terms of interpretability. In a corresponding publication (Paper 4), we introduced the Dynamic Model Tree (DMT), a novel and powerful online learning framework. In the following, we briefly summarize important results and point to future work.

The DMT is one of the first online learning frameworks to explicitly address interpretability in data streams. In this context, the DMT eliminates some of the most fundamental limitations of existing incremental decision trees. These include heuristic gain functions, active concept drift detection methods, and invalid applications of Hoeffing's inequality (see the limitations in Section 3.1.4). As a result, the DMT requires a fraction of the complexity of existing models, while delivering state of the art predictive quality. In fact, in our experiments, a stand-alone DMT was able to outperform an online ensemble of Hoeffding Trees (Section 3.1). In the future, it could also be interesting to evaluate the predictive performance of an ensemble of DMTs.

The DMT is extremely flexible. By switching the simple model or the loss function, the proposed framework can be used for both online classification and regression. While we have demonstrated the performance of the DMT as a classifier, we have not yet implemented it for online regression. In any case, the performance of the DMT depends heavily on the simple models. If the simple models are not robust or discriminative, the split and prune decisions will suffer. We implemented the DMT with simple linear models (logit and multinomial logit), which worked well in our experiments. In particular, we found that our implementation has advantages when there are linear relationships in the data. In the future, however, one could experiment with different simple models or optimization techniques (e.g., SGD with momentum).

Additionally, one might extend the DMT with the probabilistic framework presented in Section 2.3. To be precise, we could treat the parameters of the simple models as random variables. In this way, we would obtain a quantification of the importance and uncertainty of input features that could serve as an explanatory mechanism on different hierarchies. Also, we could apply FIRES (Section 2.3.1) to select important features. In this way, we could reduce the number of eligible split candidates at each node, which in turn would improve the efficiency of the DMT. Accordingly, we believe that a combination of the frameworks presented in Chapter 2 and 3 could lead to more efficient, interpretable and reliable online machine learning than before.

Chapter 4

Efficient and Effective Local Explainability in Data Streams

Automated decisions can have considerable practical implications, e.g., in job application systems, university admission platforms, or credit scoring. In these contexts, it can be crucial that we are able to mitigate bias [129] and ensure fair and transparent decisions. Otherwise, users may lose confidence in the underlying models and techniques, as the infamous case of racial bias in risk assessment of defendants has shown [4]. Therefore, automated decisions should be transparent and explainable when used in highly sensitive applications. In fact, the demand for transparency in AI is also reflected in new legislation [74]. Accordingly, it is becoming increasingly important to be able to describe the output and inner workings of a machine learning model in an understandable way.

In Chapter 3, we discussed the development of inherently interpretable online classifiers. Here, we focus on explanation techniques. While the terms interpretability and explainability are often used synonymously, we consider interpretability to be an inherent property of the predictive model, whereas explainability is typically achieved by post-hoc and external methods. Like interpretable online learning, explanation methods for evolving data streams have not received much attention in the past. Indeed, it is often unclear how common explanation methods can be applied to online learning models and how the generated explanations change in the face of incremental model updates and concept drift.

In this chapter, we look at local explanation techniques in data streams, in particular the popular family of feature attribution methods. We summarize important properties of local attributions and demonstrate their behavior under incremental model updates and concept drift. On this basis, we present a new change detection framework that enables more efficient and temporally coherent attributions in data streams. Finally, we discuss the choice of a meaningful attribution baseline, which is an important hyperparameter of many feature attribution methods. This chapter is based on the results presented in Paper 3 and 5.

4.1 Background on Local Attribution-Based Explainability

Many real-world machine learning tasks cannot be adequately handled with simple (e.g., linear) models, but require more powerful and complex techniques. State of the art models based on deep neural networks or ensembles achieve high predictive quality in a variety of scenarios, but are not inherently interpretable. In this context, explanation methods help to achieve the level of transparency required in critical applications. While explainability has been thoroughly studied in the stationary setting [116, 146, 51, 39, 71, 36], it remains largely unexplored in the data stream domain.

In general, post-hoc explanation methods can deliver explainability either at the global or local level [51]. While global explanations describe the general behavior of the model, local explanations refer to a specific observation (e.g., a customer) [71]. *Local feature attribution methods* are among the best-known and most widely used post-hoc explanation techniques. They quantify the local importance of each input feature for the prediction. In general, we can distinguish between model-agnostic and model-specific attribution methods. Model-specific approaches exploit the functional form of the predictive model, e.g., to calculate attributions more efficiently. For example, there exists a variety of attribution methods for decision trees [110] and neural networks [96, 54, 151, 152, 158]. On the other hand, model-agnostic approaches [143, 109] are applicable to any kind of complex model. Therefore, they can also be used to explain complex online learning models such as very deep Hoeffding Trees or online ensembles (Section 3.1).

There are several high-quality surveys that summarize popular local attribution methods, e.g., Guidotti et al. [71] and Carvalho et al. [36]. For this reason, and because we focus on the general behavior of local attributions in data streams, we do not provide an overview of the different methods. Instead, we introduce important attribution properties and the baseline hyperparameter. In addition, we briefly present existing work that has looked at local attributions in evolving data streams.

4.1.1 Attribution Properties

Local feature attributions should fulfil a number of meaningful properties to ensure their reliability in practice. Lundberg and Lee [109] proposed three generic properties that are widely recognized in the literature. We introduce these properties below.

Let f be the complex predictive model and let g be a local attribution method. According to Lundberg and Lee [109], we can represent the observation to be explained x by a simplified vector $x' \in \{0,1\}^m$. The simplified input vector x' represents missing features by zeros and all remaining features by ones. In addition, let h_x be a function of x that maps the simplified input vector to the original observation, i.e., $x = h_x(x')$.

The first property defined by Lundberg and Lee [109] is *local accuracy*:

$$f(x) = g(x') = \phi_0 + \sum_{j=1}^{m} \phi_j^{(x,f)} x'_j, \tag{4.1}$$

where $\phi_j^{(x,f)}$ is the attribution of feature j with respect to the observation x and the model f. The value ϕ_0 serves as a baseline. According to Lundberg and Lee [109], the baseline corresponds to the model outcome with all features missing, i.e., $\phi_0 = f(h_x(\vec{0}))$. We discuss the role of the baseline in more detail in Section 4.1.2. Intuitively, local accuracy describes that the sum of the generated attributions must be equivalent to the difference between the original model outcome $f(x)$ and the baseline outcome ϕ_0. Local accuracy is thus an important property that allows us to represent the influence of each input feature as a concrete change in the predictive outcome.

The second property is *missingness*, which is defined as

$$x'_j = 0 \Rightarrow \phi_j^{(x,f)} = 0. \tag{4.2}$$

According to Eq. (4.2), missing features should have no attributed impact on the prediction. This property primarily serves to complete the holistic attribution framework of Lundberg and Lee [109]. Since the observation to be explained usually has no missing features, the property is less relevant in practice.

The third and final property presented by Lundberg and Lee [109] concerns the *consistency* of local attributions. In particular, a local attribution is consistent between two complex models f and f' if for any simplified input vector $z' \in \{0,1\}^m$ the following holds:

$$\begin{aligned} f'(h_x(z')) - f'(h_x(z' \backslash j)) &\geq f(h_x(z')) - f(h_x(z' \backslash j)) \\ \Rightarrow \phi_j^{(x,f')} &\geq \phi_j^{(x,f)}, \end{aligned} \tag{4.3}$$

where $z' \backslash j$ denotes the simplified input z' with feature j missing, i.e., $z'_j = 0$. Consistency according to Eq. (4.3) guarantees that the attribution of an input feature does not decrease as its contribution to the model outcome increases.

For the family of additive feature attribution methods, Lundberg and Lee [109] showed that the Shapley value is the unique solution that guarantees all three of the above properties. However, variations of these properties can also be found in other works [158, 151, 157].

4.1.2 Attribution Baselines

The local accuracy property (Eq. (4.1)) due to Lundberg and Lee [109] establishes a baseline ϕ_0 that represents the model outcome for a vector of missing features. The baseline serves as a reference value for the generated attributions and thus defines their natural meaning. A baseline can be found in similar forms in other local attribution methods [143, 151, 158]. In general, the baseline should correspond to a value that represents the absence of discriminative information (missingness). In this way, a local attribution vector would express all the discriminative information contained in the corresponding observation. For example, in image recognition, the importance of pixels (features) is often measured using a black image as the baseline [151, 158].

The choice of an appropriate baseline can also be important for the evaluation and comparison of local attribution methods. In particular, the baseline is an important hyperparameter in evaluation methods based on *feature ablation tests* [84, 1, 144]. In an ablation test, the quality of attributions is assessed by gradually adding or removing highly rated input features and measuring the change in predictive performance. For example, if we compare the accuracy of a classifier before and after removing the feature with the strongest attribution, we would expect a large decrease in accuracy. Since machine learning models usually cannot handle arbitrary patterns of missing (ablated) features, the baseline serves as a surrogate value.

Heuristic approaches such as the black image baseline still dominate the literature. As an alternative, Izzo et al. [92] recently proposed the neutral baseline. The neutral baseline corresponds to a value that lies on the decision boundary of the classifier. Compared to heuristic baselines, the neutral baseline represents a more theoretically sound definition of missingness. However, depending on the model at hand, the neutral baseline may be hard to optimize, so its practical value is still limited.

In practice, the baseline can considerably influence the expressiveness and reliability of local attributions [157]. Indeed, Sturmfels et al. [156] showed that the quality of attribution methods in image classification is greatly affected by switching between different baselines. However, baselines have not yet been assessed for other areas of machine learning. These include tabular and streaming data, where the choice of baseline can be even more difficult. In data streams, for example, our understanding of missingness may change over time due to concept drift, so the baseline might have to be dynamic.

4.1.3 Local Attributions in Data Streams

Explainability is becoming increasingly important in various domains. However, with a few exceptions [159, 30, 43], feature attribution methods – or explanation methods in general – have not received much attention in the context of online learning. A possible reason for the lack of online attribution methods is that online learning models are often inherently simple and interpretable. For example, incremental decision trees already provide an intuitive way to extract local explanations in the form of decision rules by following the decision path of a particular observation. However, powerful online learning models are often limited in their interpretability by high or gradually increasing complexity (see Section 3.1). For this reason, local feature attribution methods can be useful to explain predictions in data streams.

In online learning, there is no real "post-hoc" state, as models are updated incrementally. Therefore, we have to compute explanations repeatedly over time. Indeed, Bosnić et al. [30] argued that explainability in online learning can only be achieved through a series of individual explanations. However, it is often unclear how long an attribution, once generated, will be valid in the future. Unfortunately, existing feature attribution methods do not allow us to update a local attribution along with changes of the online learning model. Accordingly, we need to explicitly monitor changes and recalculate old attributions if necessary.

In this context, changes of feature attributions over time may also be explored as a tool to detect concept drift. In fact, Demšar and Bosnić [43] showed that we can detect concept drift by measuring the dissimilarity between successive feature attributions. Specifically, the authors applied a Page-Hinkley test [131] (Section 2.2.2) to the Euclidean distance measurements between feature attributions at different time steps. However, for the detection of local concept drift at the instance level, the approach of Demšar and Bosnić [43] is too inefficient in practice.

Overall, the behavior, limitations, and potential value of local attributions for explainability in evolving data streams still need to be explored.

4.2 Behavior of Locally Accurate Attributions in Data Streams

According to the consistency property (Eq. (4.3)), a local attribution should not vary as long as the predictive model is stationary. However, this is an unrealistic assumption for streaming applications. In online learning, where the data distribution and model change over time, we expect the local attributions to change as well. Here, we investigate this behavior more formally (Paper 5).

Given that the decision boundary of an online learning model f_t changes between two time steps t_1 and t_2, we know that there must exist at least one point x, such that $f_{t_1}(x) \neq f_{t_2}(x)$. We assume that the point x has no missing features and thus corresponds to the simplified vector $x' = \vec{1}$ (see Section 4.1.1), which is a vector of all ones. With a locally accurate attribution method according to Eq. (4.1), we know that the baseline ϕ_0^t or the attribution $\phi^{(x,f_t)}$ of the observation in question must have changed:

$$f_{t_1}(x) \neq f_{t_2}(x) \overset{(4.1)}{\Leftrightarrow} \phi_0^{t_1} + \sum_{j=1}^{m} \phi_j^{(x,f_{t_1})} \neq \phi_0^{t_2} + \sum_{j=1}^{m} \phi_j^{(x,f_{t_2})}, \tag{4.4}$$

where $\phi_0^{t_1}$ and $\phi_0^{t_2}$ are the baselines at the time steps t_1 and t_2, respectively. Accordingly, due to local accuracy, every shift of the estimated decision boundary corresponds to a change in the baseline or a change in the local attribution of one or multiple instances. In a robust online learning model, shifts of the decision boundary are usually caused by

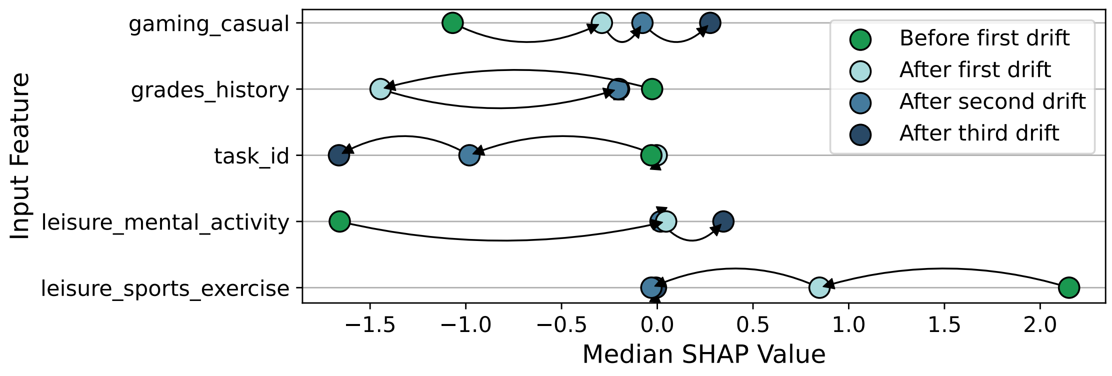


Figure 4.1: **Attribution Shift in Real-World Applications:** Local feature attributions can change over time due to updates of the predictive model and concept drift. In Paper 5, we investigate this effect for the TüEyeQ data set [94, 95]. TüEyeQ comprises socio-demographic information about 315 participants in an IQ study. The classification task is to distinguish failed from passed test items. TüEyeQ has three natural and sudden concept drifts by switching between different task blocks. Above, we depict the median SHAP [109] attributions of the participants for each task block. In particular, we show the attributions of the five features with the highest variability. This plot is taken from Paper 5, in which we also provide additional information on the data and the experimental setup. Strikingly, the median SHAP attributions change considerably over time. This behavior can be attributed to the local accuracy property (Eq. (4.1)) shared by most modern attribution methods. As a result, local attributions can typically only be considered valid as long as the online learning model is stationary. In this context, a local change detection mechanism (as presented in Section 4.3) can help to identify outdated attributions and enable more efficient recalculations.

concept drift. In this sense, changes in local attributions are often a direct consequence of concept drift. Indeed, this behavior can be observed in real-world applications (see Figure 4.1).

In general, local attributions can only be considered valid at the time they were generated. Without information on local changes in the predictive model, we would have to recalculate the feature attributions of previous observations after each model update to assess and ensure their validity.

4.3 CDLEEDS - Local Change Detection for Spatio-Temporally Coherent Attributions

Incremental model updates, e.g., due to concept drift, may shift the decision boundary of an online learning model. Due to the local accuracy (Eq. (4.1)) of most attribution methods, such shifts in turn lead to a difference between the attributions obtained at different time steps (see Eq. (4.4)). Without an approach to incrementally update the attributions, this difference can only be handled by repeatedly recomputing the local attributions over time. However, excessive recomputations are usually not feasible. Besides, in practice, we only want to recompute an attribution if it has actually changed. For this purpose, we need to be able to detect local changes in the online learning model.

Accordingly, we propose a novel approach for *Change Detection for Local Explainability in Evolving Data Streams (CDLEEDS)* (Paper 5). A naïve change detection scheme follows directly from the behavior of locally accurate attributions in data streams (see Section 4.2). In the following, we add a subscript (index i) to x_i in order to clearly indicate the observation to be explained. Based on Eq. (4.4), we can detect changes in a local attribution between two time steps without having to compute the actual attribution. Instead, we can infer changes of the attribution $\phi^{(x_i,f_t)}$ from a shift in the difference between the local model outcome $f_t(x_i)$ and the baseline ϕ_0^t:

$$\sum_{j=1}^{m} \phi_j^{(x_i,f_{t_1})} \neq \sum_{j=1}^{m} \phi_j^{(x_i,f_{t_2})} \overset{(4.1)}{\Leftrightarrow} \underbrace{f_{t_1}(x_i) - \phi_0^{t_1} \neq f_{t_2}(x_i) - \phi_0^{t_2}}_{\text{Naïve Local Change Detection}}. \tag{4.5}$$

The naïve scheme according to Eq. (4.5) might be prone to noise, since we compare the model in two isolated time steps. Also, for continuous change detection at the point x_i, we would need to obtain a prediction for each subsequent time step. Depending on the complexity of the model and the total number of observations we want to monitor, this may be too costly.

However, to infer local changes at a point x_i, it is usually sufficient to detect changes in nearby (similar) observations. In other words, if we can detect change in the neighborhood of x_i at time step t, we can assume that the decision boundary and the attribution at x_i have also changed. Based on this intuition, CDLEEDS introduces spatiotemporal neighborhoods to make the change detection scheme in Eq. (4.5) more feasible and robust. Specifically, the *spatiotemporal γ-neighborhood (STN)* of an observation x_i is a set of time steps corresponding to close previous observations:

$$\Omega^{(x_i,\gamma)} = \{t \in \{1,\ldots,T\} \mid \text{sim}(x_t,x_i) \geq \gamma\} \tag{4.6}$$

The STN of an observation can be adjusted either by the similarity threshold γ or the time interval considered (note that we have not restricted the time interval in Eq. (4.6)). Given the STNs in two consecutive intervals $\Omega_{<t}^{(x_i,\gamma)} = \{u \in \{1,\ldots,t-1\} \mid \text{sim}(x_u,x_i) \geq \gamma\}$ and $\Omega_{\geq t}^{(x_i,\gamma)} = \{v \in \{t,\ldots,T\} \mid \text{sim}(x_v,x_i) \geq \gamma\}$, we rephrase the initial scheme for local change detection at time step t:

$$\mathbb{E}_{u\in\Omega_{<t}^{(x_i,\gamma)}}[f_u(x_u) - \phi_0^u] \neq \mathbb{E}_{v\in\Omega_{\geq t}^{(x_i,\gamma)}}[f_v(x_v) - \phi_0^v] \tag{4.7}$$

We can treat the STNs as sliding windows to obtain a continuous monitoring of local change. In addition, we can use a hypothesis test to evaluate whether the mean values in the sliding windows differ significantly (see Paper 5).

In general, we want to detect local change not only for a single observation x_i, but for a large number of observations. In Paper 5, we propose a hierarchical clustering approach that allows us to implement the drift detection method presented above in a more efficient way. Specifically, we maintain only a small set of representative STNs (corresponding to the leaves of the hierarchical cluster) that allow us to approximate the STNs of each new streaming observation. The depth of the hierarchical clustering method can be controlled by γ, which specifies the minimum similarity within each leaf node.

CDLEEDS is model-agnostic and can be combined with any predictive model and attribution method. The clustering implementation of CDLEEDS is able to effectively detect local changes throughout the entire input space. In this way, CDLEEDS helps to drastically reduce the number of recalculations required to maintain meaningful attributions, as we showed in experiments (Paper 5). Moreover, the hierarchical clustering provides us with a simple mechanism to detect global changes. In fact, in our experiments, a CDLEEDS-based approach to global concept drift detection has often outperformed existing drift detectors (Section 2.2.2).

Accordingly, CDLEEDS is a powerful extension for modern feature attribution methods that enables more feasible and meaningful local explainability in data streams.

4.4 Choosing Baselines for Tabular and Streaming Data

As described in Section 4.1.2, the baseline is an important hyperparameter of many local attribution methods. Indeed, the baseline also plays a central role in the proposed change detection framework CDLEEDS (Section 4.3). In general, the choice of the baseline is difficult, especially for data streams where our notion of a good baseline may change over time.

The baseline is typically chosen heuristically and its influence on the generated attribution is often neglected. As one of the first works, Sturmfels et al. [156] presented and compared different baselines for image data. We complement their work in Paper 3, where we categorize seven common baselines according to two properties:

- **Static or Dynamic:** Existing baselines are either static, i.e., the same baseline is used for all attributions generated, or they are determined dynamically for each local attribution.

- **Deterministic or Stochastic:** Existing baselines are either deterministic or obtained in a stochastic way, e.g., by sampling.

A more formal definition of both properties can be found in Paper 3. This simple taxonomy facilitates the comparison and selection of baselines in practice.

We evaluated common baselines using ablation tests (see Section 4.1.2) on several tabular data sets (Paper 3). Strikingly, our experiments show that the explanatory power of attributions from popular methods like SHAP [109], DeepLift [151], and Integrated Gradients [158] is heavily influenced by the baseline. In fact, popular heuristic baselines like the all-zero vector [158] can lead to local attributions on tabular data that are not meaningful. In the extreme case, attributions based on a heuristic baseline may perform worse in the ablation test than a random attribution. Nevertheless, some baselines worked well in most of our experiments. In particular, the *expectation baseline* generally achieved top results in the ablation test. The expectation baseline corresponds to the expected model outcome for a random sample of training observations (e.g., as used by Lundberg and Lee [109]) and is thus a static and stochastic baseline.

While Paper 3 focuses on the effect of baselines in explaining offline learning models, we can use these insights to identify a meaningful baseline for data streams. As mentioned earlier, the expectation baseline is a good choice for most tabular data sets. However, the expectation baseline is based on a static sample of observations, which can lead to problems in data streams. Specifically, if the sample is no longer representative of the current data generating distribution, the expectation baseline will also no longer be representative, resulting in attributions that are not meaningful. Instead, we may specify the expectation baseline over a window of observations. For example, we can use a decayed windowing scheme (Section 2.2). In this case, the baseline corresponds to the exponentially weighted moving average (EWMA), a popular approach to mitigate the influence of old observations. Notably, this baseline delivered good results in our evaluation of CDLEEDS (Paper 5). Unlike the classical expectation baseline, the EWMA is a dynamic and stochastic baseline (assuming that the streaming observations are drawn randomly from the data generating distribution).

4.5 Discussion and Future Work

In this chapter, we discussed problems and approaches related to local explainability in online machine learning. In a corresponding paper (Paper 5), we demonstrated that local attributions can be prone to large variations over time. In this context, we presented CDLEEDS, a novel local change detection framework that allows for a targeted and efficient recalculation of outdated attributions. Based on Paper 3, we also discussed the choice of an appropriate baseline for local attributions in data streams. In the following, we summarize our main findings and identify open issues that should be addressed in the future.

CDLEEDS is one of the first frameworks capable of detecting concept drift at different levels of granularity. In this sense, our approach can serve as an effective alternative to the popular concept drift detection methods presented in Section 2.2.2. Indeed, we demonstrated CDLEEDS as a global concept drift detector on a number of popular data sets (Paper 5). These experiments could be extended in the future.

Our adaptation of the expectation baseline using the exponentially weighted moving average has proven successful in experiments (Paper 5). However, the neutral baseline [92] seems an even more promising alternative for local attributions in data streams. As described in Section 4.1.2, the neutral baseline corresponds to a value on the decision boundary and hence does not contain discriminative information. Per definition, the neutral baseline changes along with the decision boundary. Therefore, given a sensible online learning model, the neutral baseline will always adequately represent the active concept. However, the search-based algorithm presented by Izzo et al. [92] to find a neutral baseline is extremely costly. To be able to use the neutral baseline in data streams, we would need a more efficient algorithm.

In general, there is a lack of explanation methods designed for online learning. In this context, CDLEEDS is a powerful addition to local feature attribution methods. However, our experiments showed that local attributions are extremely sensitive to changes in the online learning model. Hence, if the model does not exhibit locally stationary behavior, local explanations may be of limited practical use. Indeed, explanations at a higher (i.e., global or subgroup) level might be more useful for dynamic data stream applications, as they tend to be more robust to marginal and local updates of the predictor. In this sense, the FIRES model presented in Section 2.3.1 may serve as a powerful global explanation mechanism. FIRES produces stable feature weights in an intuitive combination of importance and uncertainty. Likewise, the Dynamic Model Tree (Section 3.3) can provide feature importance-based explanations at different levels via the parameters of the simple models at each node.

In the future, it may be sensible to develop a new and distinct understanding of explainability in evolving data streams. In particular, we should consider how changes in the model and data distribution can be addressed by an explanation method. That is, online explanation methods should aim to describe both what the model has learned at a particular time step and how the model learns between time steps. The latter form of explainability cannot be achieved with existing attribution methods. In this context, common explanation properties (Section 4.1) might need to be reconsidered and expanded, e.g., with the help of user studies.

Chapter 5

Standardized Evaluation and Comparison of Online Learning Methods

The dynamic nature of data streams makes it difficult to evaluate online learning methods under realistic conditions. In fact, common (offline) evaluation procedures are often not readily applicable because pre-processing, training, and testing in a data stream are highly intertwined. Online learning methods must be updated and evaluated periodically, using a subset of observations at each time step. To obtain reliable results, an evaluation must also consider possible concept drift. Existing experiments in online machine learning research rely heavily on simulations and are not standardized. Indeed, there are major differences in the tools, evaluation criteria and performance measures used. As a consequence, the results cannot usually be reliably compared with each other.

This chapter discusses best practices and standards for the evaluation of online machine learning methods, including online classifiers, online feature selection models and concept drift detection methods. We briefly present previous work that has looked into the evaluation of online learning methods and identify useful resources. We then propose a number of important properties to consider when evaluating online learning methods. Finally, we introduce a new Python framework that enables the evaluation of online learning methods in a more principled, standardized, and transparent way than previous work. This chapter summarizes the contributions of Paper 6.

5.1 Background on Evaluation Practices and Open Resources

Data streams pose considerable challenges that need to be considered in the evaluation of online learning methods. In this section, we present previous work that has addressed these challenges. In addition, we briefly discuss the availability of benchmark data sets and introduce popular libraries. A detailed summary of evaluation techniques and performance measures is provided in Paper 6 (indeed, it is an important contribution of that work). For this reason, we do not discuss specific techniques below.

The evaluation of stream processing techniques and online machine learning methods has been addressed several times in the past [48, 62, 63, 24, 99]. Earlier work is mainly concerned with the evaluation of online predictive models, e.g., in the context of online ensemble learning [100, 67]. Since supervised online learning (i.e., online regression and classification) is very similar to its offline counterpart, previous work has focused on adapting existing offline validation strategies and performance measures for incremental evaluations. For example, Bifet et al. [24] presented a streaming version of the popular k-fold cross validation. Moreover, there are adaptations of performance measures such as accuracy [106] or AUC-ROC (area under the curve of the Receiver Operating Characteristic) [34]. Similarly, for evaluating online feature selection models, one may use adaptations of existing performance measures. These include, for example, the reduction rate [141], which indicates by what fraction the original feature set has been reduced, or the feature set stability [27, 128] (see Section 2.1.2), which quantifies the variability between the selected feature sets. In contrast, concept drift detection has no straightforward counterpart in offline learning. Accordingly, to evaluate concept drift detection methods, we require dedicated evaluation techniques and measures, as proposed in a number of previous papers [69, 108, 64, 23].

Although there are some best practices and de facto standards for assessing online learning methods, there is no unified evaluation framework [100]. In fact, experimental results can differ considerably between publications (see Paper 6). This may be partly because existing recommendations are scattered across a large number of papers and terminology is often used imprecisely and interchangeably. Unfortunately, there is no concise summary of evaluation techniques that covers all the major areas of online learning.

5.1.1 Availability of Benchmark Data Sets

Compared to other machine learning domains, there is a shortage of benchmark data sets for online learning [52, 121, 163, 76, 25]. Souza et al. [154] provided a detailed summary and description of popular benchmark streaming data sets. Therefore, we do not summarize the data sets again in this thesis. Instead, we briefly introduce two valuable data collections with ground truth about concept drift: Insects [154] and TüEyeQ [94, 95]. The Insects data sets were introduced by Souza et al. [154] and include sensor measurements representing the characteristics of flying insect species [154]. The data was obtained in a controlled environment under varying conditions to simulate different types of concept drift. The Insects data sets are tabular data with a multi-class target corresponding to the different insect species. TüEyeQ contains data from multiple participants of an IQ test [95]. In particular, TüEyeQ provides eye movement data together with task-specific details and socio-demographic information about each participant (tabular data). In general, TüEyeQ may be used for various predictive tasks. For example, we can train a binary online classifier on the socio-demographic and task-specific data to predict whether a given test item was passed or failed. Since the IQ test included different task blocks and the difficulty of the tasks increased over time, the observations in TüEyeQ are subject to concept drift.

Apart from Insects and TüEyeQ, real-world streaming data sets do not usually provide information on the extent to which they contain concept drift. However, in experiments we often need a ground truth about concept drift, e.g., when evaluating active concept drift detection methods (Section 2.2.2). In these cases, we can use data manipulation techniques to introduce artificial concept drift. For example, Sethi and Kantardzic [150] proposed a simple approach to simulate sudden concept drifts by randomly permuting the entries of input features after a specified point in time. Specifically, they first rank the input features according to the Information Gain. In a second step, they randomly permute a fraction of the features at the bottom or top of the ranking to achieve different forms of concept drift. However, such random permutations are often not meaningful [176]. As an alternative, Žliobaitė [176] proposed three permutation schemes that preserve the natural dynamics of the original data. The time permutation scheme simulates sudden concept drifts (Section 1.5) by shifting entire blocks of observations. The speed permutation moves sets of observations to the end of the data stream to simulate the reoccurence of data concepts. Finally, the shape permutation simulates gradual concept drifts by exchanging observations that are close in time.

As an alternative to real-world data sets, we often also find synthetically generated data streams [123, 2, 88, 155]. Indeed, most online learning libraries (Section 5.1.2) provide a number of different and well-documented data generators. Synthetic data streams can vary greatly depending on the configuration of the generator. For this reason, experimental results based on synthetic data can be misleading and should be treated with caution.

5.1.2 Popular Libraries

There is a range of open source online learning tools and models in common programming languages such as Java or Python. Summaries of existing open source software for online learning and stream processing can be found in recent surveys [8, 90]. In the following, we briefly introduce some of the most popular libraries.

One of the best known libraries for online learning is the Massive Online Analysis (MOA) [21, 23]. MOA is implemented in Java and is related to the Waikato Environment for Knowledge Analysis (WEKA). MOA provides various stream processing techniques, data generators and online learning methods. It also has extensive documentation and a graphical user interface.

Similar to other areas of machine learning, Python has quickly become a popular choice for online learning. The scikit-multiflow [118, 117] and creme [73] libraries provide the most comprehensive set of data generators, online predictive models and concept drift detection methods implemented in Python. Both packages integrate easily with standard machine learning libraries such as scikit-learn [133]. Recently, scikit-multiflow and creme have been merged into a novel project called river [120]. In addition to these extensive libraries, there are several smaller repositories to be found on Github, e.g., the tornado framework for online classification and concept drift detection [136].

However, although the above-mentioned libraries offer a wealth of functionality, the evaluation of online learning methods often requires a considerable additional implementation effort. Indeed, data stream experiments usually require many custom design decisions, e.g., regarding the meaningful combination of online feature selection, drift detection and online classification models, or the visualisation of the results.

5.2 Consolidating Evaluation Standards

Designing meaningful experiments to evaluate online machine learning methods can be difficult. Indeed, there is no unified standard for data preparation, testing and training strategy, or reported performance measures. Although some evaluation techniques are widely used, it can be difficult to survey and compare the different approaches proposed in the literature – especially for novices in the field.

To enable more informed decisions, we introduce a concise and up-to-date summary of evaluation practices for online learning (Paper 6). Our work helps to clarify important terms, compare existing techniques and thus achieve a more comparable and standardized evaluation of online learning methods in practice. Below, we briefly present and discuss general evaluation strategies as well as important properties for online feature selection, concept drift detection (Chapter 2) and online classification (Chapter 3). For more details, please refer to Paper 6.

Online learning methods are incrementally tested and trained as new streaming observations emerge. In this context, there are three commonly used evaluation strategies. Each of the three evaluation strategies has its strengths and weaknesses, which we discuss more thoroughly in Paper 6. In the *periodic holdout evaluation*, we test the online learning method at regular intervals using a sample of test observations. This test sample may be either static or updated over time, e.g., using a reservoir sampling approach (see Section 2.2.1). In contrast, in a *prequential evaluation*, the model is tested at each time step. More precisely, each incoming observation is first used to test and (once the label is available) to train the model. Finally, the *distributed k-fold evaluation* strategy is an adaptation of the popular k-fold cross-validation scheme [24]. In a distributed k-fold evaluation, we train k instances of the model in parallel. Each streaming observation is assigned to one or multiple model instances for testing or training according to one of three different schemes [24]. The results of the individual model instances can be aggregated to obtain a more robust overall evaluation.

Apart from the general evaluation strategy, we need to decide which properties of the online learning method we want to assess. Indeed, there are a number of important properties that need to be considered when developing and evaluating online learning methods. Most of these properties have already been discussed in earlier sections of this thesis. In Paper 6, we identify the following general properties:

- **Predictive Performance and Generalization:** Online machine learning methods are used for automated decision making. Accordingly, an online learning method should provide high predictive quality for previously unseen data at each time step.
- **Computational Efficiency:** Online applications often deal with limited hardware capacities. Ideally, online learning methods should be able to provide real-time predictions while having a small memory footprint.
- **Algorithmic Stability:** Online learning methods should provide stable predictions in the presence of noisy inputs.
- **Concept Drift Adaptability:** Online learning methods use different strategies to deal with concept drift, e.g., online predictive models often employ active concept drift detection (Section 2.2). Regardless of the strategy used, online learning methods should be able to adjust quickly to concept drift without drastically degrading performance.
- **Interpretability:** High-stakes decisions and new laws for AI regulation require machine learning methods to be interpretable. In the absence of an objective measure, we generally tie interpretability to the complexity of a method (Section 3.2).

While these properties apply to most online learning methods, there are additional properties related to concept drift detection and online feature selection. Specifically, we introduce the following important properties for concept drift detection (Paper 6):

- **Detection Truthfulness:** Concept drift detection methods should be able to correctly detect concept drifts while avoiding a high number of false alarms.
- **Detection Timeliness:** In order to allow fast updates of the predictive model and avoid long-term performance deterioration, concept drift detection methods should be able to detect concept drifts with a short delay.

Finally, we identify two more properties for online feature selection (Paper 6):

- **Feature Set Stability:** To be reliably used in practice, the feature sets obtained by an online feature selection model should be stable as long as the data generating distribution does not change (Section 2.1.2).
- **Feature Selectivity:** An online feature selection method should be able to select the important features for prediction at each time step while discarding irrelevant features. If the number of selected features is automatically determined, then the size of the selected feature set compared to the original number of features is an interesting property.

A more detailed discussion of each property can be found in Paper 6, where we also propose meaningful performance measures. Based on the proposed properties, it is much easier to design reliable and comparable experiments for online machine learning.

```python
from skmultiflow.trees import HoeffdingTreeClassifier
from sklearn.metrics import zero_one_loss

from float.data import DataLoader
from float.prediction.evaluation import PredictionEvaluator
from float.prediction.evaluation.measures import noise_variability
from float.pipeline import PrequentialPipeline
from float.prediction.skmultiflow import SkmultiflowClassifier

# Load a data set from main memory with the DataLoader module.
# Alternatively, we can provide a sciki-multiflow FileStream...
#   ...object via the 'stream' attribute.
data_loader = DataLoader(path='./datasets/spambase.csv',
                         target_col=-1)

# Set up an online classifier. Note that we need a wrapper to...
#   ...use scikit-multiflow functionality.
classifier = SkmultiflowClassifier(model=HoeffdingTreeClassifier(),
                                   classes=data_loader.stream.target_values)

# Set up an evaluator object for the classifier:
# Specifically, we want to measure the zero_one_loss and the...
#   ...noise_variability as an indication of stability.
# The arguments of the performance measures (measure_func)...
#   ...can be directly added to the Evaluator object...
#   ...constructor, e.g. we may specify the number of...
#   ...samples (n_samples) and the reference_measure used...
#   ...to compute the noise_variability.
evaluator = PredictionEvaluator(measure_funcs=[zero_one_loss,
                                               noise_variability],
                                n_samples=15,
                                reference_measure=zero_one_loss)

# Set up a pipeline for a prequential evaluation of the classifier.
pipeline = PrequentialPipeline(data_loader=data_loader,
                               predictor=classifier,
                               prediction_evaluator=evaluator,
                               n_max=data_loader.stream.n_samples,
                               batch_size=25)

# Run the experiment.
pipeline.run()
```

Figure 5.1: **Conducting Online Learning Experiments With the Float Python-Framework.** Here, we show the source code for a simple online learning experiment using float, which can also be found in slightly adjusted form on float's README page on Github. Further experiments and detailed information on the modules and hyperparameters can be found in the documentation for our framework.

5.3 Float - A Python-Package for Data Stream Evaluations

Although there is a variety of useful online learning libraries (Section 5.1.2), there is no Python package dedicated to the evaluation of online learning methods. With the *Frictionless Online Model Analysis and Testing (float)*, we introduce a novel Python framework that fills this gap (Paper 6). Float provides different modules (Python classes) for concept drift detection, online feature selection and online classification. Its modular structure makes it possible to combine float with the popular online learning libraries scikit-multiflow [118] and river [120]. Yet, by providing abstract base classes for all core modules, float also enables easy integration of user-defined methods.

To design an experiment, we can flexibly combine modules via a pipeline (Python class). Float offers three standardized pipelines that correspond to the evaluation strategies mentioned in Section 5.2. The pipeline automatizes the entire evaluation process – from loading the data to computing different performance measures. In this way, float allows the flexible combination of different online learning methods to perform customized experiments without the user having to deal with low-level implementation details. A basic example on how to use a float pipeline is shown in Figure 5.1.

Float is introduced in Paper 6, along with the evaluation properties outlined in Section 5.2. Although float is not intended as a library of online learning models, it contains several implementations of recent models that are not yet integrated into one of the major libraries. These include FIRES (Section 2.3.1) and ERICS (Section 2.3.2), for example. In addition, float contains a visualization module with different types of plots that can be easily created from the results of a pipeline run.

The proposed framework is thus a useful extension of existing online learning libraries and can serve as a basis for more standardized and reliable evaluations in data streams. Float is open sourced under the MIT license and can be accessed on Github[1] or PyPi[2].

5.4 Discussion and Future Work

In this chapter, we discussed pressing and recurring issues in the evaluation of online learning methods. As one of the first works, we summarized and extended common evaluation strategies by translating them into a set of important properties (Paper 6). We also presented a novel framework that enables the evaluation of online learning methods in a few lines of Python code. In the following, we briefly outline future work.

With Paper 6 and float, we created the basis for a standardized evaluation of online learning methods. In a next step, the proposed tools could be used to compare common online learning methods in large-scale experiments. These results could in turn serve as

[1]https://github.com/haugjo/float
[2]https://pypi.org/project/float-evaluation/

a benchmark for the development of new online learning methods.

Float offers a variety of modules and functions that could be expanded in the future. In particular, float currently lacks a module for generating local attributions based on CDLEEDS (Section 4.3). In addition, one may implement other performance measures and vizualization types. Likewise, float could be extended to include common hypothesis tests to automatically identify significant effects based on a distributed *k*-fold evaluation [24] (see Section 5.2).

Although we have addressed a number of issues in the evaluation of online learning methods, there are still open questions that require attention. In particular, we currently lack performance measures that adequately account for concept drift. In online feature selection, for example, we would usually expect variability in the selected feature set after a concept drift, as the online learning model needs to adjust to the new data distribution. During this time, low feature set stability (Section 2.1.2) does not necessarily indicate unstable behavior. In fact, some degree of variation may even be desirable, as it is an indicator that the model is learning the new concept. Finally, we require more real-world data streams with known concept drift to avoid using synthetic data in experiments. As an alternative, it might also be sensible to explore generative models or data augmentation techniques. In this way, we might expand the existing real-world data sets and obtain more realistic and reliable evaluations.

Chapter 6

Conclusion and Outlook

Reliable machine learning for evolving data streams requires well-designed solutions. Indeed, to cope with the dynamic and restrictive nature of online applications, online learning methods have to meet high requirements, e.g., in terms of predictive performance, resource and runtime efficiency, and stability. While a number of methods have emerged as state of the art over the years, they often have known limitations or do not address current and pressing issues, such as the interpretability of models and predictions.

In this work, we identified four issues in online learning that are not adequately addressed in existing work. We contributed to these issues in six publications. Together, our contributions cover large parts of the online learning process: In Chapter 2, we introduced a novel incremental framework that allows us to estimate the distributions of the parameters of differentiable online learning models. Based on this framework, we proposed FIRES and ERICS, novel approaches for online feature selection and concept drift detection, respectively. By explicitly accounting for uncertainty in the model, both approaches have been shown to improve important performance indicators, such as feature set stability or drift detection delay. In Chapter 3, we introduced the Dynamic Model Tree (DMT) classification framework. Inspired by the limitations of ever-growing incremental decision tree-based models on data streams, the DMT adheres to sensible properties that allow it to obtain powerful predictions with reduced complexity and higher interpretability. In a similar context, we studied the behavior of local attribution methods for black-box explainability in evolving data streams (Chapter 4). We showed that the validity of local attributions can often only be maintained over a short period of time, so that we have to recompute the attributions several times during the online training. In this context, we presented CDLEEDS, a local change and concept drift detection framework that enables more targeted recalculations and thus more feasible attribution-based explainability in data streams. Finally, in Chapter 5, we examined the state of experiments in the online learning environment, uncovering unaddressed issues and non-standardized evaluation procedures. In this context, we summarized important online learning properties based on previous work. In addition, we released a new Python package called float that enables standardized evaluation of online learning methods, including online feature selection, concept drift detection and online classification approaches.

Overall, this thesis and related papers made important methodological and practical contributions. We presented theoretical and empirical findings, proposed new techniques and frameworks, and published extensive open resources. Accordingly, our work is an important step towards more reliable machine learning in evolving data streams.

We strongly believe that online machine learning will continue to be of great interest for many practical applications. In particular, online learning techniques are valuable wherever an application is subject to temporal change but long-term human supervision or repeated retraining is not feasible. However, existing online learning techniques often lack the versatility to deal with highly challenging use cases and data types such as images or language. As a result, real-world and dynamic processes are increasingly being tackled with more complex solutions – especially from the field of deep learning. Although these approaches are very powerful, they usually have inherent limitations in terms of interpretability and resource consumption. In view of recent efforts, e.g., related to fairness, transparency, or "Green IT", online learning, and in particular the techniques presented in this thesis, can offer viable and powerful alternatives.

Bibliography

[1] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. doi: 10.5555/3327546.3327621.

[2] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, dec 1993. doi: 10.1109/69.250074.

[3] Cesare Alippi and Manuel Roveri. Just-in-time adaptive classifiers—part i: Detecting nonstationary changes. *IEEE Transactions on Neural Networks*, 19(7): 1145–1153, jul 2008. doi: 10.1109/tnn.2008.2000082.

[4] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. *ProPublica*, 23(2016):139–159, 2016.

[5] Stephen H. Bach and Marcus A. Maloof. Paired learners for concept drift. In *Eighth IEEE International Conference on Data Mining*, pages 23–32, 2008. doi: 10.1109/ICDM.2008.119.

[6] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and Rafael Morales-Bueno. Early drift detection method. In *Fourth International Workshop on Knowledge Discovery from Data Streams*, volume 6, pages 77–86, 2006.

[7] Maroua Bahri, Silviu Maniu, and Albert Bifet. A sketch-based naive bayes algorithms for evolving data streams. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, dec 2018. doi: 10.1109/bigdata.2018.8622178.

[8] Maroua Bahri, Albert Bifet, João Gama, Heitor Murilo Gomes, and Silviu Maniu. Data stream analysis: Foundations, major tasks and tools. *WIREs Data Mining and Knowledge Discovery*, 11(3), mar 2021. doi: 10.1002/widm.1405.

[9] Jean Paul Barddal and Fabrício Enembreck. Learning regularized hoeffding trees from data streams. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. ACM, apr 2019. doi: 10.1145/3297280.3297334.

[10] Jean Paul Barddal and Fabrício Enembreck. Regularized and incremental decision trees for data streams. *Annals of Telecommunications*, 75(9-10):493–503, jul 2020. doi: 10.1007/s12243-020-00782-3.

[11] Jean Paul Barddal, Lucas Loezer, Fabrício Enembreck, and Riccardo Lanzuolo. Lessons learned from data stream classification applied to credit scoring. *Expert Systems with Applications*, 162:113899, dec 2020. doi: 10.1016/j.eswa.2020.113899.

[12] Roberto S.M. Barros, Danilo R.L. Cabral, Paulo M. Gonçalves, and Silas G.T.C. Santos. RDDM: Reactive drift detection method. *Expert Systems with Applications*, 90:344–355, dec 2017. doi: 10.1016/j.eswa.2017.08.023.

[13] Mariam Barry, Albert Bifet, Raja Chiky, Jacob Montiel, and Vinh-Thuy Tran. Challenges of machine learning for data streams in the banking industry. In *Big Data Analytics*, pages 106–118. Springer International Publishing, 2021. doi: 10.1007/978-3-030-93620-4_9.

[14] Richard Bellman. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957. doi: https://doi.org/10.1515/9781400835386.

[15] Adrien Bibal and Benoît Frénay. Interpretability of machine learning models and representations: an introduction. In *24th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2016.

[16] Albert Bifet and João Gama. IoT data stream analytics. *Annals of Telecommunications*, 75(9-10):491–492, sep 2020. doi: 10.1007/s12243-020-00811-1.

[17] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, apr 2007. doi: 10.1137/1.9781611972771.42.

[18] Albert Bifet and Ricard Gavaldà. Adaptive learning from evolving data streams. In *Advances in Intelligent Data Analysis VIII*, pages 249–260. Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-642-03915-7_22.

[19] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *Machine Learning and Knowledge Discovery in Databases*, pages 135–150. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-15880-3_15.

[20] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Eibe Frank. Fast perceptron decision tree learning from evolving data streams. In *Advances in Knowledge Discovery and Data Mining*, pages 299–310. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-13672-6_30.

[21] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. Moa: Massive online analysis, a framework for stream classification and clustering. In *Proceedings of the First Workshop on Applications of Pattern Analysis*, volume 11 of *Proceedings of Machine Learning Research*, pages 44–50, Cumberland Lodge, Windsor, UK, 01–03 Sep 2010. PMLR.

[22] Albert Bifet, Bernhard Pfahringer, Jesse Read, and Geoff Holmes. Efficient data stream classification via probabilistic adaptive windows. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC '13*. ACM Press, 2013. doi: 10.1145/2480362.2480516.

[23] Albert Bifet, Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Indrė Žliobaitė. CD-MOA: Change detection framework for massive online analysis. In *Advances in Intelligent Data Analysis XII*, pages 92–103. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-41398-8_9.

[24] Albert Bifet, Gianmarco de Francisci Morales, Jesse Read, Geoff Holmes, and Bernhard Pfahringer. Efficient online evaluation of big data stream classifiers. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, aug 2015. ACM. doi: 10.1145/2783258.2783372.

[25] Jock A. Blackard and Denis J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151, dec 1999. doi: 10.1016/s0168-1699(99)00046-0.

[26] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos. Recent advances and emerging challenges of feature selection in the context of big data. *Knowledge-Based Systems*, 86:33–45, sep 2015. doi: 10.1016/j.knosys.2015.05.014.

[27] Andrea Bommert and Michel Lang. Stabm: Stability measures for feature selection. *Journal of Open Source Software*, 6(59):3010, mar 2021. doi: 10.21105/joss.03010.

[28] Vadim Borisov, Johannes Haug, and Gjergji Kasneci. Cancelout: A layer for feature selection in deep neural networks. In *International Conference on Artificial Neural Networks*, pages 72–83. Springer, 2019. doi: https://doi.org/10.1007/978-3-030-30484-3_6.

[29] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *arXiv*, 2021. doi: https://doi.org/10.48550/arXiv.2110.01889.

[30] Zoran Bosnić, Jaka Demšar, Grega Kešpret, Pedro Pereira Rodrigues, João Gama, and Igor Kononenko. Enhancing data stream predictions with reliability estimators and explanation. *Engineering Applications of Artificial Intelligence*, 34:178–192, sep 2014. doi: 10.1016/j.engappai.2014.06.001.

[31] Olivier Bousquet and André Elisseeff. Stability and generalization. *The Journal of Machine Learning Research*, 2:499–526, 2002. doi: 10.1162/153244302760200704.

[32] Alejandra Bringas Colmenarejo, Luca Nannini, Alisa Rieger, Kristen M. Scott, Xuan Zhao, Gourab K Patro, Gjergji Kasneci, and Katharina Kinder-Kurlanda. Fairness in agreement with european values: An interdisciplinary perspective on ai regulation. In *Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '22, page 107–118, New York, NY, USA, 2022. Association for Computing Machinery. doi: 10.1145/3514094.3534158.

[33] Klaus Broelemann and Gjergji Kasneci. A gradient-based split criterion for highly accurate and transparent model trees. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, aug 2019. doi: 10.24963/ijcai.2019/281.

[34] Dariusz Brzezinski and Jerzy Stefanowski. Prequential AUC: Properties of the area under the ROC curve for data streams with concept drift. *Knowledge and Information Systems*, 52(2):531–562, jan 2017. doi: 10.1007/s10115-017-1022-8.

[35] Danilo Rafael De Lima Cabral and Roberto Souto Maior De Barros. Concept drift detection based on fisher's exact test. *Information Sciences*, 442-443:220–234, may 2018. doi: 10.1016/j.ins.2018.02.054.

[36] Diogo V. Carvalho, Eduardo M. Pereira, and Jaime S. Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, jul 2019. doi: 10.3390/electronics8080832.

[37] Vitor R. Carvalho and William W. Cohen. Single-pass online learning. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, 2006. doi: 10.1145/1150402.1150466.

[38] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, jan 2014. doi: 10.1016/j.compeleceng.2013.11.024.

[39] Chaofan Chen, Oscar Li, Chaofan Tao, Alina Jade Barnett, Jonathan Su, and Cynthia Rudin. This looks like that: Deep learning for interpretable image recognition. *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, June 2018. doi: 10.5555/3454287.3455088.

[40] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016. doi: 10.1145/2939672.2939785.

[41] Andrea Dal Pozzolo, Olivier Caelen, Yann-Aël Le Borgne, Serge Waterschoot, and Gianluca Bontempi. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Systems with Applications*, 41(10):4915–4928, 2014. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2014.02.026.

[42] Tamraparni Dasu, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi. An information-theoretic approach to detecting changes in multi-dimensional data streams. *Interfaces*, 01 2006.

[43] Jaka Demšar and Zoran Bosnić. Detecting concept drift in data streams using model explanation. *Expert Systems with Applications*, 92:546–559, feb 2018. doi: 10.1016/j.eswa.2017.10.003.

[44] Yanlei Diao, Boduo Li, Anna Liu, Liping Peng, Charles Sutton, Thanh Tran, and Michael Zink. Capturing data uncertainty in high-volume stream processing. *4th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2009. doi: https://doi.org/10.48550/arXiv.0909.1777.

[45] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, nov 2015. doi: 10.1109/mci.2015.2471196.

[46] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2000. doi: https://doi.org/10.1145/347090.347107.

[47] Pedro Domingos and Geoff Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12(4):945–949, dec 2003. doi: 10.1198/1061860032544.

[48] Pedro M. Domingos and Geoff Hulten. Catching up with the data: Research issues in mining data streams. In *Workshop on Research Issues on Data Mining and Knowledge Discovery*, 2001.

[49] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proceedings of the 25th International Conference on Machine learning*. ACM Press, 2008. doi: 10.1145/1390156.1390190.

[50] Lei Du, Qinbao Song, and Xiaolin Jia. Detecting concept drift: An information entropy based method using an adaptive sliding window. *Intelligent Data Analysis*, 18(3):337–364, apr 2014. doi: 10.3233/ida-140645.

[51] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77, dec 2019. doi: 10.1145/3359786.

[52] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[53] Piotr Duda, Maciej Jaworski, Andrzej Cader, and Lipo Wang. On training deep neural networks using a streaming approach. *Journal of Artificial Intelligence and Soft Computing Research*, 10(1):15–26, dec 2019. doi: 10.2478/jaiscr-2020-0002.

[54] Ruth C. Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, Oct 2017. doi: https://doi.org/10.1109/ICCV.2017.371.

[55] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, aug 1997. doi: 10.1006/jcss.1997.1504.

[56] Isvani Frias-Blanco, Jose del Campo-Avila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustin Ortiz-Diaz, and Yaile Caballero-Mota. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823, mar 2015. doi: 10.1109/tkde.2014.2345382.

[57] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *Proceedings of the First International Workshop on Comprehensibility and Explanation in AI and ML*, November 2017. doi: https://doi.org/10.48550/arXiv.1711.09784.

[58] João Gama. A survey on learning from data streams: Current and future trends. *Progress in Artificial Intelligence*, 1(1):45–55, jan 2012. doi: 10.1007/s13748-011-0002-6.

[59] João Gama and Gladys Castillo. Learning with local drift detection. In *Advanced Data Mining and Applications*, pages 42–55. Springer Berlin Heidelberg, 2006. doi: 10.1007/11811305_4.

[60] João Gama, Ricardo Rocha, and Pedro Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, 2003. doi: 10.1145/956750.956813.

[61] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence – SBIA 2004*, pages 286–295. Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-540-28645-5_29.

[62] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM Press, 2009. doi: 10.1145/1557019.1557060.

[63] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, oct 2012. doi: 10.1007/s10994-012-5320-9.

[64] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):1–37, apr 2014. doi: 10.1145/2523813.

[65] Adel Ghazikhani, Reza Monsefi, and Hadi Sadoghi Yazdi. Ensemble of online neural networks for non-stationary and imbalanced data streams. *Neurocomputing*, 122:535–544, dec 2013. doi: 10.1016/j.neucom.2013.05.003.

[66] Heitor M. Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9-10):1469–1495, jun 2017. doi: 10.1007/s10994-017-5642-8.

[67] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. A survey on ensemble learning for data stream classification. *ACM Computing Surveys*, 50(2):1–36, jun 2017. doi: 10.1145/3054925.

[68] Heitor Murilo Gomes, Jesse Read, and Albert Bifet. Streaming random patches for evolving data stream classification. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, nov 2019. doi: 10.1109/icdm.2019.00034.

[69] Paulo M. Gonçalves, Silas G.T. de Carvalho Santos, Roberto S.M. Barros, and Davi C.L. Vieira. A comparative study on concept drift detectors. *Expert Systems with Applications*, 41(18):8144–8156, dec 2014. doi: 10.1016/j.eswa.2014.07.019.

[70] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, 61(7):56–66, jun 2018. doi: 10.1145/3134599.

[71] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5):1–42, jan 2019. doi: 10.1145/3236009.

[72] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003. doi: 10.5555/944919.944968.

[73] Max Halford, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, and Adil Zouitine. Creme, a Python library for online machine learning, 2019. URL `https://github.com/MaxHalford/creme`.

[74] Ronan Hamon, Henrik Junklewitz, and Ignacio Sanchez. Robustness and explainability of artificial intelligence. *Publications Office of the European Union*, 2020. doi: 10.2760/57493.

[75] Maayan Harel, Shie Mannor, Ran El-Yaniv, and Koby Crammer. Concept drift detection through resampling. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1009–1017. PMLR, 2014. doi: 10.5555/3044805.3045005.

[76] M. Harries, University of New South Wales. School of Computer Science, and Engineering. *Splice-2 Comparative Evaluation: Electricity Pricing*. PANDORA electronic collection. University of New South Wales, School of Computer Science and Engineering, 1999.

[77] Johannes Haug and Gjergji Kasneci. Learning parameter distributions to detect concept drift in data streams. In *25th International Conference on Pattern Recognition (ICPR)*. IEEE, jan 2021. doi: 10.1109/icpr48806.2021.9412499.

[78] Johannes Haug, Martin Pawelczyk, Klaus Broelemann, and Gjergji Kasneci. Leveraging model inherent variable importance for stable online feature selection. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, New York, NY, USA, jul 2020. ACM. doi: 10.1145/3394486.3403200.

[79] Johannes Haug, Stefan Zürn, Peter El-Jiz, and Gjergji Kasneci. On baselines for local feature attributions. *AAAI 2021 Workshop on Explainable Agency in Artificial Intelligence*, 2021. doi: https://doi.org/10.48550/arXiv.2101.00905.

[80] Johannes Haug, Alexander Braun, Stefan Zürn, and Gjergji Kasneci. Change detection for local explainability in evolving data streams. *31st ACM International Conference on Information and Knowledge Management*, 2022. doi: 10.48550/arXiv.2209.02764.

[81] Johannes Haug, Klaus Broelemann, and Gjergji Kasneci. Dynamic model tree for interpretable data stream learning. *IEEE 38th International Conference on Data Engineering*, 2022. doi: 10.1109/ICDE53745.2022.00237.

[82] Johannes Haug, Effi Tramountani, and Gjergji Kasneci. Standardized evaluation of machine learning methods for evolving data streams. *arXiv*, 2022. doi: https://doi.org/10.48550/arXiv.2204.13625.

[83] Geoffrey Holmes, Richard Kirkby, and Bernhard Pfahringer. Stress-testing hoeffding trees. In *Knowledge Discovery in Databases: PKDD 2005*, pages 495–502. Springer Berlin Heidelberg, 2005. doi: 10.1007/11564126_50.

[84] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates, Inc., 2019. doi: 10.5555/3454287.3455160.

[85] Bo-Jian Hou, Lijun Zhang, and Zhi-Hua Zhou. Learning with feature evolvable streams. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. doi: 10.5555/3294771.3294906.

[86] Chang How Tan, Vincent CS Lee, and Mahsa Salehi. Online semi-supervised concept drift detection with density estimation. *arXiv*, September 2019. doi: https://doi.org/10.48550/arXiv.1909.11251.

[87] Hao Huang, Shinjae Yoo, and Shiva Prasad Kasiviswanathan. Unsupervised feature selection on data streams. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*. ACM, oct 2015. doi: 10.1145/2806416.2806521.

[88] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, 2001. doi: https://doi.org/10.1145/502512.502529.

[89] Elena Ikonomovska, João Gama, and Sašo Džeroski. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1):128–168, oct 2010. doi: 10.1007/s10618-010-0201-y.

[90] Wissem Inoubli, Sabeur Aridhi, Haithem Mezni, Mondher Maddouri, and Engelbert Mephu Nguifo. A comparative study on streaming frameworks for big data. In *44th International Conference on Very Large Data Bases : Workshop LADaS - Latin American Data Science*, pages 1–8, Rio de Janeiro, Brazil, August 2018.

[91] Ozan Irsoy, Olcay Taner Yıldız, and Ethem Alpaydın. Soft decision trees. In *Proceedings of the 21st International Conference on Pattern Recognition*, pages 1819–1822, 2012.

[92] C Izzo, A Lipani, R Okhrati, and F Medda. A baseline for shapley values in mlps: from missingness to neutrality. In *Proceedings of the European Symposium*

on Artificial Neural Networks, Computational Intelligence and Machine Learning, pages 605–610. i6doc publication, 2021. doi: https://doi.org/10.14428/esann/2021.ES2021-18.

[93] A. Kalousis, J. Prados, and M. Hilario. Stability of feature selection algorithms. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE, 2005. doi: 10.1109/icdm.2005.135.

[94] Enkelejda Kasneci, Gjergji Kasneci, Tobias Appel, Johannes Haug, Franz Wortha, Maike Tibus, Ulrich Trautwein, and Peter Gerjets. TüEyeQ, a rich IQ test performance data set with eye movement, educational and socio-demographic information. *Harvard Dataverse*, 2020. doi: 10.7910/DVN/JGOCKI.

[95] Enkelejda Kasneci, Gjergji Kasneci, Tobias Appel, Johannes Haug, Franz Wortha, Maike Tibus, Ulrich Trautwein, and Peter Gerjets. Tüeyeq, a rich iq test performance data set with eye movement, educational and socio-demographic information. *Scientific Data*, 8(1):1–14, 2021. doi: https://doi.org/10.1038/s41597-021-00938-3.

[96] Gjergji Kasneci and Thomas Gottron. LICON: A linear weighting scheme for the contribution of input variables in deep artificial neural networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, oct 2016. doi: 10.1145/2983323.2983746.

[97] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. On the utility of incremental feature selection for the classification of textual data streams. In *Panhellenic Conference on Informatics*, pages 338–348. Springer, 2005. doi: https://doi.org/10.1007/11573036_32.

[98] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, page 180–191. VLDB Endowment, 2004. ISBN 0120884690. doi: 10.5555/1316689.1316707.

[99] Taiwo Kolajo, Olawande Daramola, and Ayodele Adebiyi. Big data stream analysis: A systematic literature review. *Journal of Big Data*, 6(1), jun 2019. doi: 10.1186/s40537-019-0210-7.

[100] Bartosz Krawczyk, Leandro L. Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156, sep 2017. doi: 10.1016/j.inffus.2017.02.004.

[101] Ludmila I. Kuncheva. Classifier ensembles for changing environments. In *Multiple Classifier Systems*, pages 1–15. Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-540-25966-4_1.

[102] Haiguang Li, Xindong Wu, Zhao Li, and Wei Ding. Online group feature selection from feature streams. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013. doi: 10.5555/2891460.2891700.

[103] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Computing Surveys*, 50(6):1–45, nov 2018. doi: 10.1145/3136625.

[104] Yanbin Liu, Yan Yan, Ling Chen, Yahong Han, and Yi Yang. Adaptive sparse confidence-weighted learning for online feature selection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4408–4415, jul 2019. doi: 10.1609/aaai.v33i01.33014408.

[105] Viktor Losing, Barbara Hammer, and Heiko Wersing. KNN classifier with self adjusting memory for heterogeneous concept drift. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, dec 2016. doi: 10.1109/icdm.2016.0040.

[106] Viktor Losing, Barbara Hammer, and Heiko Wersing. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 275:1261–1274, jan 2018. doi: 10.1016/j.neucom.2017.06.084.

[107] Viktor Losing, Heiko Wersing, and Barbara Hammer. Enhancing very fast decision trees with local split-time predictions. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, nov 2018. doi: 10.1109/icdm.2018.00044.

[108] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering 31, no. 12 (2018): 2346-2363*, April 2018. doi: 10.1109/TKDE.2018.2876857.

[109] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Curran Associates, Inc., 2017. doi: 10.5555/3295222.3295230.

[110] Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. Consistent individualized feature attribution for tree ensembles. *arXiv*, February 2018. doi: https://doi.org/10.48550/arXiv.1802.03888.

[111] Chaitanya Manapragada, Geoffrey I. Webb, and Mahsa Salehi. Extremely fast decision tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, jul 2018. doi: 10.1145/3219819.3220005.

[112] Chaitanya Manapragada, Geoffrey I Webb, Mahsa Salehi, and Albert Bifet. Emergent and unspecified behaviors in streaming decision trees. *arXiv*, 2020. doi: https://doi.org/10.48550/arXiv.2010.08199.

[113] Chaitanya Manapragada, Heitor M Gomes, Mahsa Salehi, Albert Bifet, and Geoffrey I Webb. An eager splitting strategy for online decision trees in ensembles. *Data Mining and Knowledge Discovery*, pages 1–54, 2022. doi: https://doi.org/10.1007/s10618-021-00816-x.

[114] Mohammad M. Masud, Qing Chen, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. Classification and novel class detection of data streams in a dynamic feature space. In *Machine Learning and Knowledge Discovery in Databases*, pages 337–352. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-15883-4_22.

[115] Pawel Matuszyk, Georg Krempl, and Myra Spiliopoulou. Correcting the usage of the hoeffding inequality in stream mining. In *Advances in Intelligent Data Analysis XII*, pages 298–309. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-41398-8_26.

[116] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, feb 2019. doi: 10.1016/j.artint.2018.07.007.

[117] Jacob Montiel. Learning from evolving data streams. *Proceedings of the 19th Python in Science Conference*, 2020. doi: 10.25080/majora-342d178e-025.

[118] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19 (72):1–5, 2018. doi: https://doi.org/10.5555/3291125.3309634.

[119] Jacob Montiel, Rory Mitchell, Eibe Frank, Bernhard Pfahringer, Talel Abdessalem, and Albert Bifet. Adaptive XGBoost for evolving data streams. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2020. doi: 10.1109/ijcnn48605.2020.9207555.

[120] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. River: Machine learning for streaming data in python. *Journal of Machine Learning Research*, 22(110):1–8, 2021. doi: https://doi.org/10.48550/arXiv.2012.04740.

[121] Sergio Moro, Raul Laureano, and Paulo Cortez. Using data mining for bank direct marketing: An application of the crisp-dm methodology. *Proceedings of the European Simulation and Modelling Conference*, 2011.

[122] Michal Moshkovitz, Yao-Yuan Yang, and Kamalika Chaudhuri. Connecting interpretability and robustness in decision trees through separation. In *International Conference on Machine Learning*, pages 7839–7849. PMLR, 2021. doi: https://doi.org/10.48550/arXiv.2102.07048.

[123] Anand Narasimhamurthy and Ludmila I. Kuncheva. A framework for generating data to simulate changing environments. In *Proceedings of the 25th IASTED International Multi-Conference: Artificial Intelligence and Applications*, AIAP'07, page 384–389, USA, 2007. ACTA Press. doi: 10.5555/1295303.1295369.

[124] J. A. Nelder and R. W. M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370, 1972. doi: 10.2307/2344614.

[125] Hai-Long Nguyen, Yew-Kwong Woon, Wee-Keong Ng, and Li Wan. Heterogeneous ensemble for feature drifts in data streams. In *Advances in Knowledge Discovery and Data Mining*, pages 1–12. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-30220-6_1.

[126] Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. A survey on data stream clustering and classification. *Knowledge and Information Systems*, 45(3): 535–569, dec 2014. doi: 10.1007/s10115-014-0808-1.

[127] Kyosuke Nishida and Koichiro Yamauchi. Detecting concept drift using statistical testing. In *Discovery Science*, pages 264–269. Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-75488-6_27.

[128] Sarah Nogueira, Konstantinos Sechidis, and Gavin Brown. On the stability of feature selection algorithms. *Journal of Machine Learning Research*, 18(1):6345–6398, 2018. doi: https://dl.acm.org/doi/10.5555/3122009.3242031.

[129] Eirini Ntoutsi, Pavlos Fafalios, Ujwal Gadiraju, Vasileios Iosifidis, Wolfgang Nejdl, Maria-Esther Vidal, Salvatore Ruggieri, Franco Turini, Symeon Papadopoulos, Emmanouil Krasanakis, Ioannis Kompatsiaris, Katharina Kinder-Kurlanda, Claudia Wagner, Fariba Karimi, Miriam Fernandez, Harith Alani, Bettina Berendt, Tina Kruegel, Christian Heinze, Klaus Broelemann, Gjergji Kasneci, Thanassis Tiropanis, and Steffen Staab. Bias in data-driven artificial intelligence systems—an introductory survey. *WIREs Data Mining and Knowledge Discovery*, 10(3), feb 2020. doi: 10.1002/widm.1356.

[130] Nikunj C. Oza. Online bagging and boosting. In *IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2005. doi: 10.1109/icsmc.2005.1571498.

[131] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954. doi: https://doi.org/10.1093/biomet/41.1-2.100.

[132] Martin Pawelczyk, Klaus Broelemann, and Gjergji Kasneci. Learning model-agnostic counterfactual explanations for tabular data. In *Proceedings of The Web Conference 2020*. ACM, apr 2020. doi: 10.1145/3366423.3380087.

[133] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. doi: 10.5555/1953048.2078195.

[134] Simon Perkins, Kevin Lacker, and James Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *The Journal of Machine Learning Research*, 3:1333–1356, 2003. doi: 10.5555/944919.944976.

[135] Ali Pesaranghader and Herna L. Viktor. Fast hoeffding drift detection method for evolving data streams. In *Machine Learning and Knowledge Discovery in Databases*, pages 96–111. Springer International Publishing, 2016. doi: 10.1007/978-3-319-46227-1_7.

[136] Ali Pesaranghader, Herna Viktor, and Eric Paquet. Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams. *Machine Learning*, 107(11):1711–1743, 2018.

[137] Ali Pesaranghader, Herna L. Viktor, and Eric Paquet. McDiarmid drift detection methods for evolving data streams. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2018. doi: 10.1109/ijcnn.2018.8489260.

[138] Duncan Potts and Claude Sammut. Incremental learning of linear model trees. *Machine Learning*, 61(1-3):5–48, jun 2005. doi: 10.1007/s10994-005-1121-8.

[139] Mahardhika Pratama, Plamen P. Angelov, Jie Lu, Edwin Lughofer, Manjeevan Seera, and C. P. Lim. A randomized neural network for data streams. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, may 2017. doi: 10.1109/ijcnn.2017.7966286.

[140] John R Quinlan. Learning with continuous classes. In *5th Australian Joint Conference on Artificial Intelligence*, volume 92, pages 343–348. World Scientific, 1992.

[141] Sergio Ramírez-Gallego, Bartosz Krawczyk, Salvador García, Michał Woźniak, and Francisco Herrera. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing*, 239:39–57, may 2017. doi: 10.1016/j.neucom.2017.01.078.

[142] Jesse Read, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes. Batch-incremental versus instance-incremental learning in dynamic and evolving data. In *Advances in Intelligent Data Analysis XI*, pages 313–323. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-34156-4_29.

[143] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. doi: 10.1145/2939672.2939778.

[144] Yao Rong, Tobias Leemann, Vadim Borisov, Gjergji Kasneci, and Enkelejda Kasneci. A consistent and efficient evaluation strategy for attribution methods. *Thirty-ninth International Conference on Machine Learning*, 2022. doi: https://doi.org/10.48550/arXiv.2202.00449.

[145] Gordon J Ross, Niall M Adams, Dimitris K Tasoulis, and David J Hand. Exponentially weighted moving average charts for detecting concept drift. *Pattern recognition letters*, 33(2):191–198, 2012. doi: https://doi.org/10.1016/j.patrec.2011.08.019.

[146] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence, Vol 1, May 2019, 206-215*, November 2018. doi: https://doi.org/10.1038/s42256-019-0048-x.

[147] Leszek Rutkowski, Lena Pietruczuk, Piotr Duda, and Maciej Jaworski. Decision trees for mining data streams based on the mcdiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1272–1279, jun 2013. doi: 10.1109/tkde.2012.66.

[148] Leszek Rutkowski, Maciej Jaworski, Lena Pietruczuk, and Piotr Duda. The CART decision tree for mining data streams. *Information Sciences*, 266:1–15, may 2014. doi: 10.1016/j.ins.2013.12.060.

[149] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. Online deep learning: Learning deep neural networks on the fly. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, jul 2018. doi: 10.24963/ijcai.2018/369.

[150] Tegjyot Singh Sethi and Mehmed Kantardzic. On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications*, 82:77–99, oct 2017. doi: 10.1016/j.eswa.2017.04.008.

[151] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, 06–11 Aug 2017. doi: https://dl.acm.org/doi/10.5555/3305890.3306006.

[152] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv*, 2017. doi: https://doi.org/10.48550/arXiv.1706.03825.

[153] Parinaz Sobhani and Hamid Beigy. New drift detection method for data streams. In *International Conference on Adaptive and Intelligent Systems*, pages 88–97. Springer, 2011. doi: https://doi.org/10.1007/978-3-642-23857-4_12.

[154] Vinicius M. A. Souza, Denis M. dos Reis, André G. Maletzke, and Gustavo E. A. P. A. Batista. Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery*, 34(6):1805–1858, jul 2020. doi: 10.1007/s10618-020-00698-5.

[155] Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM Press, 2001. doi: 10.1145/502512.502568.

[156] Pascal Sturmfels, Scott Lundberg, and Su-In Lee. Visualizing the impact of feature attribution baselines. *Distill*, 2020. doi: 10.23915/distill.00022.

[157] Mukund Sundararajan and Amir Najmi. The many shapley values for model explanation. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9269–9278. PMLR, 13–18 Jul 2020. doi: https://dl.acm.org/doi/abs/10.5555/3524938.3525797.

[158] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR, 06–11 Aug 2017. doi: https://dl.acm.org/doi/10.5555/3305890.3306024.

[159] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant. Sketching linear classifiers over data streams. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, may 2018. doi: 10.1145/3183713.3196930.

[160] Thanh TL Tran, Liping Peng, Boduo Li, Yanlei Diao, and Anna Liu. Pods: a new model and processing algorithms for uncertain data streams. In *Proceedings of*

the 2010 ACM SIGMOD International Conference on Management of data, pages 159–170, 2010. doi: https://doi.org/10.1145/1807167.1807187.

[161] Peter Turney. Technical note: Bias and the quantification of stability. *Machine Learning*, 20(1/2):23–33, 1995. doi: 10.1023/a:1022682001417.

[162] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984. doi: https://doi.org/10.1145/1968.1972.

[163] Alexander Vergara, Shankar Vembu, Tuba Ayhan, Margaret A. Ryan, Margie L. Homer, and Ramón Huerta. Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical*, 166-167:320–329, may 2012. doi: 10.1016/j.snb.2012.01.074.

[164] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, mar 1985. doi: 10.1145/3147.3165.

[165] Heng Wang and Zubin Abraham. Concept drift detection for streaming data. In *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2015. doi: 10.1109/ijcnn.2015.7280398.

[166] Jialei Wang, Peilin Zhao, Steven C. H. Hoi, and Rong Jin. Online feature selection and its applications. *IEEE Transactions on Knowledge and Data Engineering*, 26 (3):698–710, mar 2014. doi: 10.1109/tkde.2013.32.

[167] Jing Wang, Jie Shen, and Ping Li. Provable variable selection for streaming features. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5171–5179. PMLR, 10–15 Jul 2018.

[168] Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30 (4):964–994, apr 2016. doi: 10.1007/s10618-015-0448-4.

[169] Xindong Wu, Kui Yu, Wei Ding, Hao Wang, and Xingquan Zhu. Online feature selection with streaming features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(5):1178–1192, may 2013. doi: 10.1109/tpami.2012.197.

[170] Kui Yu, Xindong Wu, Wei Ding, and Jian Pei. Towards scalable and accurate online feature selection for big data. In *2014 IEEE International Conference on Data Mining*. IEEE, dec 2014. doi: 10.1109/icdm.2014.63.

[171] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the Twentieth International Conference on Machine Learning*, page 856–863. AAAI Press, 2003. ISBN 1577351894.

[172] Shujian Yu, Zubin Abraham, Heng Wang, Mohak Shah, Yantao Wei, and José C. Príncipe. Concept drift detection and adaptation with hierarchical hypothesis testing. *Journal of the Franklin Institute*, 356(5):3187–3215, mar 2019. doi: 10.1016/j.jfranklin.2019.01.043.

[173] Peng Zhao, Le-Wen Cai, and Zhi-Hua Zhou. Handling concept drift via model reuse. *Machine Learning*, 109(3):533–568, oct 2019. doi: 10.1007/s10994-019-05835-w.

[174] Jing Zhou, Dean P Foster, Robert A Stine, Lyle H Ungar, and Isabelle Guyon. Streamwise feature selection. *Journal of Machine Learning Research*, 7(9), 2006. doi: https://dl.acm.org/doi/10.5555/1248547.1248614.

[175] Indre Zliobaite. Learning under concept drift: an overview. *arXiv*, abs/1010.4784, 01 2010. doi: https://doi.org/10.48550/arXiv.1010.4784.

[176] Indrė Žliobaitė. Controlled permutations for testing adaptive learning models. *Knowledge and Information Systems*, 39(3):565–578, mar 2013. doi: 10.1007/s10115-013-0629-7.

Appendix A

Publications

This thesis is based on six papers. All papers are publicly available. The papers provided here are identical to the versions published online.

I am the main contributor and first author of each publication. A more detailed statement about the contributions of each author can be found in the following sections.

A.1 Leveraging Model Inherent Variable Importance for Stable Online Feature Selection

Publication: Published in the proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2020).

Contribution: I developed major parts of the general framework and the implementations of the FIRES model. I also performed the experiments and wrote most parts of the paper. Martin Pawelczyk, Klaus Broelemann, and Gjergji Kasneci contributed to the paper by challenging and improving early ideas, formalizations, and the analysis of results. All co-authors helped revise the final manuscript.

Leveraging Model Inherent Variable Importance for Stable Online Feature Selection

Johannes Haug
University of Tuebingen
Tuebingen, Germany
johannes-christian.haug@uni-tuebingen.de

Martin Pawelczyk
University of Tuebingen
Tuebingen, Germany
martin.pawelczyk@uni-tuebingen.de

Klaus Broelemann
Schufa Holding AG
Wiesbaden, Germany
klaus.broelemann@schufa.de

Gjergji Kasneci
University of Tuebingen
Tuebingen, Germany
gjergji.kasneci@uni-tuebingen.de

ABSTRACT

Feature selection can be a crucial factor in obtaining robust and accurate predictions. Online feature selection models, however, operate under considerable restrictions; they need to efficiently extract salient input features based on a bounded set of observations, while enabling robust and accurate predictions. In this work, we introduce *FIRES*, a novel framework for online feature selection. The proposed feature weighting mechanism leverages the importance information inherent in the parameters of a predictive model. By treating model parameters as random variables, we can penalize features with high uncertainty and thus generate more stable feature sets. Our framework is generic in that it leaves the choice of the underlying model to the user. Strikingly, experiments suggest that the model complexity has only a minor effect on the discriminative power and stability of the selected feature sets. In fact, using a simple linear model, *FIRES* obtains feature sets that compete with state-of-the-art methods, while dramatically reducing computation time. In addition, experiments show that the proposed framework is clearly superior in terms of feature selection stability.

CCS CONCEPTS

• **Computing methodologies** → **Online learning settings**; **Feature selection**; • **Mathematics of computing** → *Dimensionality reduction*; • **Information systems** → *Data streams*; Uncertainty.

KEYWORDS

feature selection; data streams; stability; uncertainty

ACM Reference Format:
Johannes Haug, Martin Pawelczyk, Klaus Broelemann, and Gjergji Kasneci. 2020. Leveraging Model Inherent Variable Importance for Stable Online Feature Selection. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403200

1 INTRODUCTION

Online feature selection has been shown to improve the predictive quality in high-dimensional streaming applications. Aiming for real time predictions, we need online feature selection models that are both effective and efficient. Recently, we also witness a demand for interpretable and stable machine learning methods [31]. Yet, the stability of feature selection models remains largely unexplored.

In practice, feature selection is primarily used to mitigate the so-called *curse of dimensionality*. This term refers to the negative effects on the predictive model that we often observe in high-dimensional applications; such as weak generalization abilities, for example. In this context, feature selection has successfully been applied to both offline and online machine learning applications [3, 8, 14, 22].

Data streams are a potentially unbounded sequence of time steps. As such, data streams preclude us from storing all observations that appear over time. Consequently, at each time step t, feature selection models can analyse only a subset of the data to identify relevant features. Besides, temporal dynamics, e.g. concept drift, may change the underlying data distributions and thereby shift the attentive relation of features [12]. To sustain high predictive power, online feature selection models must be flexible with respect to shifting distributions. For this reason, online feature selection usually proves to be more challenging than batch feature selection.

Online models should not only be flexible with regard to the data distribution, but also robust against small variations of the input or random noise. Otherwise, the reliability of a model may suffer. Robustness is also one of the key requirements of a report published by the European Commission [15]. For online feature selection, this means that we aim to avoid drastic variations of the selected features in subsequent time steps. Yet, whenever a data distribution changes, we must adjust the feature set accordingly. Only few authors have examined the stability of feature selection models [1, 18, 28], which leaves plenty of room for further investigation.

Ideally, we aim to uncover a stable set of discriminative features at every time step t. Feature selection stability, e.g. defined by [28], usually corresponds to a low variation of the selected feature set. We could reduce the variation and thereby maximize the stability of a feature set by selecting only those features we are certain about. Still, we aim to select a feature set that is highly discriminative

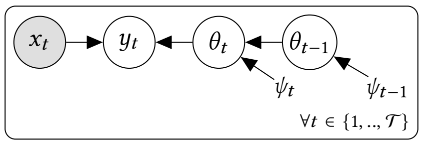

$$\begin{tikzcd} x_t \arrow[r] & y_t & \theta_t \arrow[l] & \theta_{t-1} \arrow[l] \\ & & \psi_t \arrow[u] & \psi_{t-1} \arrow[u] \end{tikzcd}$$

Figure 1: *Graphical Model.* The target variable y_t at time step $t \in \{1,..,\mathcal{T}\}$ depends on a feature vector x_t (observed variable = shaded grey) and model parameters θ_t. We treat the parameters θ_t as random variables that are parameterised by ψ_t, which in turn contains information about feature importance and uncertainty. We update the distribution of θ at every time step t with respect to the new observations.

with respect to the current data generating distribution. In order to meet both requirements, one would have to weigh the features according to their importance and uncertainty regarding the decision task at hand. We translate these considerations into three sensible properties for stable feature weighting in data streams:

Property 1. (Attentive Weights) *Feature weights must preserve attentive relations of features. Given an arbitrary feature x_j, let μ_{tj} be its measured importance and let ω_{tj} be its weight at some time step t. The feature weight ω_{tj} must be a function of μ_{tj}, such that $\mu_{tj} = 0 \Rightarrow \omega_{tj} \leq 0$.*

Intuitively, we would expect the weight of a feature to be exactly zero, if its associated importance is zero. But since there might be dependencies between the different weights, we allow the weights in such cases to become smaller than zero, thus ensuring a higher flexibility in the possible weight configurations.

Property 2. (Monotonic Weights) *Feature weights must be a strictly monotonic function of importance and uncertainty. Given two arbitrary features $x_i \neq x_j$, let $|\mu_{ti}|, |\mu_{tj}|$ be their absolute measured importance and let $\sigma_{ti}, \sigma_{tj} \geq 0$ be the respective measure of uncertainty at some time step t. Two conditions must hold:*

2.1 Given $|\mu_{ti}| = |\mu_{tj}|$, the following holds: $\sigma_{ti} \geq \sigma_{tj} \Leftrightarrow \omega_{ti} \leq \omega_{tj}$. Otherwise, if $\sigma_{ti} < \sigma_{tj}$, meaning we are more certain about feature x_i's than feature x_j's discriminative power, it holds that $\omega_{ti} > \omega_{tj}$, and vice versa.

2.2 Given $\sigma_{ti} = \sigma_{tj}$, the following holds: $|\mu_{ti}| \geq |\mu_{tj}| \Leftrightarrow \omega_{ti} \geq \omega_{tj}$. Otherwise, if $|\mu_{ti}| < |\mu_{tj}|$, meaning that feature x_i is less discriminative than feature x_j, it holds that $\omega_{ti} < \omega_{tj}$, and vice versa.

The second property specifies that features with high importance and low uncertainty must be given a higher weight than features with low importance and high uncertainty.

Property 3. (Consistent Weights) *For a stable target distribution, feature weights must eventually yield a consistent ranking. Let $\mathcal{R}(\omega_t)$ be the ranking of features according to their weights at time step t. Assume $\exists \bar{t}$, such that $P(y_t|x_t) = P(y_{t+1}|x_{t+1})\ \forall t \geq \bar{t}$. As $t \geq \bar{t} \rightarrow \infty$, it holds that $\mathcal{R}(\omega_t) = \mathcal{R}(\omega_{t-1})$.*

Consistent weights eventually yield a stable ranking of features, if the conditional target distribution does not change anymore.

These three properties can help guide the development of robust feature weighting schemes. To the best of our knowledge, they are also the first formal definition of valuable properties for stable feature weighting in data streams.

Table 1: Important Variables and Notation

Notation	Description
$t \in \{1,..,\mathcal{T}\}$	Time step.
$x_t \in \mathbb{R}^{B \times J}$	Observations at time step t with J features and a batch size of B.
$y_t = [y_{t1},..,y_{tB}]$	Target variable at time step t.
$\theta_t = [\theta_{t1},..,\theta_{tk},..,\theta_{tK}]$	Parameters of a model $\mathcal{M}_{\theta_t}$; $K \geq J$.
$P(\theta_{tk}\|\psi_{tk})$	Probability distribution of θ_{tk}, parameterised by ψ_{tk}.
$\omega_t = [\omega_{t1},..,\omega_{tj},..,\omega_{tJ}]$	Feature weights at time step t.
$M \in \mathbb{N}$	Number of selected features; $M \leq J$.

In order to fulfill the properties specified before, we need a measure of feature importance and corresponding uncertainty. If a predictive model $\mathcal{M}_\theta$ is trained on the input features, we expect its parameters θ to contain the required information. Specifically, we can extract the latent importance and uncertainty of features regarding the prediction at time step t, by treating θ_t as a random variable. Accordingly, θ_t is parameterised by ψ_t, which contains the sufficient statistics. Given that all parameters initially follow the same distribution, we can optimize ψ_t for every new observation using gradient updates (e.g. stochastic gradient ascent). If we update ψ_t at every time step, the parameters contain the most current information about the importance and uncertainty of input features. These considerations translate into the graphical model in Figure 1 and form the basis of a novel framework for *F̲ast, I̲nterpretable and R̲obust feature E̲valuation and S̲election* (*FIRES*). *FIRES* selects features with high importance, penalizing high uncertainty, to generate a feature set that is both discriminative and stable.

In summary, the contributions of this work are:

(a) A specification of sensible properties which, when fulfilled, help to create more reliable feature weights in data streams.
(b) A flexible and generic framework for online feature weighting and selection that satisfies the proposed properties.
(c) A concrete application of the proposed framework to three common model families: Generalized Linear Models (GLM) [26], Artificial Neural Nets (ANN) and Soft Decision Trees (SDT) [11, 17] (an open source implementation can be found at https://github.com/haugjo/fires).
(d) An evaluation on several synthetic and real-world data sets, which shows that the proposed framework is superior to existing work in terms of speed, robustness and predictive accuracy.

The remainder of this paper is organized as follows: We introduce the general objective and feature weighting scheme of our framework in Section 2. Here, we also show that *FIRES* produces attentive, monotonic and consistent weights as defined by the Properties 1 to 3. We describe three explicit specifications of *FIRES* in Section 2.1. In Section 2.2, we show how feature selection stability can be evaluated in streaming applications. Finally, we cover related work in Section 3 and evaluate our framework in a series of experiments in Section 4.

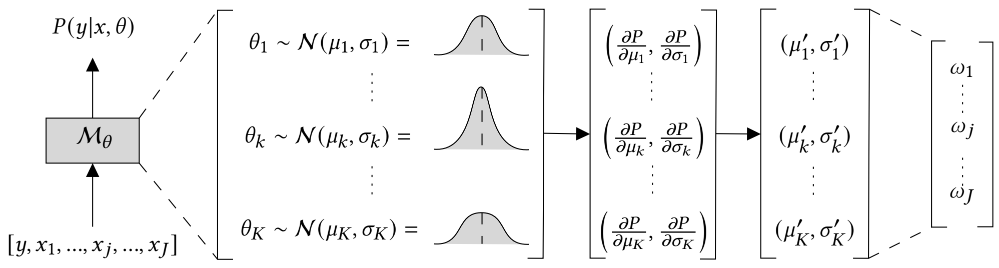

$P(y|x,\theta)$
$\mathcal{M}_\theta$
$[y, x_1, ..., x_j, ..., x_J]$
$\theta_1 \sim \mathcal{N}(\mu_1, \sigma_1) =$
$\theta_k \sim \mathcal{N}(\mu_k, \sigma_k) =$
$\theta_K \sim \mathcal{N}(\mu_K, \sigma_K) =$
$\left(\frac{\partial P}{\partial \mu_1}, \frac{\partial P}{\partial \sigma_1}\right)$
$\left(\frac{\partial P}{\partial \mu_k}, \frac{\partial P}{\partial \sigma_k}\right)$
$\left(\frac{\partial P}{\partial \mu_K}, \frac{\partial P}{\partial \sigma_K}\right)$
(μ'_1, σ'_1)
(μ'_k, σ'_k)
(μ'_K, σ'_K)
ω_1
ω_j
ω_J

Figure 2: *The FIRES Framework.* By treating the parameters of a model $\mathcal{M}_\theta$ as random variables, the proposed framework is able to extract the importance and uncertainty of every input feature with respect to the prediction. For illustration, let θ_k $\forall k$ be normally distributed parameters. *FIRES* optimizes the mean μ_k (importance) and standard deviation σ_k (uncertainty) of all K parameters, by using gradient updates. Based on the updated parameters, *FIRES* then computes feature weights ω_j $\forall j$.

2 THE FIRES FRAMEWORK

The parameters of a predictive model encode every input feature's importance in the prediction. By treating model parameters as random variables, we can quantify the importance and uncertainty of each feature. These estimates can then be used for feature weighting at every time step. This is the core idea of the proposed framework *FIRES*, which is illustrated in Figure 2. Table 1 introduces relevant notation. Let θ be a vector of model parameters whose distribution is parameterised by ψ. Specifically, for every parameter θ_k we choose a distribution, so that ψ_k comprises an importance and uncertainty measure regarding the predictive power of θ_k. We then look for the K distribution parameters that optimize the prediction.

General Objective: We translate these considerations into an objective: Find the distribution parameters ψ that maximize the log-likelihood given the observed data, i.e.

$$\arg\max_{\Psi_\mathcal{T}} \mathcal{L}(\Psi_\mathcal{T}, Y_\mathcal{T}, X_\mathcal{T}) = \arg\max_{\Psi_\mathcal{T}} \sum_{t=1}^{\mathcal{T}} log\, P(y_t|x_t, \psi_t), \quad (1)$$

with observations x_t and corresponding labels y_t. Note that we optimize the logarithm of the likelihood, because it is easier to compute. By the nature of data streams, we never have access to the full data set before time step $\mathcal{T}$. Hence, we cannot compute (1) in closed form. Instead, we optimize ψ incrementally using stochastic gradient ascent. Alternatively, one could also use online variational Bayes [6] to infer posterior parameters. However, gradient based optimization is very efficient, which can be a considerable advantage in data stream applications. The gradient of the log-likelihood with respect to ψ_t is

$$\nabla_{\psi_t}\mathcal{L} = \frac{1}{P(y_t|x_t,\psi_t)} \nabla_{\psi_t} P(y_t|x_t,\psi_t), \quad (2)$$

with the marginal likelihood

$$P(y_t|x_t,\psi_t) = \int P(y_t|x_t,\theta_t)\, P(\theta_t|\psi_t)\, d\theta_t. \quad (3)$$

We update ψ_t with a learning rate α in iterations of the form:

$$\psi'_t = \psi_t + \alpha \nabla_{\psi_t}\mathcal{L} \quad (4)$$

Feature Weighting Scheme: Given the updated distribution parameters, we can compute feature weights in a next step. Note that we may have a one-to-many mapping between input features and model parameters, depending on the predictive model at hand. In this case, we have to aggregate relevant parameters, which we will show in Section 2.1.4. In the following, we assume that there is a single (aggregated) parameter per input feature. Let μ_t, σ_t be the estimated importance and uncertainty of features at time step t. Our goal is to maximize the feature weights ω_t whenever a feature is of high importance and to minimize the weights under high uncertainty. In this way, we aim to obtain optimal feature weights that are both discriminative and stable. We express this trade-off in an objective function:

$$\arg\max_{\omega_\mathcal{T}} \sum_{t=1}^{\mathcal{T}} \left(\underbrace{\sum_{j=1}^{J} \omega_{tj}\mu_{tj}^2}_{\text{importance}} - \lambda_s \underbrace{\sum_{j=1}^{J} \omega_{tj}\sigma_{tj}^2}_{\text{uncertainty}} - \lambda_r \underbrace{\sum_{j=1}^{J} \omega_{tj}^2}_{\text{regularizer}} \right)$$

$$= \arg\max_{\omega_\mathcal{T}} \sum_{t=1}^{\mathcal{T}} \sum_{j=1}^{J} \omega_{tj}(\mu_{tj}^2 - \lambda_s\sigma_{tj}^2 - \lambda_r\omega_{tj}) \quad (5)$$

Note that we regularize the objective with the squared $\ell 2$-norm to obtain small feature weights. Besides, we specify two scaling factors $\lambda_s \geq 0$ and $\lambda_r \geq 0$, which scale the uncertainty penalty and regularization term, respectively. These scaling factors allow us to adjust the sensitivity of the weighting scheme with respect to both penalties. For example, if we have a critical application that requires high robustness (e.g. in medicine), we can increase λ_s to impose a stronger penalty on uncertain parameters. Choosing an adequate λ_s is usually not trivial. In general, a larger λ_s improves the robustness of the feature weights, but limits the flexibility of the model in the face of concept drift.

From now on, we omit time indices to avoid overloading the exposition, e.g. $\omega_t = \omega$. The considerations that follow account for a single time step t. We further assume a batch size of $B = 1$. To maximize (5) for some ω_j, we evaluate the partial derivative at zero:

$$\frac{\partial}{\partial \omega_j} = \mu_j^2 - \lambda_s\sigma_j^2 - 2\lambda_r\omega_j \overset{!}{=} 0$$

$$\Leftrightarrow \ -2\lambda_r\omega_j = -\mu_j^2 + \lambda_s\sigma_j^2$$

$$\Leftrightarrow \ \omega_j^* = \frac{1}{2\lambda_r}\left(\mu_j^2 - \lambda_s\sigma_j^2\right) \quad (6)$$

In accordance with Property 1 to 3, we show that the weights obtained from (6) are attentive, monotonic and consistent:

LEMMA 2.1. *Equation* (6) *produces attentive weights as specified by Property 1.*

PROOF. This property follows immediately from (6). Since $\sigma_j^2 \geq 0$ and $\lambda_s, \lambda_r \geq 0$, for $\mu_j = 0$ we get $\omega_j \leq 0$. □

LEMMA 2.2. *Equation* (6) *produces monotonic weights as specified by Property 2.*

PROOF. Given two features $x_i \neq x_j$, Property 2 specifies two sub-criteria, which we proof independently:

> Given $|\mu_i| = |\mu_j|$, we can show $\sigma_i \geq \sigma_j \Leftrightarrow \omega_i \leq \omega_j$:

$$\sigma_i \geq \sigma_j$$
$$\Leftrightarrow c - b\sigma_i^2 \leq c - b\sigma_j^2; \quad b, c \geq 0$$

For $c = \frac{1}{2\lambda_r}\mu_i^2 = \frac{1}{2\lambda_r}\mu_j^2$ and $b = \frac{\lambda_s}{2\lambda_r}$ we get

$$\Leftrightarrow \frac{1}{2\lambda_r}\mu_i^2 - \frac{\lambda_s}{2\lambda_r}\sigma_i^2 \leq \frac{1}{2\lambda_r}\mu_j^2 - \frac{\lambda_s}{2\lambda_r}\sigma_j^2$$
$$\Leftrightarrow \omega_i \leq \omega_j$$

> Given $\sigma_i = \sigma_j$, we can show $|\mu_i| \geq |\mu_j| \Leftrightarrow \omega_i \geq \omega_j$:

$$|\mu_i| \geq |\mu_j|$$
$$\Leftrightarrow b\mu_i^2 - c \geq b\mu_j^2 - c; \quad b, c \geq 0$$

for $b = \frac{1}{2\lambda_r}$ and $c = \frac{\lambda_s}{2\lambda_r}\sigma_i^2 = \frac{\lambda_s}{2\lambda_r}\sigma_j^2$, we get

$$\Leftrightarrow \frac{1}{2\lambda_r}\mu_i^2 - \frac{\lambda_s}{2\lambda_r}\sigma_i^2 \geq \frac{1}{2\lambda_r}\mu_j^2 - \frac{\lambda_s}{2\lambda_r}\sigma_j^2$$
$$\Leftrightarrow \omega_i \geq \omega_j$$

□

LEMMA 2.3. *Equation* (6) *produces consistent weights as specified by Property 3.*

PROOF. $\exists \bar{t}$, such that

$$P(y_t|x_t, \psi_t) = P(y_{t+1}|x_{t+1}, \psi_{t+1}), \ \forall t \geq \bar{t},$$

which by (3) can be formulated in terms of the marginal likelihood. Consequently, since the marginal likelihood function does not change after time step $\bar{t}$, the SGA updates in (4) will eventually converge to a local optimum. Let $t^* \geq \bar{t}$ be the time of convergence. Notably, t^* specifies the time step at which $P(\theta|\psi)$ and the distribution parameters ψ have been learnt, such that $\psi_t = \psi_{t^*}$ $\forall t \geq t^*$. By (6), we compute feature weights ω as a function of ψ. Consequently, it also holds that $\omega_t = \omega_{t^*}$ $\forall t \geq t^*$. For the ranking of features, denoted by $\mathcal{R}(\omega)$, this implies $\mathcal{R}(\omega_t) = \mathcal{R}(\omega_{t^*})$ $\forall t \geq t^*$. □

2.1 Illustrating the *FIRES* Mechanics

The proposed framework has three variable components, which we can specify according to the requirements of the learning task at hand:

(1) The prior distribution of model parameters θ
(2) The prior distribution of the target variable y
(3) The predictive model $\mathcal{M}_\theta$ used to compute the marginal likelihood (3)

The flexibility of *FIRES* allows us to obtain robust and discriminative feature sets in any streaming scenario. By way of illustration, we make the following assumptions:

The prior distribution of θ must be specified so that ψ contains a measure of importance and uncertainty. The Gaussian normal distribution meets this requirement. In our case, the mean value refers to the expected importance of θ in the prediction. In addition, the standard deviation measures the uncertainty regarding the expected importance. Since the normal distribution is well-explored and occurs in many natural phenomena, it is an obvious choice. Accordingly, we get $\theta_k \sim \mathcal{N}(\psi_k)$ $\forall k$, where ψ_k comprises the mean μ_k and the standard deviation σ_k.

In general, we infer the distribution of the target variable y from the data. For illustration, we assume a Bernoulli distributed target, i.e. $y \in \{-1, 1\}$. Most existing work supports binary classification. The Bernoulli distribution is therefore an appropriate choice for the evaluation of our framework.

Finally, we need to choose a predictive model $\mathcal{M}_\theta$. *FIRES* supports any predictive model type, as long as its parameters represent the importance and uncertainty of the input features. To illustrate this, we apply *FIRES* to three common model families.

2.1.1 ***FIRES And Generalized Linear Models.*** Generalized Linear Models (GLM) [26] use a link function to map linear models of the form $\sum_{j=1}^{J} \theta_j x_j + \theta_{J+1}$ to a target distribution. Since we map to the Bernoulli space, we use the cumulative distribution function of the standard normal distribution, Φ, which is known as a Probit link. Conveniently, we can associate each input feature with a single model parameter. Hence, by using a GLM, we can avoid the previously discussed parameter aggregation step. We discard θ_{J+1}, as it is not linked to any specific input feature. With Lemma A.1 and A.2 (see Appendix) the marginal likelihood becomes

$$P(y = 1|x, \psi) = \int \Phi\left(\sum_{j=1}^{J} \theta_j x_j\right) P(\theta|\psi)\, d\theta$$
$$= \Phi\left(\frac{1}{\rho}\sum_{j=1}^{J} \mu_j x_j\right); \quad \rho = \sqrt{1 + \sum_{j=1}^{J} \sigma_j^2 x_j^2}.$$

Since Φ is symmetric, we can further generalize to

$$P(y|x, \psi) = \Phi\left(\frac{y}{\rho}\sum_{j=1}^{J} \mu_j x_j\right).$$

We then compute the corresponding partial derivatives as

$$\frac{\partial}{\partial \mu_j} P(y|x, \psi) = \phi\left(\frac{y}{\rho}\sum_{i=1}^{J} \mu_i x_i\right) \cdot \frac{y}{\rho} x_j,$$
$$\frac{\partial}{\partial \sigma_j} P(y|x, \psi) = \phi\left(\frac{y}{\rho}\sum_{i=1}^{J} \mu_i x_i\right) \cdot \frac{y}{-2\rho^3} 2x_j^2 \sigma_j \sum_{i=1}^{J} \mu_i x_i,$$

where ϕ is the probability density function of the standard normal distribution.

2.1.2 ***FIRES And Artificial Neural Nets.*** In general, Artificial Neural Nets (ANN) make predictions through a series of linear transformations and nonlinear activations. Due to the nonlinearity and complexity of ANNs, we usually cannot solve the integral of

(3) in closed form. Instead, we approximate the marginal likelihood using the well-known Monte Carlo method. Let $f_\theta(x)$ be an ANN of arbitrary depth. We approximate (3) by sampling L-times with Monte Carlo:

$$\begin{aligned} P(y|x,\psi) &= \int f_\theta(x)\, P(\theta|\psi)\, d\theta \\ &\approx \frac{1}{L}\sum_{l=1}^{L} f_{\theta^{(l)}}(x); \quad \theta_k^{(l)} = \sigma_k r_k^{(l)} + \mu_k\ \forall k \end{aligned}$$

Note that we apply a reparameterisation trick: By sampling $r_k^{(l)} \sim \mathcal{N}(0,1)$, we move stochasticity away from μ_k and σ_k, which allows us to compute their partial derivatives:

$$\frac{\partial P(y|x,\psi)}{\partial \mu_k} = \frac{1}{L}\sum_{l=1}^{L} \frac{\partial}{\partial \theta_k^{(l)}} f_{\theta^{(l)}}(x)\, \frac{\partial \theta_k^{(l)}}{\partial \mu_k} = \frac{1}{L}\sum_{l=1}^{L} \frac{\partial}{\partial \theta_k^{(l)}} f_{\theta^{(l)}}(x),$$

$$\frac{\partial P(y|x,\psi)}{\partial \sigma_k} = \frac{1}{L}\sum_{l=1}^{L} \frac{\partial}{\partial \theta_k^{(l)}} f_{\theta^{(l)}}(x)\, \frac{\partial \theta_k^{(l)}}{\partial \sigma_k} = \frac{1}{L}\sum_{l=1}^{L} \frac{\partial}{\partial \theta_k^{(l)}} f_{\theta^{(l)}}(x)\, r_k^{(l)}$$

We obtain $\frac{\partial}{\partial \theta_k^{(l)}} f_{\theta^{(l)}}(x)$ by backpropagation.

2.1.3 ***FIRES And Soft Decision Trees.*** Binary decisions as in regular CART Decision Trees are not differentiable. Accordingly, we have to choose a Decision Tree model that has differentiable parameters to compute the gradient of (3). One such model is the Soft Decision Tree (SDT) [11, 17]. Let n be the index of an inner node of the SDT. SDTs replace the binary split at n with a logistic function:

$$p_n(x) = \frac{1}{1 + e^{-(\sum_{j=1}^{J} \theta_{nj} x_j)}}$$

This function yields the probability by which we choose n's right child branch given x. Note that the logistic function is differentiable with respect to the parameters θ_n. Similar to ANNs, however, we are now faced with nonlinearity and higher complexity of the model. Therefore, we approximate the marginal likelihood using Monte Carlo and the reparameterisation trick. With f_θ being an SDT, the partial derivatives of (3) correspond to those of the ANN.

2.1.4 ***Aggregating Parameters.*** The number of model parameters $\theta = [\theta_1, .., \theta_k, .., \theta_K]$ might exceed the number of input features $x = [x_1, .., x_j, .., x_J]$, i.e. $K \geq J$. Whenever this is the case, we have to aggregate parameters, since we require a single importance and uncertainty score per input feature to compute feature weights in (6). Next, we show how to aggregate the parameters of an SDT and an ANN to create a meaningful representation.

For SDTs the aggregation is fairly simple. Each inner node is a logistic function that comprises exactly one parameter per input feature. Let N be the number of inner nodes. Accordingly, we have a total of $J \times N$ parameters in the SDT model. We aggregate the parameters associated with an input feature x_j by computing the mean over all inner nodes:

$$\theta_j = \frac{1}{N}\sum_{n=1}^{N} \theta_{nj}; \quad \theta_j \sim \mathcal{N}\left(\frac{1}{N}\sum_{n=1}^{N} \mu_{nj}, \frac{1}{N}\sum_{n=1}^{N} \sigma_{nj}\right)$$

Due to the multi-layer architecture of an ANN, its parameters are usually associated with more than one input feature. For this reason, we cannot apply the same methodology as that proposed for the SDT. Instead, for each input feature x_j, we aggregate all parameters that lie along j's path to the output layer. Let h be the index of a layer of the ANN, where $h = 1$ denotes the input layer. Let further $\mathcal{U}_h^j$ be the set of all nodes of layer h that belong to the path of x_j. We sum up the average parameters along all layers and nodes on j's path to obtain a single aggregated parameter θ_j:

$$\theta_j = \sum_{h=1}^{H-1} \frac{1}{|\mathcal{U}_h^j||\mathcal{U}_{h+1}^j|} \sum_{n \in \mathcal{U}_h^j} \sum_{i \in \mathcal{U}_{h+1}^j} \theta_{ni};$$

$$\theta_j \sim \mathcal{N}\left(\sum_{h=1}^{H-1} \frac{1}{|\mathcal{U}_h^j||\mathcal{U}_{h+1}^j|} \sum_{n \in \mathcal{U}_h^j} \sum_{i \in \mathcal{U}_{h+1}^j} \mu_{ni}, \right.$$
$$\left. \sum_{h=1}^{H-1} \frac{1}{|\mathcal{U}_h^j||\mathcal{U}_{h+1}^j|} \sum_{n \in \mathcal{U}_h^j} \sum_{i \in \mathcal{U}_{h+1}^j} \sigma_{ni}\right)$$

where H is the total number of layers and $|\cdot|$ is the cardinality of a set. Note that θ_{ni} is the parameter that connects node n in layer h to node i in layer $h+1$.

2.2 Feature Selection Stability in Data Streams

In view of increasing threats such as adversarial attacks, the stability of machine learning models has received much attention in recent years [13]. Stability usually describes the robustness of a machine learning model against (adversarial or random) perturbations of the data [5, 32]. In this context, a feature selection model is considered stable, if the set of selected features does not change after we slightly perturb the input [18]. To measure feature selection stability, we can therefore monitor the variability of the feature set [28]. Nogueira et al. [28] developed a stability measure, which is a generalization of various existing methods. Let $\mathcal{Z} = [A_1, .., A_r]^T$ be a matrix that contains r feature vectors, which we denote by $A_r \in \{0,1\}^J$. Specifically, A_r is the feature vector that was obtained for the r'th sample, such that selected features correspond to 1 and 0 otherwise. Feature selection stability according to Nogueira et al. [28] is then defined as:

$$\Gamma(\mathcal{Z}) = 1 - \frac{\frac{1}{J}\sum_{j=1}^{J} s_j^2}{\frac{M}{J}\left(1 - \frac{M}{J}\right)}, \tag{7}$$

where M is the number of selected features and $s_j^2 = \frac{r}{r-1}\hat{p}_j(1-\hat{p}_j)$ is the unbiased sample variance of the selection of feature j. Moreover, let $\hat{p}_j = \frac{1}{r}\sum_{i=1}^{r} z_{ij}$, where z_{ij} denotes one element of $\mathcal{Z}$. According to (7), the feature selection stability decreases, if the total variability $\sum_{j=1}^{J} s_j^2$ increases. On the other hand, if $s_j^2 = 0\ \forall j$, i.e. there is no variability in the selected features, the stability reaches its maximum value at $\Gamma(\mathcal{Z}) = 1$.

Nogueira et al. [28] show that (7) has a clean statistical interpretation. In addition, the measure can be calculated in linear time, making it a sensible choice for evaluating online feature selection models. However, to calculate (7), we would have to sample r feature vectors, which can be costly. A naïve but very efficient approach is to use the feature vectors in a shifting window instead. Let A_t be the active feature set at time step t. We then define $\mathcal{Z}_t = [A_{t-r+1}, ..., A_t]^T$, where r depicts the size of the shifting window. We compute the sample variance s_j^2 in the same fashion as before. However, (7) is now restricted to the observations between time step t and $t - r + 1$.

The shifting window approach allows us to update the stability measure for each new feature vector at each time step. However, the approach might return low stability values, when the shifting window falls within a period of concept drift. Accordingly, we should control the sensitivity of the stability measure by a sensible selection of the window size. An alternative to shifting windows are Cross-Validation based schemes recently proposed by Barddal et al. [1], based on an idea of Bifet et al. [2].

Finally, note that *FIRES* produces consistent feature rankings when the target distribution is stable (Property 3). In this case, we will ultimately achieve zero variance in subsequent feature sets, thereby maximizing the feature selection stability according to (7).

3 RELATED WORK

Traditionally, feature selection models are categorized into filters, wrappers and embedded methods [8, 14, 30]. Embedded methods merge feature selection with the prediction, filter methods are decoupled from the prediction, and wrappers use predictive models to weigh and select features [30]. Evidently, *FIRES* belongs to the group of wrappers.

Data streams are usually defined as an unbounded sequence of observations, where all features are known in advance. This assumption is shared by most related literature. In practice, however, we often observe streaming features, i.e. features that appear successively over time. The approaches dealing with streaming features often assume that only a fixed set of observations is available. We should therefore distinguish between online feature selection methods for "streaming observations" and "streaming features". Next, we present prominent and recent works in both categories.

Streaming Features: Zhou et al. [38] proposed a model that selects features based on a potential reduction of error. The threshold used for feature selection is updated with a penalty method called alpha-investing. Later, Wu et al. [36] introduced *Online Streaming Feature Selection (OSFS)*, which constructs the Markov blanket of a class and gradually removes irrelevant or redundant features. Likewise, the *Scalable and Accurate Online Approach (SAOLA)* removes redundant features by computing a lower bound on pairwise feature correlations [37]. Another approach is *Group Feature Selection from Feature Streams (GFSFS)*, which uncovers and removes all feature groups that have a low mutual information with the target variable [21]. Finally, Wang et al. [34] introduced *Online Leverage Scores for Feature Selection*, a model that selects features based on the approximate statistical leverage score.

Streaming Observations: An early approach is *Grafting*, which uses gradient updates to iteratively adjust feature weights [29]. Later, Nguyen et al. [27] employed an ensemble model called *Heterogeneous Ensemble with Feature Drift for Data Streams (HEFT)* to compute a symmetrical uncertainty measure that can be used to weigh and select features. *Online Feature Selection (OFS)* is another wrapper that adjusts feature weights based on misclassifications of a Perceptron [35]. OFS truncates the weight vector at each time step to retain only the top features. The *Feature Selection on Data Streams (FSDS)* model by Huang et al. [16] maintains an approximated low-rank matrix representation of all observed data. FSDS computes feature weights with a Ridge-regression model trained on the low-rank matrix. Recently, Borisov et al. [4] proposed *Cancel Out*, a sparse layer for feature selection in neural nets, which exploits the gradient information obtained during training. Another recent proposal is the *Adaptive Sparse Confidence-Weighted (ASCW)* model that obtains feature weights from an ensemble of sparse learners [23].

Some online feature selection models can process "streaming features" and "streaming observations" simultaneously. One example is the *Extremal Feature Selection (EFS)* by Carvalho and Cohen [7], which ranks features by the absolute difference between the positive and negative weights of a modified balanced Winnow algorithm. Another approach uses statistical measures like χ^2 to rank and select features [20]. Finally, there are online predictive models that offer embedded feature selection, e.g. *DXMiner* [24], as well as explanation models like *LICON* [19], which quantify the influence of input features. For further consultation of related work, we refer the fellow reader to the surveys of Guyon and Elisseeff [14], Li et al. [22] and Ramírez-Gallego et al. [30].

For the evaluation of our framework, we assume that all features are known in advance. Note, however, that *FIRES* can also support streaming features by dynamically adding parameters to the likelihood model (we leave a detailed analysis for future work).

4 EXPERIMENTS

Next, we evaluate the *FIRES* framework in multiple experiments. Specifically, we compare the three instantiations of *FIRES* introduced above (see Sections 2.1.1 to 2.1.3). Besides, we compare our framework to three state-of-the-art online feature selection models: OFS [35], FSDS [16] and EFS [7]. The hyperparameters of each related model were selected as specified in the corresponding papers. We have also defined a set of default hyperparameters for the three *FIRES* models. Details about the hyperparameter search can be found in the Appendix. We chose a binary classification context for the evaluation, as it is a basic problem that all models should handle well.

4.1 Data Sets

We have limited ourselves to known data sets that are either generated by streaming applications or are closely related to them. Table 2 shows the properties of all data sets. Further information about the data sets and the preprocessing is included in the Appendix.

The Human Activity Recognition (HAR) and the MNIST data set are multiclass data sets. We have transferred both data sets into a binary classification setting as follows: HAR contains measurements of a motion sensor. The labels denote corresponding activities. For our evaluation, we treated HAR as a one-vs-all classification of the activity "Walking". MNIST is a popular digit recognition data set. Here, we chose the label "3" for a one-vs-all classification.

The Gisette and Madelon data sets were introduced as part of a NIPS feature selection challenge. We also used a data set from the 1999 KDD Cup that describes fraudulent and benign network

Table 2: *Data sets.* "Types" denotes the data types included in the data set (continuous, categorical). The synthetic RBF (Radial Basis Function) and RTG (Random Tree Generator) data sets were generated with scikit-multiflow [25].

Name	#Samples	#Features	Types
HAR (one-vs-all)	10,299	562	cont.
Spambase	4,601	57	cont.
Usenet (20 Newsgroups)	5,931	658	cat.
Gisette ('03 NIPS challenge)	7,000	5,000	cont.
Madelon ('03 NIPS challenge)	2,600	500	cont.
Dota	100,000	116	cat.
KDD Cup 1999 Data	100,000	41	cont., cat.
MNIST (one-vs-all)	70,000	784	cont.
RBF (synthetic)	10,000	10,000	cont.
RTG (synthetic)	10,000	450	cont., cat.

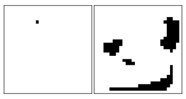
activity. The Dota data set contains (won/lost) results of the strategy game Dota 2. Its features correspond to player information, such as rank or game character. Since KDD and Dota are fairly large data sets, we took a random sample of 100,000 observations each to compute the experiments in a reasonable time. Finally, Usenet is a streaming adaptation of the 20 Newsgroups data set and Spambase contains the specifications of various spam and non-spam emails.

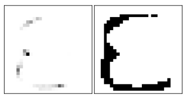
Moreover, we generated two synthetic data sets with scikit-multiflow [25]. Specifically, we obtained 10,000 instances with 10,000 continuous features from the Random RBF Generator (RBF = Radial Basis Function). In addition, we generated another 10,000 instances with 50 categorical and 200 continuous features using the Random Tree Generator (RTG). Here, each categorical feature has five unique, one-hot-encoded values, giving a total of 450 features. We used the default hyperparameters of both generators and defined a random state for reproducibility.

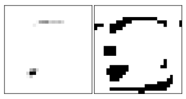
4.2 Results

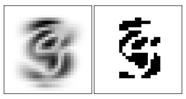
Feature selection generally aims to identify input patterns that are discriminative with respect to the target. We show that *FIRES* does indeed identify discriminative features by selecting the MNIST data set for illustration. The task was to distinguish the class labels 3, 8 and 9, which can be difficult due to their similarity. By successively using each class as the true label, we obtained three different feature weights, which are shown in Figure 3. In this experiment, we used *FIRES* with the ANN base model. Strikingly, while all other models had difficulty in selecting a meaningful set of features, *FIRES* effectively captured the true pattern of the positive class. In fact, *FIRES* has produced feature weights and feature sets that are easy for humans to interpret.

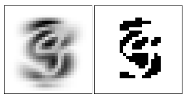
Note that all models we compare are wrapper methods [30]. As such, they use a predictive model for feature selection, but do not make predictions themselves. To assess the predictive power of feature sets, we therefore had to choose an online classifier that was trained on the selected features. We chose a Perceptron algorithm because it is relatively simple, but still impressively demonstrates the positive effect of online feature selection. As Table 3 shows, feature selection can increase the performance of a Perceptron to the point where it can compete with state-of-the-art online predictive

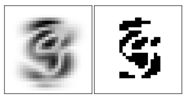

(a) EFS[7], 3 vs. all

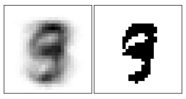

(b) EFS[7], 8 vs. all

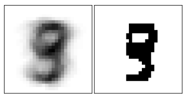

(c) EFS[7], 9 vs. all

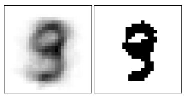

(d) FSDS[16], 3 vs. all

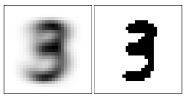

(e) FSDS[16], 8 vs. all

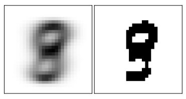

(f) FSDS[16], 9 vs. all


(g) OFS[35], 3 vs. all


(h) OFS[35], 8 vs. all


(i) OFS[35], 9 vs. all


(j) FIRES, 3 vs. all


(k) FIRES, 8 vs. all

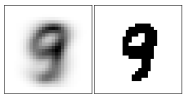

(l) FIRES, 9 vs. all

Figure 3: *Discriminative Features.* We have sampled all observations from MNIST with the label $3, 8$ or 9. We successively selected each label as the positive class and carried out a feature selection. The subplots above illustrate the feature weights (left, high weights being dark) and selected features (right, $M = 115$) after observing the first 1000 instances. While OFS, FSDS and EFS struggle to select meaningful features, *FIRES* (ours) effectively identifies the discriminative features of the positive class after observing very little data.

models. For comparison, we trained an Online Boosting [33] model and a Very Fast Decision Tree (VFDT) [9] on the full feature space. All predictive models were trained in an interleaved test-then-train fashion (i.e. prequential evaluation). The Online Boosting model shows very poor performance for the RBF data set. This is due to the fact that the Naïve Bayes models, which we used as base learners, were unable to enumerate the high-dimensional RBF data set, given the relatively small sample size. Nevertheless, we have kept the same boosting architecture in all experiments for reasons of comparability. We chose accuracy as a prediction metric, because it is a common choice in the literature. Moreover, since we do not take into account extremely imbalanced data, accuracy provides a meaningful assessment of the discriminative power of each model.

Table 3 and Table 4 exhibit the average accuracy, computation time and stability of multiple evaluations with varying batch sizes (25,50,75,100) and fractions of selected features (0.1, 0.15, 0.2). In Figure 4 we show how the accuracy and stability develops over time. In addition, Figure 5 illustrates how the different models manage to balance accuracy and stability. We selected Gisette for illustration, because it is a common benchmark data set for feature selection. Note, however, that we have observed similar results for all remaining data sets.

The results show that our framework is among the top models for online feature selection in terms of computation time, predictive accuracy and stability. Strikingly, *FIRES* coupled with the least

Table 3: *Accuracy and Computation Time.* We tested our framework (*FIRES*) with three different base models; a GLM, an ANN and an SDT (Sections 2.1.1 - 2.1.3). The Online Boosting [33] and Very Fast Decision Tree (VFDT) [9] models were trained on the full feature set to serve as a benchmark. All models are evaluated on an i7-8550U CPU with 16 Gb RAM, running Windows 10. Further details about the experimental setup can be found in the Appendix. Here, we show the accuracy (acc.) and computation time (feature selection + model training; milliseconds) observed on average per time step. Strikingly, while all three *FIRES* models are competitive, *FIRES-GLM* takes first place on average in terms of both predictive accuracy and computation time.

	Benchmark Models				Feature Selection Models (with Perceptron)											
	Boosting [33]		VFDT [9]		FIRES-GLM		FIRES-ANN		FIRES-SDT		OFS [35]		EFS [7]		FSDS [16]	
Datasets	acc.	ms	acc.	ms	acc.	ms	acc.	ms	acc.	ms	acc.	ms	acc.	ms	acc.	ms
HAR	0.788	2264.805	0.819	192.515	0.87	**2.729**	0.842	38.156	0.872	27.497	0.928	20.892	0.87	43.309	**0.919**	4.195
Spambase	**0.83**	234.741	0.638	21.096	0.742	**1.761**	0.685	28.341	0.66	23.333	0.721	10.135	0.577	15.463	0.657	1.969
Usenet	0.507	2448.95	0.504	206.354	0.556	**2.996**	0.541	41.897	0.541	27.948	**0.563**	44.696	0.531	99.758	0.537	4.331
Gisette	0.673	15026.415	0.687	1781.348	**0.933**	**22.391**	0.902	164.682	0.906	42.29	0.911	291.428	0.881	667.923	0.921	36.806
Madelon	**0.551**	1398.29	0.481	245.073	0.523	**2.849**	0.509	38.992	0.508	27.482	0.522	51.872	0.511	82.638	0.539	4.217
Dota	**0.552**	330.895	0.521	54.143	0.516	**2.200**	0.505	30.645	0.505	23.62	0.514	18.084	0.504	24.396	0.505	2.944
KDD	**0.989**	110.659	0.985	12.226	0.969	2.073	0.928	28.614	0.956	22.96	0.962	1.829	0.971	10.756	0.785	**1.977**
MNIST	0.906	1960.237	0.894	275.872	0.930	**3.298**	0.884	43.129	0.940	28.607	**0.950**	14.597	0.879	24.094	0.930	6.075
RBF	0.323	26265.709	0.738	3567.720	0.973	**34.953**	**0.995**	226.017	0.973	38.409	0.748	1356.477	0.973	1594.846	0.984	85.514
RTG	0.812	1244.723	**0.835**	203.776	0.793	**2.609**	0.730	35.698	0.743	26.166	0.743	31.511	0.789	45.706	0.719	4.165
Mean	0.693	5128.542	0.710	656.012	**0.781**	**7.786**	0.752	67.617	0.760	28.831	0.756	184.152	0.749	260.889	0.750	15.219
Rank	8.	8.	7.	7.	**1.**	**1.**	4.	4.	2.	3.	3.	5.	6.	6.	5.	2.

Table 4: *Feature Selection Stability.* Here, we show the average feature selection stability per time step according to (7). The size of the shifting window was 10. All *FIRES* models produce consistently stable feature sets, with the GLM based model ranking first place on average.

	Feature Selection Models					
Datasets	FIRES -GLM	FIRES -ANN	FIRES -SDT	OFS [35]	EFS [7]	FSDS [16]
HAR	0.985	0.652	0.865	0.756	0.921	**0.986**
Spambase	0.901	0.819	0.710	**0.971**	0.822	0.908
Usenet	0.820	0.931	0.747	0.775	0.749	**0.937**
Gisette	0.937	**0.987**	0.756	0.295	0.845	0.949
Madelon	0.526	0.489	0.682	0.158	**0.783**	0.281
Dota	0.978	0.950	0.932	0.444	**0.993**	0.700
KDD	0.997	0.996	0.978	0.940	0.990	**0.999**
MNIST	**0.996**	0.753	0.950	0.703	0.981	0.989
RBF	0.959	**0.993**	0.793	0.018	0.906	0.812
RTG	**0.856**	0.582	0.827	0.457	0.840	0.080
Mean (Rank)	**0.896**	0.815	0.824	0.552	0.883	0.764
Rank	**1.**	3.	4.	6.	2.	5.

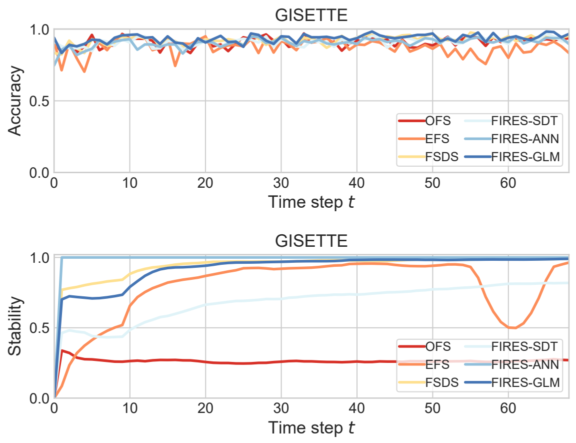


Figure 4: *Accuracy and Stability over Time.* Here, we show how the accuracy and stability scores develop over time. We used the Gisette data for illustration. We trained on batches of size 100 and used a shifting window of size 10 to compute stability (Eq. (7)). All feature selection models achieve similar accuracy. The *FIRES* models maximize stability over time, which was also observed for the remaining data sets.

complex base model (GLM) takes first place on average in all three categories (see Table 3 and 4). The GLM based model generates discriminative feature sets, even if the data is not linearly separable (e.g. MNIST), which may seem strange at first. For feature weighting, however, it is sufficient if we capture the relative importance of the input features in the prediction. Therefore, a model does not necessarily have to have a high classification accuracy. Since GLMs only have to learn a few parameters, they tend to recognize important features faster than other, more complex models. Furthermore, *FIRES-ANN* and *FIRES-SDT* are subject to uncertainty due to the sampling we use to approximate the marginal likelihood. In this experiment, *FIRES-GLM* achieved better results than all related models. Still, whenever a linear model may not be sufficient, the *FIRES* framework is flexible enough to allow base models of higher complexity.

5 CONCLUSION

In this work, we introduced a novel feature selection framework for high-dimensional streaming applications, called *FIRES*. Using probabilistic principles, our framework extracts importance and

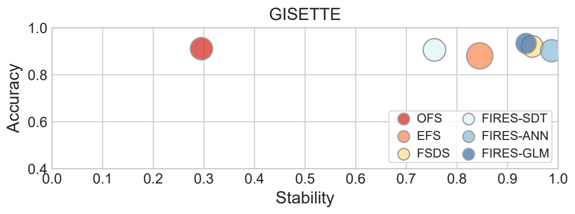


Figure 5: *Accuracy vs. Stability.* In practice, predictions must be both robust and accurate. The proposed framework (*FIRES*) aims at maximizing both aspects and produces feature sets accordingly, which we exemplary show for the Gisette data set. The circle size indicates the variance in accuracy. A perfect result would lie in the top right corner of the plot. Note that in all data sets, we always found at least one *FIRES*-model in the top region (here, it is the models *FIRES-GLM* and *FIRES-ANN*).

uncertainty scores regarding the current predictive power of every input feature. We weigh high importance against uncertainty, producing feature sets that are both discriminative and robust. The proposed framework is modular and can therefore be adapted to the requirements of any learning task. For illustration, we applied *FIRES* to three common linear and nonlinear models and evaluated it using several real-world and synthetic data sets. Experiments show that *FIRES* produces more stable and discriminative feature sets than state-of-the-art online feature selection approaches, while offering a clear advantage in terms of computational efficiency.

REFERENCES

[1] Jean Paul Barddal, Fabrício Enembreck, Heitor Murilo Gomes, Albert Bifet, and Bernhard Pfahringer. 2019. Boosting decision stumps for dynamic feature selection on data streams. *Information Systems* 83 (2019), 13–29.

[2] Albert Bifet, Gianmarco de Francisci Morales, Jesse Read, Geoff Holmes, and Bernhard Pfahringer. 2015. Efficient online evaluation of big data stream classifiers. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 59–68.

[3] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. 2015. Recent advances and emerging challenges of feature selection in the context of big data. *Knowledge-Based Systems* 86 (2015), 33–45.

[4] Vadim Borisov, Johannes Haug, and Gjergji Kasneci. 2019. CancelOut: A Layer for Feature Selection in Deep Neural Networks. In *International Conference on Artificial Neural Networks*. Springer, 72–83.

[5] Olivier Bousquet and André Elisseeff. 2002. Stability and generalization. *Journal of machine learning research* 2, Mar (2002), 499–526.

[6] Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C Wilson, and Michael I Jordan. 2013. Streaming variational bayes. In *Advances in neural information processing systems*. 1727–1735.

[7] Vitor R Carvalho and William W Cohen. 2006. Single-pass online learning: Performance, voting schemes and online feature selection. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 548–553.

[8] Girish Chandrashekar and Ferat Sahin. 2014. A survey on feature selection methods. *Computers & Electrical Engineering* 40, 1 (2014), 16–28.

[9] Pedro Domingos and Geoff Hulten. 2000. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. 71–80.

[10] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[11] Nicholas Frosst and Geoffrey Hinton. 2017. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784* (2017).

[12] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 44.

[13] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. 2018. Making machine learning robust against adversarial inputs. *Commun. ACM* 61, 7 (2018), 56–66.

[14] Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *Journal of machine learning research* 3, Mar (2003), 1157–1182.

[15] Ronan Hamon, Henrik Junklewitz, and Ignacio Sanchez. 2020. Robustness and Explainability of Artificial Intelligence. *Publications Office of the European Union* (2020).

[16] Hao Huang, Shinjae Yoo, and Shiva Prasad Kasiviswanathan. 2015. Unsupervised feature selection on data streams. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 1031–1040.

[17] Ozan Irsoy, Olcay Taner Yıldız, and Ethem Alpaydın. 2012. Soft decision trees. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. IEEE, 1819–1822.

[18] Alexandros Kalousis, Julien Prados, and Melanie Hilario. 2005. Stability of feature selection algorithms. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE, 8–pp.

[19] Gjergji Kasneci and Thomas Gottron. 2016. Licon: A linear weighting scheme for the contribution of input variables in deep artificial neural networks. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*. 45–54.

[20] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. 2005. On the utility of incremental feature selection for the classification of textual data streams. In *Panhellenic Conference on Informatics*. Springer, 338–348.

[21] Haiguang Li, Xindong Wu, Zhao Li, and Wei Ding. 2013. Online group feature selection from feature streams. In *Twenty-seventh AAAI conference on artificial intelligence*.

[22] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. 2018. Feature selection: A data perspective. *ACM Computing Surveys (CSUR)* 50, 6 (2018), 94.

[23] Yanbin Liu, Yan Yan, Ling Chen, Yahong Han, and Yi Yang. 2019. Adaptive Sparse Confidence-Weighted Learning for Online Feature Selection. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 01 (2019), 4408–4415. https://doi.org/10.1609/aaai.v33i01.33014408

[24] Mohammad M Masud, Qing Chen, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. 2010. Classification and novel class detection of data streams in a dynamic feature space. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 337–352.

[25] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. 2018. Scikit-multiflow: a multi-output streaming framework. *The Journal of Machine Learning Research* 19, 1 (2018), 2915–2914.

[26] John Ashworth Nelder and Robert WM Wedderburn. 1972. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)* 135, 3 (1972), 370–384.

[27] Hai-Long Nguyen, Yew-Kwong Woon, Wee-Keong Ng, and Li Wan. 2012. Heterogeneous ensemble for feature drifts in data streams. In *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 1–12.

[28] Sarah Nogueira, Konstantinos Sechidis, and Gavin Brown. 2017. On the Stability of Feature Selection Algorithms. *Journal of Machine Learning Research* 18 (2017), 174–1.

[29] Simon Perkins, Kevin Lacker, and James Theiler. 2003. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of machine learning research* 3, Mar (2003), 1333–1356.

[30] Sergio Ramírez-Gallego, Bartosz Krawczyk, Salvador García, Michał Woźniak, and Francisco Herrera. 2017. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing* 239 (2017), 39–57.

[31] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1, 5 (2019), 206–215.

[32] Peter Turney. 1995. Bias and the quantification of stability. *Machine Learning* 20, 1-2 (1995), 23–33.

[33] Boyu Wang and Joelle Pineau. 2016. Online bagging and boosting for imbalanced data streams. *IEEE Transactions on Knowledge and Data Engineering* 28, 12 (2016), 3353–3366.

[34] Jing Wang, Jie Shen, and Ping Li. 2018. Provable variable selection for streaming features. In *International Conference on Machine Learning*. 5158–5166.

[35] Jialei Wang, Peilin Zhao, Steven CH Hoi, and Rong Jin. 2013. Online feature selection and its applications. *IEEE Transactions on Knowledge and Data Engineering* 26, 3 (2013), 698–710.

[36] Xindong Wu, Kui Yu, Wei Ding, Hao Wang, and Xingquan Zhu. 2012. Online feature selection with streaming features. *IEEE transactions on pattern analysis and machine intelligence* 35, 5 (2012), 1178–1192.

[37] Kui Yu, Xindong Wu, Wei Ding, and Jian Pei. 2014. Towards scalable and accurate online feature selection for big data. In *2014 IEEE International Conference on Data Mining*. IEEE, 660–669.

[38] Jing Zhou, Dean P Foster, Robert A Stine, and Lyle H Ungar. 2006. Streamwise feature selection. *Journal of Machine Learning Research* 7, Sep (2006), 1861–1885.

A IMPORTANT FORMULAE

LEMMA A.1. *Let $X \sim \mathcal{N}(\mu, \sigma)$ be a normally-distributed random variable and let Φ be the cumulative distribution function (CDF) of a standard normal distribution. Then the following equation holds:*

$$\mathbb{E}\left[\Phi(\alpha X + \beta)\right] = \int_{-\infty}^{\infty} \Phi(\alpha x + \beta) P(X = x)\, dx$$
$$= \Phi\left(\frac{\beta + \alpha\mu}{\sqrt{1 + \alpha^2\sigma^2}}\right) \tag{8}$$

PROOF. Let $Y \sim \mathcal{N}(0, 1)$ be a standard-normal-distributed random variable (independent of X). We can then rewrite Φ as: $\Phi(r) = P(Y \leq r)$. Using this, we get:

$$\begin{aligned}&\mathbb{E}\left[\Phi(\alpha X + \beta)\right]\\ =\ &\mathbb{E}\left[P(Y \leq \alpha X + \beta)\right]\\ =\ &P(Y \leq \alpha X + \beta)\\ =\ &P(Y - \alpha X \leq \beta)\end{aligned}$$

The linear combination $Y - \alpha X$ is again normal-distributed with mean $-\alpha\mu$ and variance $\sqrt{1^2 + \alpha^2\sigma^2}$. Hence, we can write

$$Y - \alpha X = \sqrt{1 + \alpha^2\sigma^2} Z - \alpha\mu$$

where $Z \sim \mathcal{N}(0, 1)$ is again standard-normal-distributed. We then get:

$$\begin{aligned}&Y - \alpha X \leq \beta\\ \Leftrightarrow\quad &\sqrt{1 + \alpha^2\sigma^2} Z - \alpha\mu \leq \beta\\ \Leftrightarrow\quad &Z \leq \frac{\beta + \alpha\mu}{\sqrt{1 + \alpha^2\sigma^2}}\end{aligned}$$

Hence, we get

$$\begin{aligned}&\mathbb{E}\left[\Phi(\alpha X + \beta)\right]\\ =\ &P(Y - \alpha X \leq \beta)\\ =\ &P\left(Z \leq \frac{\beta + \alpha\mu}{\sqrt{1 + \alpha^2\sigma^2}}\right)\\ =\ &\Phi\left(\frac{\beta + \alpha\mu}{\sqrt{1 + \alpha^2\sigma^2}}\right)\end{aligned}$$

□

LEMMA A.2. *Let $X = (X_1, ..., X_n)$ be normally-distributed random variables with $X_i \sim \mathcal{N}(\mu_i, \sigma_i)$. Let $\alpha_i, \beta \in \mathbb{R}$ be coefficients and let Φ be the CDF of a standard normal distribution. Then the following equation holds:*

$$\mathbb{E}\left[\Phi\left(\sum_{i=1}^{n} \alpha_i X_i + \beta\right)\right]$$
$$= \int_{-\infty}^{\infty} \cdot\cdot \int_{-\infty}^{\infty} \Phi\left(\sum_{i=1}^{n} \alpha_i x_i + \beta\right) \prod_{i=1}^{n} P(X_i = x_i)\, dx_n \,..\, dx_1$$
$$= \Phi\left(\frac{\beta + \sum_{i=1}^{n} \alpha_i\mu_i}{\sqrt{1 + \sum_{i=1}^{n} \alpha_i^2\sigma_i^2}}\right) \tag{9}$$

PROOF. Proof by mathematical induction over n:
Base case $n = 1$: Lemma A.1

Inductive step $n \mapsto n + 1$: Let (9) hold for n. For $n + 1$, we get:

$$\mathbb{E}\left[\Phi\left(\sum_{i=1}^{n+1} \alpha_i X_i + \beta\right)\right]$$
$$= \mathbb{E}\left[\Phi\left(\sum_{i=1}^{n} \alpha_i X_i + \underbrace{\alpha_{n+1} X_{n+1} + \beta}_{=:\tilde{\beta}}\right)\right]$$
$$\overset{(9)}{=} \mathbb{E}\left[\Phi\left(\frac{\tilde{\beta} + \sum_{i=1}^{n} \alpha_i\mu_i}{\sqrt{1 + \sum_{i=1}^{n} \alpha_i^2\sigma_i^2}}\right)\right]$$
$$= \mathbb{E}\left[\Phi\left(\frac{\alpha_{n+1}}{\sqrt{1 + \sum_{i=1}^{n} \alpha_i^2\sigma_i^2}} X_{n+1} + \frac{\beta + \sum_{i=1}^{n} \alpha_i\mu_i}{\sqrt{1 + \sum_{i=1}^{n} \alpha_i^2\sigma_i^2}}\right)\right]$$
$$\overset{(8)}{=} \Phi\left(\frac{\frac{\beta + \sum_{i=1}^{n} \alpha_i\mu_i}{\sqrt{1 + \sum_{i=1}^{n} \alpha_i^2\sigma_i^2}} + \frac{\alpha_{n+1}}{\sqrt{1 + \sum_{i=1}^{n} \alpha_i^2\sigma_i^2}} \mu_{n+1}}{\sqrt{1 + \left(\frac{\alpha_{n+1}}{\sqrt{1 + \sum_{i=1}^{n} \alpha_i^2\sigma_i^2}}\right)^2 \sigma_{n+1}^2}}\right)$$
$$= \Phi\left(\frac{\beta + \sum_{i=1}^{n} \alpha_i\mu_i + \alpha_{n+1}\mu_{n+1}}{\sqrt{1 + \sum_{i=1}^{n} \alpha_i^2\sigma_i^2 + \alpha_{n+1}^2\sigma_{n+1}^2}}\right)$$
$$= \Phi\left(\frac{\beta + \sum_{i=1}^{n+1} \alpha_i\mu_i}{\sqrt{1 + \sum_{i=1}^{n+1} \alpha_i^2\sigma_i^2}}\right)$$

□

B EXPERIMENTAL SETTING

B.1 Python Packages

All experiments were conducted on an i7-8550U CPU with 16 Gb RAM, running Windows 10. We have set up an Anaconda (v4.8.2) environment with Python (v3.7.1). We mostly used standard Python packages, including *numpy* (v1.16.1), *pandas* (v0.24.1), *scipy* (v1.2.1), and *scikit-learn* (v0.20.2). The ANN base model was implemented with *pytorch* (v1.0.1). Besides, we used the SDT implementation provided at https://github.com/AaronX121/Soft-Decision-Tree/blob/master/SDT.py, extracting the gradients after every training step.

During the evaluation, we further used the *FileStream*, *RandomRBFGenerator*, *RandomTreeGenerator*, *PerceptronMask*, *HoeffdingTree* (VFDT) and *OnlineBoosting* functions of *scikit-multiflow* (v0.4.1). Finally, we generated plots with *matplotlib* (v3.0.2) and *seaborn* (v0.9.0).

B.2 *scikit-multiflow* Hyperparameters

If not explicitly specified here, we have used the default hyperparameters of scikit-multiflow [25].

For the *OnlineBoosting()* function, we have specified the following hyperparameters:

- **base_estimator** = NaiveBayes()
- **n_estimators** = 3
- **drift_detection** = False
- **random_state** = 0

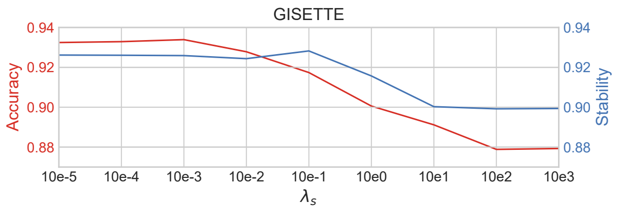


Figure 6: *Choosing the λ_s Penalty Term.* The larger λ_s, the higher we weigh uncertainty against importance when computing feature weights (see Eq. (5)). Here, we illustrate the effect of λ_s on the average accuracy per time step (red) and the average feature selection stability per time step (blue, according to Eq. (7)). We used the Gisette data set for illustration. If we increase λ_s, we initially also increase the stability. Yet, for $\lambda_s > 1$, the feature selection stability suffers. We observed similar effects for all remaining data sets. This suggests that stable feature sets depend not only on the uncertainty, but also on the importance of features. $\lambda_s = 0.01$ was set as our default value throughout the experiments, because it provided the best balance between predictive power and stability in all data sets.

Besides, we used the *HoeffdingTree()* function to train a VFDT model. We set the parameter *leaf_prediction* to *'mc'* (majority class), since the default choice *'nba'* (adaptive Naïve Bayes) is very inefficient in high-dimensional applications.

B.3 *FIRES* Hyperparameters

We assumed a standard normal distribution for all initial model parameters θ, i.e. $\mu_k = 0, \sigma_k = 1\ \forall k$. The remaining hyperparameters of *FIRES* were optimized in a grid search. Below we list the search space and final value of every hyperparameter:

- $\boldsymbol{\alpha}$: Learning rate for updates of μ and σ, see Eq. (4): search space=$[0.01, 0.025, 0.1, 1, 10]$, final value=0.01.
- $\boldsymbol{\lambda_s}$: Penalty factor for uncertainty in the weight objective, see Eq. (5): search space=$[10e-5, 10e-4, 10e-3, 10e-2, 10e-1, 10e0, 10e1, 10e2, 10e3]$, final value=0.01. In Figure 6, we exemplary show the effect of different λ_s for the Gisette data.
- $\boldsymbol{\lambda_r}$: Regularization factor in the weight objective, see Eq. (5): search space=$[0.01, 0.1, 1]$, final value=0.01.

There have been additional hyperparameters for the ANN and SDT based models:

- **Learning rate (ANN+SDT):** Learning rate for gradient updates after backpropagation: search space=$[0.01, 0.1, 1]$, final value=0.01.
- **Monte Carlo samples (ANN+SDT):** Number of times we sample from the parameter distribution in order to compute the Monte Carlo approximation of the marginal likelihood: search space=$[3, 5, 7, 9]$, final value=5.
- **#Hidden layers (ANN):** Number of fully connected hidden layers in the ANN: search space=$[3, 5, 7]$, final value=3
- **Hidden layer size (ANN):** Nodes per hidden layer: search space=$[50, 100, 150, 200]$, final value=100.
- **Tree depth (SDT):** Maximum depth of the SDT: search space=$[3, 5, 7]$, final value=3.
- **Penalty coefficient (SDT):** Frosst and Hinton [11] specify a coefficient that is used to regularize the output of every inner node: search space=$[0.001, 0.01, 0.1]$, final value=0.01.

Note that we have evaluated every possible combination of hyperparameters, choosing the values that maximized the tradeoff between predictive power and stable feature sets. Similar to related work, we selected the search spaces empirically. The values listed above correspond to the default hyperparameters that we have used throughout the experiments.

C DATA SETS AND PREPROCESSING

The Usenet data was obtained from http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift. All remaining data sets are available at the UCI Machine Learning Repository [10]. We used *pandas.factorize()* to encode categorical features. Moreover, we normalized all features into a range $[0, 1]$, using the *MinMaxScaler()* of *scipy*. Otherwise, we did not preprocess the data.

D PSEUDO CODE

Algorithm 1 depicts the pseudo code for the computation of feature weights at time step t. Note that the gradient of the log-likelihood might need to be approximated, depending on the underlying predictive model. In the main paper, we show how to use Monte Carlo approximation to compute the gradient for an Artificial Neural Net (ANN) and a Soft Decision Tree (SDT).

Algorithm 1: Feature weighting with *FIRES* at time step t

```
Data: Observations $x_t \in \mathbb{R}^{B \times J}$ and corresponding labels
      $y_t = [y_{t1}, .., y_{tB}]$ ($B$ = batch size, $J$ = no. of features);
      Sufficient statistics of $K$ model parameters from the
      previous time step: $\mu_{t-1}, \sigma_{t-1} \in \mathbb{R}^K$
Result: Feature weights: $\omega_t \in \mathbb{R}^J$; Updated statistics:
        $\mu_t, \sigma_t \in \mathbb{R}^K$
begin
    /* Define the log-likelihood $\mathcal{L}$ for some base
       model and compute the gradient                    */
    $\nabla_\mu \mathcal{L} \leftarrow$ Eq. (2);
    $\nabla_\sigma \mathcal{L} \leftarrow$ Eq. (2);
    /* Update the sufficient statistics                  */
    $\mu_t = \mu_{t-1} + \alpha_\mu \nabla_\mu \mathcal{L}$;
    $\sigma_t = \sigma_{t-1} + \alpha_\sigma \nabla_\sigma \mathcal{L}$;
    if $\#parameters\ K > \#input\ features\ J$ then
        $\mu'_t, \sigma'_t \in \mathbb{R}^J \leftarrow aggregate(\mu_t, \sigma_t)$;
    end
    /* Compute feature weights                           */
    $\omega_t \leftarrow$ Eq. (6);
end
```

A.2 Learning Parameter Distributions to Detect Concept Drift in Data Streams

Publication: Published in the proceedings of the 25th International Conference on Pattern Recognition (ICPR 2020).

Contribution: Gjergji Kasneci and I had the idea to use the probabilistic framework presented in Paper 1 for concept drift detection. On this basis, I formalized, implemented, and evaluated the ERICS framework. I also wrote most parts of the paper. Gjergji Kasneci was involved in all phases of the project through detailed feedback.

Learning Parameter Distributions to Detect Concept Drift in Data Streams

Johannes Haug
University of Tuebingen
Tuebingen, Germany
johannes-christian.haug@uni-tuebingen.de

Gjergji Kasneci
University of Tuebingen
Tuebingen, Germany
gjergji.kasneci@uni-tuebingen.de

***Abstract*—Data distributions in streaming environments are usually not stationary. In order to maintain a high predictive quality at all times, online learning models need to adapt to distributional changes, which are known as concept drift. The timely and robust identification of concept drift can be difficult, as we never have access to the true distribution of streaming data. In this work, we propose a novel framework for the detection of real concept drift, called *ERICS*. By treating the parameters of a predictive model as random variables, we show that concept drift corresponds to a change in the distribution of optimal parameters. To this end, we adopt common measures from information theory. The proposed framework is completely model-agnostic. By choosing an appropriate base model, *ERICS* is also capable to detect concept drift at the input level, which is a significant advantage over existing approaches. An evaluation on several synthetic and real-world data sets suggests that the proposed framework identifies concept drift more effectively and precisely than various existing works.**

I. Introduction

Data streams are a potentially unbounded sequence of observations. As such, data streams are subject to a number of external factors, e.g. seasonal or catastrophic events. Hence, the distributions of a data stream are usually not stationary, but change over time, which is known as concept drift.

Concept drift can seriously affect the quality of predictions, if it goes unnoticed. Concept drift detection models help identify and handle distributional changes, allowing us to maintain a high predictive performance over time. Ideally, concept drift detection models are sensitive enough to detect drift with only a short delay. However, concept drift detection should also be robust against small perturbations of the input in order to avoid false positives and thus be reliable.

Let X and Y be random variables that correspond to the streaming observations and the associated labels. According to [1], concept drift resembles a difference in the joint probability $P(Y, X)$ at different time steps $t, u \in \{1, .., \mathcal{T}\}$, i.e.

$$\begin{gathered} P_t(Y, X) \neq P_u(Y, X) \\ \Leftrightarrow P_t(Y|X)P_t(X) \neq P_u(Y|X)P_u(X). \end{gathered}$$

We call $P_t(Y, X)$ the active concept at time step t. Moreover, we distinguish between real and virtual concept drift. Virtual concept drift describes a change in $P(X)$, i.e. $P_t(X) \neq P_u(X)$. Hence, virtual concept drift is independent from the target distribution and does not change the decision boundary [2]. On the other hand, real concept drift, sometimes called concept shift, corresponds to a change in the conditional target distribution, i.e. $P_t(Y|X) \neq P_u(Y|X)$. Real concept drift shifts the decision boundary, which may influence subsequent predictions [2]. It is therefore crucial to detect changes of $P(Y|X)$ in time to avoid dramatic drops in predictive performance. In this paper, we investigate the effective and robust identification of real concept drift.

Unfortunately, concept drift does not follow a clear pattern in practice. Instead, we might observe large differences in the duration and magnitude of concept drift. To this end, we distinguish between different types of concept drift [1]–[3]: Sudden drift describes an abrupt change from one concept to another. Incremental drift is a steady transition of concepts over some time period. In a gradual drift, the concepts alternate temporarily, until a new concept ultimately replaces the old one. Sometimes we also observe mixtures of different concept drift types and recurring or cyclic concepts. For further information, we refer the fellow reader to [1]. In general, concept drift detection models should allow timely and accurate detection of all types of concept drift.

In a data stream, we can only access a fraction of the data at every time step t. To detect real concept drift, we thus need to approximate $P_t(Y|X)$, by using a predictive model f_{θ_t}. Accordingly, we get $P_t(Y|X) \approx P(Y|X, \theta_t)$, with parameters $\theta_t = (\theta_{tk})_{k=1}^K$. We optimize the model parameters, given the new observations in every time step. Consequently, θ_t represents our most current information about the active concept at time step t. A concept drift detection model should therefore adhere to changes of the model parameters through the following two properties:

Property 1. Model-Aware Concept Drift Detection. Let θ_t, θ_u be the parameters of a predictive model f_θ at two time steps t and u. Let further $\mathcal{D}$ be a statistical divergence measure (e.g., Kullback–Leibler, Jensen-Shannon, etc.). Concept drift detection is *model-aware*, if for a detected drift between any two time steps t and u, we observe $\mathcal{D}(\theta_t, \theta_u) > 0$.

Accordingly, we associate concept drift with updates of the predictive model f_θ. Given that f_θ is robust, model-awareness reduces the sensitivity of a concept drift detection scheme to random input perturbations, which in turn reduces the risk of false alarms.

Property 2. Explainable Concept Drift Detection. Concept drift detection at time step t is *explainable* with respect to the predictive model f_{θ_t}, if the concept drift can be associated with individual model parameters, i.e. each dimension of θ_t.

If we associate concept drift with individual parameters, we can make more targeted model updates. Hence, we may avoid unnecessary and costly adaptations of the predictive model. Moreover, some parameter distributions even allow us to relate concept drift to specific input features. In this way, concept drift becomes much more transparent.

In this paper, we propose a novel framework for *Effective and Robust Identification of Concept Shift (ERICS). ERICS* complies with the Properties 1 and 2. We use the probabilistic framework introduced in [4] to model the distribution of the parameters θ at every time step. Specifically, we express real concept drift in terms of the marginal likelihood and the parameter distribution $P(\theta;\psi)$, which is itself parameterized by ψ. Unlike many existing models, *ERICS* does not need to access the streaming data directly [5]. Instead, we detect concept drift by investigating the differential entropy and Kullback-Leibler (KL) divergence of $P(\theta;\psi)$ at different time steps. In this context, we show that concept drift corresponds to changes in the distributional uncertainty of model parameters. In other words, real concept drift can be measured as a change in the average number of bits required to encode the parameters of the predictive model. By specifying an adequate parameter distribution, we can identify concept drift at the input level, which offers a significant advantage over existing approaches in terms of explainability. In fact, the proposed framework can be applied to almost any parameter distribution and online predictive model. For illustration, we apply *ERICS* to a Probit model. In experiments on both synthetic and real-world data sets, we show that the proposed framework can detect different types of concept drift, while having a lower average delay than state-of-the-art methods. Indeed, *ERICS* outperforms existing approaches with respect to the recall and precision of concept drift alerts.

In summary, we propose a generic and flexible framework that leverages the uncertainty patterns of model parameters for more effective concept drift detection in data streams. An open source version of *ERICS* is available at https://github.com/haugjo/erics.

II. ERICS: A Concept Drift Detection Framework

Real concept drift corresponds to a change of the conditional target distribution $P(Y|X)$ [1]. However, data streams are potentially infinite and so the true distribution $P(Y|X)$ remains unknown. Hence, we may use a predictive model f_θ to approximate $P(Y|X)$. Since we update the model parameters θ for every new observation, θ_t represents our most current information about the active concept at time step t. Consequently, we may identify concept drift by investigating changes in θ over time.

To this end, we adopt the general framework of [4] and treat the parameters θ as a random variable, i.e. $\theta \sim P(\theta;\psi)$. Analogously, we optimize the distribution parameters ψ at every time step with respect to the log-likelihood. This optimization problem can be expressed in terms of the marginal likelihood $P(Y|X,\psi)$ [4]. Hence, the marginal likelihood relates to the optimal parameter distribution under the active concept. Accordingly, we may associate concept drift between two time steps t and u with a difference of the marginal likelihood for the distribution parameters ψ_t and ψ_u:

$$\begin{aligned} &P(Y|X;\psi_t) \neq P(Y|X;\psi_u) \\ \Leftrightarrow\ &|P(Y|X;\psi_t) - P(Y|X;\psi_u)| > 0 \\ \Leftrightarrow\ &\left| \int P(Y|X,\theta)\big[P(\theta;\psi_t) - P(\theta;\psi_u)\big]\ d\theta \right| > 0. \end{aligned} \quad (1)$$

From (1), we may obtain a general scheme for concept drift detection. To this end, we rephrase (1) in terms of the differential entropy and KL-divergence, which are common measures from information theory. The entropy of a random variable corresponds to the average degree of uncertainty of the possible outcomes. Besides, entropy is often described as the average number of bits required to encode a sample of the distribution. On the other hand, the KL-divergence measures the difference between two probability distributions. It is frequently applied in Bayesian inference models, where it describes the information gained by updating from a prior to a posterior distribution. We can derive the following proportionality:

$$\begin{aligned} &\int P(Y|X,\theta)\big[P(\theta;\psi_t) - P(\theta;\psi_u)\big]\ d\theta \\ &\propto \int P(\theta;\psi_t)\ d\theta - \int P(\theta;\psi_u)\ d\theta \\ &\propto \int P(\theta;\psi_t)\log P(\theta;\psi_t)\ d\theta - \int P(\theta;\psi_u)\log P(\theta;\psi_t)\ d\theta \\ &= H[P(\theta;\psi_u), P(\theta;\psi_t)] - h[P(\theta;\psi_t)] \\ &= h[P(\theta;\psi_u)] - h[P(\theta;\psi_t)] + D_{KL}[P(\theta;\psi_u)\|P(\theta;\psi_t)], \end{aligned} \quad (2)$$

where $h[P(\theta;\psi_t)]$ is the differential entropy of the parameter distribution at time step t. Note that we have rephrased the cross entropy $H[P(\theta;\psi_u), P(\theta;\psi_t)]$ by using the KL-divergence D_{KL}. We may now substitute (2) into (1) to derive a general scheme for concept drift detection:

$$\Big|\underbrace{h[P(\theta;\psi_u)] - h[P(\theta;\psi_t)]}_{\Delta\text{Uncertainty}} + \underbrace{D_{KL}[P(\theta;\psi_u)\|P(\theta;\psi_t)]}_{\Delta\text{Distribution}}\Big| > 0 \quad (3)$$

Intuitively, real concept drift thus corresponds to a change in the uncertainty of the optimal parameters and a divergence of the parameter distribution. On the other hand, stable concepts are characterized by a static parameter distribution and uncertainty.

Note that (3) has another interpretation in the context of Bayesian inference. As mentioned before, the KL-divergence $D_{KL}[P(\theta;\psi_u)\|P(\theta;\psi_t)]$ can be interpreted as the information gained from inferring the posterior $P(\theta;\psi_u)$ from a prior $P(\theta;\psi_t)$. According to (3), we thus find that every difference in parameter uncertainty (entropy) between time step t and

u, which can not be attributed to the inference of posterior parameters, may be traced back to a concept drift.

Finally, we show that the proposed concept drift detection scheme adheres to the Properties 1 and 2.

Proof that ERICS is model-aware (Property 1): By construction, we model the parameters θ through a distribution $P(\theta;\psi)$. According to (3), we write

$$\Big|\int P(\theta;\psi_t)\log P(\theta;\psi_t)\ d\theta - \int P(\theta;\psi_u)\log P(\theta;\psi_t)\ d\theta\Big|,$$

which is 0 iff $P(\theta;\psi_t) = P(\theta;\psi_u)$. Consequently, we find that Equation (3) evaluates to true, iff $P(\theta;\psi_t) \neq P(\theta;\psi_u)$. By definition, for any sensible statistical divergence measure $\mathcal{D}$, we know that $\mathcal{D}(P(\theta;\psi_t), P(\theta;\psi_u)) = 0 \Leftrightarrow P(\theta;\psi_t) = P(\theta;\psi_u)$. Equation (3) holds true, and thus $P(\theta;\psi_t) \neq P(\theta;\psi_u) \Leftrightarrow \mathcal{D}(P(\theta;\psi_t), P(\theta;\psi_u)) > 0$ ■

Proof that ERICS is explainable (Property 2): By construction, any parametric distribution $P(\theta;\psi)$ used in Equation (3) can be evaluated for each parameter individually, i.e. we have $P(\theta_k;\psi_k)\ \forall k$. ■

A. Continuous Concept Drift Detection

Based on the general scheme (3), we are able to identify concept drift between any two time steps t and u. In practice, we are mainly interested in concept drifts between successive time steps $t-1$ and t. However, if we were to study (3) for two time steps only, our concept drift detection model might become too sensitive to random variations of the predictive model. To be more robust, we examine the moving average of (3) instead. Specifically, we compute the moving average at time step t over M time steps as

$$\text{MA}_t = \frac{1}{M}\sum_{i=t-M+1}^{t}\Big(\big|h[P(\theta;\psi_i)] - h[P(\theta;\psi_{i-1})] + D_{KL}[P(\theta;\psi_i)\|P(\theta;\psi_{i-1})]\big|\Big). \quad (4)$$

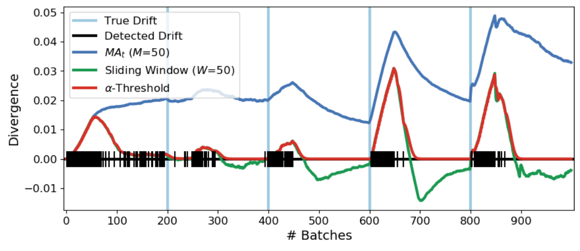
As before, the moving average contains our latest information on the model parameters and the active concept. We can adjust the sensitivity of our framework by selecting M appropriately. In general, the larger we select M, the more robust the framework becomes. However, a large M might also hide concept drifts of small magnitude or short duration.

So far we have treated all changes of the parameter distribution as an indication of concept drift. Indeed, this is in line with the general definition of concept drift [1]. Still, we argue that only certain changes in the parameter distribution have practical relevance. For example, suppose that we use stochastic gradient descent (SGD) to optimize the model parameters at every time step. If we start from an arbitrary initialization, the distribution of optimal parameters usually changes significantly in early training iterations. However, given that the concept $P(Y|X)$ is stationary, SGD will almost surely converge to a local optimum. Consequently, we will ultimately minimize the entropy and KL-divergence of $P(\theta;\psi)$ in successive time steps. In other words, (4) will tend to decrease as long as we optimize the parameters ψ with respect to the active concept. However, if the decision boundary changes due to a real concept drift, SGD-updates will aim for a different optimum. This change of the objective will temporarily lead to more uncertainty in the model and thus increase the entropy of the parameter distribution.

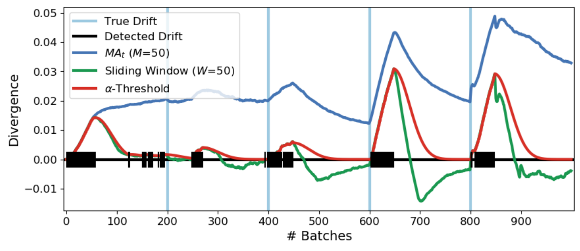
We exploit this temporal pattern for concept drift detection. To this end, we measure the total change of (4) in a sliding window of size W:

$$\sum_{j=t-W+1}^{t}\big(\text{MA}_j - \text{MA}_{j-1}\big) > \alpha_t \Leftrightarrow \text{Drift at } t, \quad (5)$$

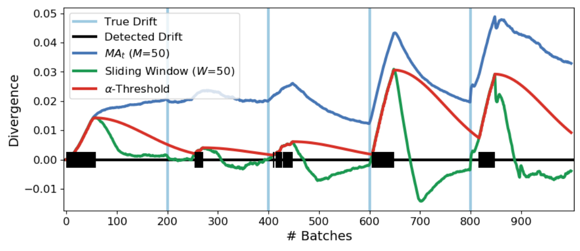
where $\alpha_t \geq 0$ is an adaptive threshold. As before, we may control the robustness of the concept drift detection with the



(a) $\beta = 0.01$



(b) $\beta = 0.001$



(c) $\beta = 0.0001$

Fig. 1. *Updating the α-Threshold.* The proposed framework uses a dynamic threshold α (red line) to detect concept drift. According to (4), we track a moving average of the divergence of the parameter distribution (dark blue line). If the total divergence in a sliding window (green line) exceeds the threshold, *ERICS* detects a concept drift (black vertical lines). By adjusting the hyperparameter β, we can control the iterative updates of α and thus regulate the sensitivity of *ERICS* after a drift is detected. Generally, the larger we choose β, the more sensitive *ERICS* becomes to changes of the parameter distribution. Here, we depict different β for the KDD data set [6]. We artificially generated four sudden concept drifts (blue vertical lines). In this example, small update steps (i.e. small β) are preferable to give the predictive model enough time to adapt to the new concept. Note that the early alerts correspond to the initial training phase of the predictive model. Hence, we would ignore them in practice.

sliding window size W. Whenever we detect concept drift, i.e. (5) evaluates to true, we redefine α_t as

$$\alpha_t = \sum_{j=t-W+1}^{t} \big(\text{MA}_j - \text{MA}_{j-1}\big). \tag{6}$$

In this way, we temporarily tolerate all changes to the predictive model up to a magnitude of (6). We consider these changes to be the after-effects of the concept drift. We then update α_t in an iterative fashion. Let β be a user-defined hyperparameter in the interval $[0,1]$. Each update depends on the current α-value, the β-hyperparameter and the time elapsed since the last concept drift alert, which we denote by Δ_{Drift}:

$$\alpha_t = \alpha_{t-1} - (\alpha_{t-1} * \beta * \Delta_{Drift}) \tag{7}$$

Note that α_t will asymptotically approach 0 over time, if there is no concept drift. In this way, we gradually reduce the tolerance of our framework after a drift is detected.

The choice of a suitable β usually depends on the application at hand. By way of illustration, we applied *ERICS* with different β to the KDD data set [6]. We used [7]'s method to induce sudden concept drift after every 20% of observations. For more information, see Section V. Figure 1 illustrates the components of *ERICS* for three different β-values. Notably, the larger we chose β, the more drifts we detected. Since we were dealing with a sudden concept drift in this particular example, we could be less sensitive and apply smaller update steps. For $\beta = 0.0001$, we achieved good first results in all our experiments. Therefore, this value can generally be used as a starting point for further optimization.

To conclude our general framework, we provide a pseudo code implementation in Figure 2.

```
Require: $[\psi_t, .., \psi_{t-M}]$; $[\text{MA}_{t-1}, .., \text{MA}_{t-W}]$; $\alpha_{t-1}$; $\Delta_{Drift}$
  $\alpha_t \leftarrow Eq.~(7)$
  $\Delta_{Drift} \leftarrow \Delta_{Drift} + 1$
  $\text{MA}_t \leftarrow Eq.~(4)$
  $sumWindow \leftarrow \sum_{j=t-W+1}^{t} \big(\text{MA}_j - \text{MA}_{j-1}\big)$

  if $sumWindow > \alpha_t$ then
    $\alpha_t \leftarrow sumWindow$
    $\Delta_{Drift} \leftarrow 1$
  end if

  return $\alpha_t$; $\text{MA}_t$; $\Delta_{Drift}$
```

Fig. 2. *Pseudo Code.* Concept drift detection with *ERICS* at time step t.

B. Limitations and Advantages

The proposed framework does not access streaming observations directly, but uses the parameters of a predictive model instead. Accordingly, our approach is much more memory efficient than many related works. Yet, if the parameter distribution does not change in a drift period, concept drift may go unnoticed. In general, however, *ERICS* can detect all concept drifts that affect the predictive outcome.

One should also be aware that some predictive models are prone to adversarial attacks. Accordingly, *ERICS* can only be as robust as its underlying predictive model. This sensitivity to the predictive model is shared by most existing works. With *ERICS*, the possibility of misuse is drastically reduced, as we closely monitor the distribution of the model parameters at all times.

III. Illustrating ERICS

ERICS is model-agnostic. This means that the framework can be applied to different predictive models f_θ and parameter distributions $P(\theta;\psi)$. In this way, we enable maximum flexibility with regard to possible streaming applications. By way of illustration, we adopt a Probit model with independent normally distributed parameters. This setup has achieved state-of-the-art results in online feature selection [4]. Besides, it offers dramatic computational advantages due to its low complexity. In line with [4], we optimize ψ at every time step with respect to the log-likelihood for the Probit model.

The assumption of independent model parameters may appear restrictive, but in practice it often leads to good results, e.g. in the case of local feature attributions [8], [9] or feature selection [4], [10]. In fact, the independence assumption allows us to identify the parameters affected by concept drift and thus to comply with Property 2. Since the Probit model comprises one parameter per input feature, we can readily associate concept drift with individual input variables.

Accordingly, let $P(\theta;\psi_t) = \mathcal{N}(\psi_t = (\mu_t, \Sigma_t))$, where $\mu_t = (\mu_{tk})_{k=1}^K$ is a vector of mean values and Σ_t is the diagonal covariance matrix, where the diagonal entries correspond to the vector $\sigma_t^2 = (\sigma_{tk}^2)_{k=1}^K$. The differential entropy of $P(\theta;\psi_t)$ is

$$h\big[P(\theta;\psi_t)\big] = \frac{1}{2}\Big(K + K\ln(2\pi) + \ln\prod_{k=1}^{K}\sigma_{tk}^2\Big).$$

The KL-divergence between $P(\theta;\psi_t)$ and $P(\theta;\psi_{t-1})$ is

$$D_{KL}[P(\theta;\psi_t)\|P(\theta;\psi_{t-1})] \\ = \frac{1}{2}\left(\sum_{k=1}^{K}\frac{\sigma_{tk}^2 + (\mu_{t-1,k}-\mu_{tk})^2}{\sigma_{t-1,k}^2} - K + \ln\frac{\prod_{k=1}^K \sigma_{t-1,k}^2}{\prod_{k=1}^K \sigma_{tk}^2}\right).$$

According to (4), we then write the moving average as

$$\text{MA}_t = \frac{1}{2M}\sum_{i=t-M+1}^{t}\left|\sum_{k=1}^{K}\frac{\sigma_{ik}^2 + (\mu_{i-1,k}-\mu_{ik})^2}{\sigma_{i-1,k}^2} - K\right|. \tag{8}$$

Note that (8) scales linearly with the number of parameters K, i.e. it has $\mathcal{O}(K)$ time complexity.

In order to identify concept drift at individual parameters (which is equivalent to examining individual features, since we use a Probit model), we can investigate the moving average of a specific parameter θ_k:

$$\text{MA}_{tk} = \frac{1}{2M}\sum_{i=t-M+1}^{t}\left|\frac{\sigma_{ik}^2 + (\mu_{i-1,k}-\mu_{ik})^2}{\sigma_{i-1,k}^2} - 1\right| \tag{9}$$

In this case, we maintain a different threshold α_k per parameter. Note that (9) has a constant time complexity.

IV. Related Work

In this section, we briefly introduce some of the most prominent and recent contributions to concept drift detection.

DDM monitors changes in the classification error of a predictive model [11]. Whenever the observed error changes significantly, DDM issues a warning or an alert. We find various modifications of this general scheme, including [12] and [13]. Another well-known method for concept drift adaptation is ADWIN [14]. Here, the authors maintain a sliding window, whose size changes dynamically according to the current rate of distributional change. [15] also employ a sliding window approach and provide a feasible implementation of Fisher's Exact test, which they use for concept drift detection. Similar to our framework, [16] use a sliding window and the entropy to detect concept drift. However, they examine entropy with regard to the predictive result and disregard the model parameters. FHDDM applies a sliding window to classification results and tracks significant differences between the current probability of correct predictions and the previously observed maximal probability [17]. To this end, FHDDM employs a threshold that is based on the Hoeffding bound. In a later approach, the same authors instead use McDiarmid's inequality to detect concept drift [18]. EWMA is a method that monitors an increase in the probability that observations are misclassified [19]. The authors use an exponentially weighted moving average, which places greater weight on the most recent instances in order to detect changes. [20] also focus on the predictive outcome. Specifically, they investigate the distribution of the loss function via resampling. Likewise, the LFR method uses certain test statistics to detect concept drift by identifying changes through statistical hypothesis testing [21]. Finally, [22] compare the labels of close data points in successive batches to detect concept drift.

In addition, we find various approaches that examine ensembles of online learners to deal with concept drift. For example, [23] compare two models; one that is trained with all streaming observations and another that is trained only with the latest observations. Likewise, [24] analyze the density of the posterior distributions of an incremental and a static estimator.

More information about the progress in concept drift detection can be found in [1]–[3], [25].

Conceptually, our work differs substantially from the remaining literature. Instead of directly examining the streaming observations or the predictive outcome, *ERICS* monitors changes in the parameters of a predictive model.

V. Experiments

We evaluated *ERICS* in multiple experiments. All experiments were conducted on an i5-8250U CPU with 8 Gb of RAM, running 64-bit Windows 10 and Python 3.7.3. We compared our framework to the popular concept drift detection methods ADWIN [14], DDM [11], EWMA [19], FHDDM [17], MDDM [18] and RDDM [13]. We used the predefined implementations of these models as provided by the Tornado framework [26]. Besides, we applied the default set of parameters throughout all experiments. Note that all related models require classifications of a predictive model. To this end, we trained a Very Fast Decision Tree (VFDT) [27] in an interleaved test-then-train evaluation. The VFDT is a state-of-the-art online learner, which uses the Hoeffding bound to incrementally construct a decision tree for streaming data. We used the VFDT implementation of scikit-multiflow [28] in our experiments. Note that we consider a simple binary classification scenario in all our experiments, since it should be handled well by all models.

TABLE I
Synthetic and Real World Data Sets

Name	#Samples	#Features	Data Types
SEA (synth.)	100,000	3	cont.
Agrawal (synth.)	100,000	9	cont.
Hyperplane (synth.)	100,000	20	cont.
Mixed (synth.)	100,000	9	cont.
Spambase	4,599	57	cont.
Adult	48,840	54	cont., cat.
HAR (binary)	7,450	562	cont.
KDD (sample)	100,000	41	cont., cat.
Dota	102,944	116	cat.
MNIST (binary)	10,398	784	cont.

TABLE II
Hyperparameters of ERICS per Data Set

Data Set	M	W	β	Epochs	LR μ	LR σ
SEA	75	50	0.0001	10	0.01	0.01
Agrawal	100	50	0.001	10	0.01	0.01
Hyperplane	100	50	0.0001	10	0.01	0.01
Mixed	100	50	0.0001	10	0.1	0.01
Spambase	35	25	0.001	10	0.1	0.01
Adult	50	50	0.001	10	0.1	0.01
HAR	25	50	0.001	10	0.1	0.01
KDD	50	50	0.0001	10	0.01	0.01
Dota	75	50	0.0001	10	0.01	0.01
MNIST	25	20	0.001	50	0.1	0.01

We optimized the hyperparameters of *ERICS* in a grid search. The search space was either chosen empirically or according to [4]. Table II lists all hyperparameters per data set. The hyperparameters "Epochs", "LR (learning rate) μ" and "LR σ" control the training of the Probit model, which we adopted from [4].

A. Data Sets

In order to evaluate the timeliness and precision of a concept drift detection model, we require ground truth. Consequently, we generated multiple synthetic data sets using the scikit-multiflow package [28]. Detailed information about each generator can be obtained from the corresponding documentation. We exhibit the properties of all data sets in Table I. Note that we simulated multiple types of concept drift. Specifically, we produced sudden concept drifts with the SEA generator. To this end, we specified a drift duration (*width* parameter) of 1. We alternated between the classification functions 0-3 to produce the different concepts. With the Agrawal generator, we simulated gradual drift of different duration. Again, we alternated between the classification functions 0-3 to shift the data distribution. With the rotating Hyperplane generator, we simulated an incremental drift over the full length of the data set. We generated 20 features with the Hyperplane generator, out of which 10 features were subject to concept drift by a magnitude of 0.5. Finally, we produced a Mixed drift using the Agrawal generator. The Mixed data contains both sudden and gradual drift, which we obtained by alternating the classification functions 0-4. All synthetic data sets contain 10% noisy data. We obtained 100,000 observations from each data stream generator.

In addition, we evaluated the proposed framework on real world data. However, since real world data usually does not provide any ground truth information, we had to artificially induce concept drift. For this reason, we applied the methodology of [7] to induce sudden concept drift in five well-known data sets from the online learning literature. First, we randomly shuffled the data to remove any natural (unknown) concept drifts. Next, we ranked all features according to their information gain. We then selected the top 50% of the ranked features and randomly permuted their values. In this way, we generated sudden drifts after every 20% of the observations. Specifically, we introduced concept drift to the real-world data sets Spambase, Adult, Human Activity Recognition (HAR), KDD 1999 and Dota2, which we took from the UCI Machine Learning repository [6]. Note that we drew a random sample of 100,000 observations from the KDD 1999 data to allow for feasible computations.

Besides, we used the MNIST data set to evaluate partial concept drift detection at the input level. We selected all observations that are either labelled *3* or *8*, since these numbers are difficult to distinguish. In the first half of the observations, we treated *3* as the true class. In the second half of the observations, we switched the true class to *8*. In this way, we simulated a sudden concept drift of all input features.

For all real world data sets, we normalized the continuous features to the range $[0, 1]$ and one-hot-encoded the categorical features. In the Adult data set, we imputed all NaN-values by a new category *unknown*. Moreover, we altered the labels of the HAR data set to simulate binary classification between the class *moving* (original labels *walking*, *walking_downstairs* and *walking_upstairs*) and *non-moving* (original labels *sitting*, *laying* and *standing*). We trained the online predictive models (Probit and VFDT) in batches of the following size: For Spambase and HAR we chose a batch size of 10. Adult was processed in batches of size 50. For all remaining data sets, we trained on batches of 100 observations.

B. Delay, Recall and Precision

In our first experiment, we applied the concept drift detection models to all synthetic and real-world data sets. Figure 3 exhibits the drift alerts of every model. The blue vertical lines and shaded areas indicate periods of concept drift. Each black vertical line corresponds to one drift alert. Most models identify concept drift in early iterations. This is due to the initial training phase of the predictive model and therefore has no practical relevance. For the upcoming evaluations, we have therefore ignored all drift alerts in the first 80 batches.

TABLE III
Average Delay in Number of Batches

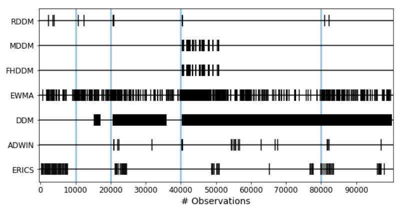
	Drift Detection Models						
Datasets	ERICS	ADWIN	DDM	EWMA	FHDDM	MDDM	RDDM
SEA	52.75	34.55	16.45	**0.26**	178.61	178.58	7.42
Agrawal	71.33	16.80	**0.00**	0.31	0.023	0.20	137.22
Hyperplane	**2.00**	42.27	2.23	2.36	11.24	11.22	2.42
Mixed	34.00	27.95	**0.34**	11.55	75.26	75.25	227.98
Spambase	**7.35**	79.0	60.23	29.96	71.63	71.58	117.45
Adult	**2.61**	63.15	488.39	172.57	488.39	488.39	488.39
HAR	13.05	372.45	372.45	**1.55**	372.45	372.45	77.10
KDD	22.01	50.35	**0.00**	0.55	100.25	100.21	13.21
Dota	44.04	25.32	514.72	43.40	3.56	**3.54**	116.46
Mean	**27.68**	79.09	161.65	29.17	144.60	144.60	131.96
Rank	**1**	3	7	2	5	5	4

By Figure 3, the proposed framework *ERICS* performs well in all data sets. Given the low complexity of the underlying Probit model, some concept drifts do not infer a change of the parameter distribution immediately. This can be seen in small delays, such as for the Agrawal data, for example. Still, *ERICS* achieves the smallest average delay of all concept drift detection models, which is shown in Table III.

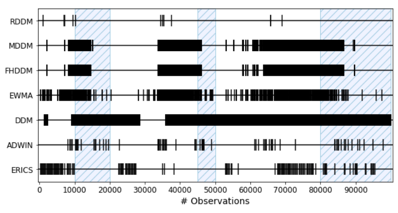
Strikingly, *ERICS* generally seems to produce fewer false alarms than related models. We find support for this intuition by examining the average recall (Figure 4) and precision (Figure 5) over all data sets. Similar to [29], we evaluated the detected drifts for different detection ranges. The detection range corresponds to the number of batches after a known drift, during which we consider an alert as a true positive. Whenever there is no drift alert in the detection range, we count this as a false negative. Besides, all drift alerts outside of the detection range are false positives. We used these scores to compute the recall and precision values. Again, we find that *ERICS* tends to struggle in the early stages, right after a drift happens. As mentioned before, we attribute this to the slowly updating Probit model that we used for illustration. The VFDT, which is used by all related models, is much more complex and can thus adapt to changes faster. Additionally, we must treat some recall scores with care. For example, in four data sets, the DDM model detects drift in almost every time step. Hence, it achieves perfect recall, although the drift alerts are not reliable at all. Still, *ERICS* ultimately outperforms all related models in terms of both recall and precision. The superiority of our framework is even more apparent, if we look at the harmonic mean of precision and recall, which is the F1 score that we show in Figure 6.

C. Detecting Drift at the Input Level

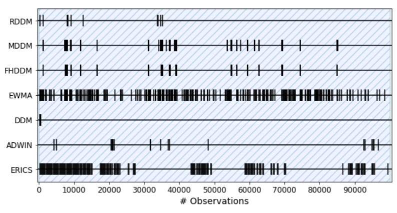
As mentioned before, by using a Probit model and treating parameters as independently Gaussian distributed, we are able to associate concept drift with specific input features. By means of illustration, we apply *ERICS* to a sample of the MNIST data set, which we induced with concept drift. In Figure 7, we exhibit the mean of all observations corresponding to the true class before and after the concept drift (left



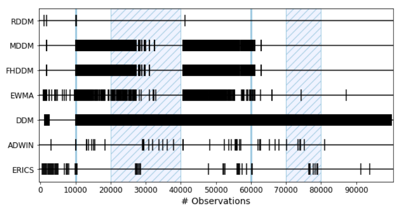
(a) SEA



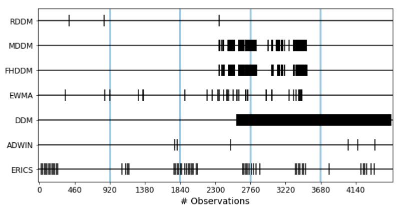
(b) Agrawal



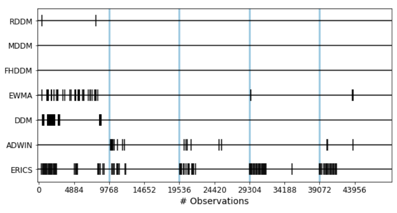
(c) Hyperplane



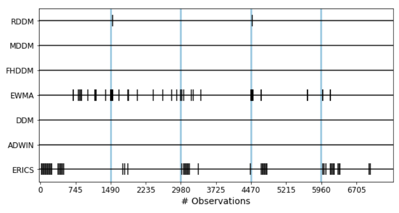
(d) Mixed



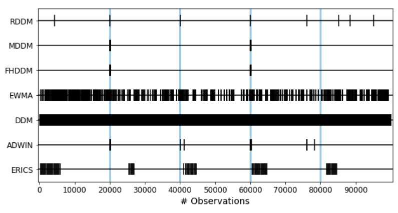
(e) Spambase



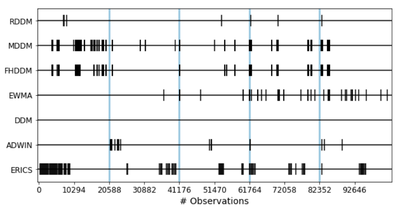
(f) Adult



(g) HAR



(h) KDD



(i) Dota

Fig. 3. *Drifts Alerts.* For all data sets, we illustrate the drift alerts obtained from each concept drift detection model. The blue vertical lines and shaded areas correspond to known concept drifts. Each black marker stands for one drift alert. Early drift alerts can be attributed to the initial training of the predictive model and were therefore ignored. Notably, *ERICS* (ours) seems to detect most concept drifts, while triggering considerably fewer false alarms than most related models. We find support for this intuition in the remaining figures.

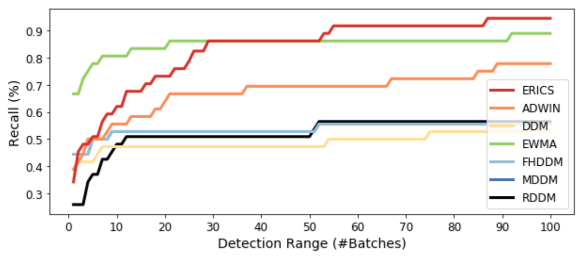


Fig. 4. *Recall.* We show the average recall over all data sets for different detection ranges. *ERICS* (ours) ultimately detects more than 90% of the known concept drifts. The apparent disadvantage of *ERICS* in early batches can be attributed to the slower update speed of the Probit model as compared to the VFDT [27], which was used by the remaining concept drift detection methods. Besides, the recall scores should be considered with care, since some methods tend to detect drift at almost every time step and are thus not reliable.

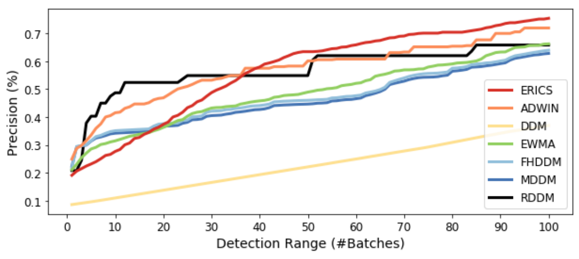


Fig. 5. *Precision.* We show the average precision over all data sets for different detection ranges. *ERICS* (ours) tends to identify concept drift later than some related models. Therefore, we count fewer true positives in small detection ranges, which leads to lower precision. However, for larger detection ranges, our framework is more precise than any other model in the evaluation.

subplots). We also show the absolute difference between those mean values. In the outer most subplot on the right, we illustrate the drift alerts per input feature in the first 15 batches after the concept drift. The color intensity corresponds to the number of drift alerts (where many alerts correspond to darker patterns). Strikingly, the frequency of drift alerts closely maps the absolute difference between the two concepts. This shows that *ERICS* is generally able to identify the input features that are most affected by concept drift. We expect this pattern to become even clearer, when using more complex base models.

VI. Conclusion

In this work, we proposed a novel and generic framework for the detection of concept drift in streaming applications. Our framework monitors changes in the parameters of a predictive model to effectively identify distributional changes of the input. We exploit common measures from information theory, by showing that real concept drift corresponds to

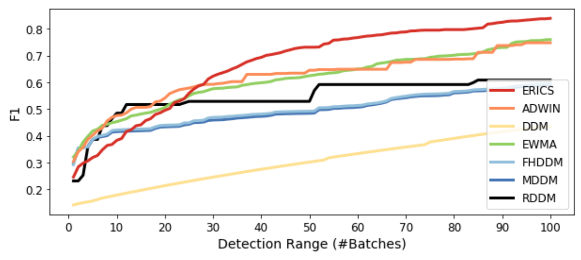


Fig. 6. *F1:* We illustrate the F1 measure, which is the harmonic mean of the precision and recall shown in earlier plots. Here, the advantage of *ERICS* (ours) is most apparent, since it significantly outperforms all related methods for a detection range greater than 30 batches.

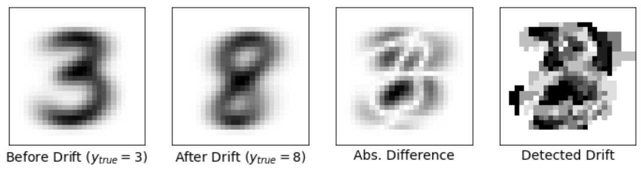


Fig. 7. *Partial Drift Detection.* By choosing an appropriate base model and parameter distribution, *ERICS* can attribute concept drift to individual input features. We selected all observations of MNIST with the label *3* or *8* and induced concept drift by changing the true class after half of the observations. In the left subplots, we exhibit the mean of the true class before and after the concept drift. The third subplot depicts the absolute difference of these mean values. In the right subplot, we show the alerts of *ERICS* in the first 15 batches after the concept drift. The color intensity corresponds to the frequency of drift alerts per input feature. Strikingly, the drift alerts seem to map the absolute difference between both concepts. This suggests, that *ERICS* does indeed identify concept drift for those input features that are most affected by a distributional change.

changes of the uncertainty regarding the optimal parameters. Given an appropriate parameter distribution, the proposed framework can also attribute drift to specific input features. In experiments, we highlighted the advantages of our approach over multiple existing methods, using both synthetic and real-world data. Strikingly, *ERICS* detects concept drift with less delay on average, while outperforming existing models in terms of both recall and precision.

References

[1] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 964–994, 2016.
[2] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, p. 44, 2014.
[3] I. Žliobaitė, "Learning under concept drift: an overview," *arXiv preprint arXiv:1010.4784*, 2010.
[4] J. Haug, M. Pawelczyk, K. Broelemann, and G. Kasneci, "Leveraging model inherent variable importance for stable online feature selection," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1478–1502.
[5] P. Zhao, L.-W. Cai, and Z.-H. Zhou, "Handling concept drift via model reuse," *Machine Learning*, vol. 109, no. 3, pp. 533–568, 2020.
[6] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[7] T. S. Sethi and M. Kantardzic, "On the reliable detection of concept drift from streaming unlabeled data," *Expert Systems with Applications*, vol. 82, pp. 77–99, 2017.
[8] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in neural information processing systems*, 2017, pp. 4765–4774.
[9] G. Kasneci and T. Gottron, "Licon: A linear weighting scheme for the contribution ofinput variables in deep artificial neural networks," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016, pp. 45–54.
[10] V. Borisov, J. Haug, and G. Kasneci, "Cancelout: A layer for feature selection in deep neural networks," in *International Conference on Artificial Neural Networks*. Springer, 2019, pp. 72–83.
[11] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Brazilian symposium on artificial intelligence*. Springer, 2004, pp. 286–295.
[12] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early drift detection method," in *Fourth international workshop on knowledge discovery from data streams*, vol. 6, 2006, pp. 77–86.
[13] R. S. Barros, D. R. Cabral, P. M. Gonçalves Jr, and S. G. Santos, "Rddm: Reactive drift detection method," *Expert Systems with Applications*, vol. 90, pp. 344–355, 2017.
[14] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 2007, pp. 443–448.
[15] D. R. de Lima Cabral and R. S. M. de Barros, "Concept drift detection based on fisher's exact test," *Information Sciences*, vol. 442, pp. 220–234, 2018.
[16] L. Du, Q. Song, and X. Jia, "Detecting concept drift: an information entropy based method using an adaptive sliding window," *Intelligent Data Analysis*, vol. 18, no. 3, pp. 337–364, 2014.
[17] A. Pesaranghader and H. L. Viktor, "Fast hoeffding drift detection method for evolving data streams," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2016, pp. 96–111.
[18] A. Pesaranghader, H. L. Viktor, and E. Paquet, "Mcdiarmid drift detection methods for evolving data streams," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–9.
[19] G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand, "Exponentially weighted moving average charts for detecting concept drift," *Pattern recognition letters*, vol. 33, no. 2, pp. 191–198, 2012.
[20] M. Harel, S. Mannor, R. El-Yaniv, and K. Crammer, "Concept drift detection through resampling," in *International Conference on Machine Learning*, 2014, pp. 1009–1017.
[21] H. Wang and Z. Abraham, "Concept drift detection for streaming data," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–9.
[22] P. Sobhani and H. Beigy, "New drift detection method for data streams," in *International conference on adaptive and intelligent systems*. Springer, 2011, pp. 88–97.
[23] S. H. Bach and M. A. Maloof, "Paired learners for concept drift," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 23–32.
[24] C. H. Tan, V. Lee, and M. Salehi, "Online semi-supervised concept drift detection with density estimation," *arXiv preprint arXiv:1909.11251*, 2019.
[25] P. M. Gonçalves Jr, S. G. de Carvalho Santos, R. S. Barros, and D. C. Vieira, "A comparative study on concept drift detectors," *Expert Systems with Applications*, vol. 41, no. 18, pp. 8144–8156, 2014.
[26] A. Pesaranghader, H. Viktor, and E. Paquet, "Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams," *Machine Learning*, vol. 107, no. 11, pp. 1711–1743, 2018.
[27] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 97–106.
[28] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 2915–2914, 2018.
[29] S. Yu and Z. Abraham, "Concept drift detection with hierarchical hypothesis testing," in *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 2017, pp. 768–776.

A.3 On Baselines for Local Feature Attributions

Publication: Published as part of the Explainable Agency in Artificial Intelligence Workshop at the 35th AAAI Conference on Artificial Intelligence (AAAI 2021)

Contribution: I developed the taxonomy of existing attribution baselines, conducted the literature review, and wrote large sections of the paper. I also outlined the general evaluation strategy. The experiments were conducted as part of a student project by Stefan Zürn and Peter El-Jiz, which I supervised. Gjergji Kasneci challenged our ideas and helped us revise the experiments and the manuscript.

On Baselines for Local Feature Attributions

Johannes Haug, Stefan Zürn, Peter El-Jiz, Gjergji Kasneci
University of Tuebingen, Germany
{johannes-christian.haug, gjergji.kasneci}@uni-tuebingen.de;{stefan.zuern, peter.el-jiz}@student.uni-tuebingen.de

Abstract

High-performing predictive models, such as neural nets, usually operate as black boxes, which raises serious concerns about their interpretability. Local feature attribution methods help to explain black box models and are therefore a powerful tool for assessing the reliability and fairness of predictions. To this end, most attribution models compare the importance of input features with a reference value, often called baseline. Recent studies show that the baseline can heavily impact the quality of feature attributions. Yet, we frequently find simplistic baselines, such as the zero vector, in practice. In this paper, we show empirically that baselines can significantly alter the discriminative power of feature attributions. We conduct our analysis on tabular data sets, thus complementing recent works on image data. Besides, we propose a new taxonomy of baseline methods. Our experimental study illustrates the sensitivity of popular attribution models to the baseline, thus laying the foundation for a more in-depth discussion on sensible baseline methods for tabular data.

Introduction

Neural nets and other complex predictive models perform well in a variety of applications. In practice, however, complex black-box models can give rise to serious concerns about interpretability. Feature weighting and local attribution methods help to explain complex models and thus improve the interpretability of predictions (Haug et al. 2020; Kasneci and Gottron 2016; Lundberg and Lee 2017; Ribeiro, Singh, and Guestrin 2016; Shrikumar, Greenside, and Kundaje 2017; Sundararajan, Taly, and Yan 2017). Accordingly, feature attributions are an important step towards more transparent and fair machine learning.

Local attribution methods usually assess the importance of features with respect to a reference input or baseline value (Sundararajan and Najmi 2019; Izzo et al. 2020). The baseline is closely related to the concept of missingness, i.e. the (approximate) neutral value that a feature would take, if it were considered missing. For example, Lundberg and Lee (2017) replace the missing values in a sampled feature coalition with their expected value and Sundararajan, Taly, and Yan (2017) propose to use a black image as baseline in object recognition tasks.

The concept of missingness is domain-specific and may therefore vary in practice. For example, suppose $x_i \in [0, 255]$ denotes one pixel of an image. Here, $x_i = 0$ is a black pixel, which may indeed be considered missing information. However, suppose that $x_i \in \{0, 1, 2, 3\}$ is a nominal feature. In this case, $x_i = 0$ does not represent missingness. Accordingly, the zero baseline considered in this example would render the generated attributions meaningless. Indeed, several authors have recently expressed concerns about the appropriateness of popular baseline methods (Sundararajan and Najmi 2019; Sturmfels, Lundberg, and Lee 2020; Izzo et al. 2020). In practice, a baseline should be selected with respect to the data distribution at hand.

In this paper, we propose a novel taxonomy of baseline methods, which may help compare and select baselines in research and practice. Along the taxonomy, we briefly introduce common baseline methods (Lundberg and Lee 2017; Sundararajan and Najmi 2019; Izzo et al. 2020; Sturmfels, Lundberg, and Lee 2020). In a next step, we investigate the effect of different baselines on the feature attributions generated by four state-of-the-art attribution models. Our experiments focus on tabular data sets, which are often used as benchmarks in the explainability and fairness literature. We thereby complement and extend a recent study of Sturmfels, Lundberg, and Lee (2020), which illustrates the effects of different baselines for the classification of images. Our results suggest that the baseline can have a dramatic impact on the discriminative quality of the generated feature attributions. Strikingly, there was no universally best-performing baseline method. However, certain baselines rarely yielded discriminative feature attributions, suggesting that they may not be suitable for tabular data.

In summary, our contribution is two-fold: We categorize existing baseline methods into a new taxonomy. Besides, we provide an experimental evaluation of common baselines using several attribution models and tabular data sets. Hence, our work may serve as a reference for a deeper discussion of baseline methods in the context of tabular data and a more principled choice of baselines in practical applications.

A Taxonomy of Baseline Methods

Many different baseline methods have been introduced in recent years (Sundararajan and Najmi 2019; Izzo et al. 2020; Sturmfels, Lundberg, and Lee 2020). Yet, to the best of our

Table 1: **Baseline Taxonomy.** We categorize common baseline methods according to the Definitions 1 and 2.

Baseline Name	Static/Dynamic	Deterministic/Stochastic
Constant (e.g. zero baseline)	static	deterministic
Maximum Distance (Sturmfels, Lundberg, and Lee 2020)	dynamic	deterministic
Blurred (Fong and Vedaldi 2017; Sturmfels, Lundberg, and Lee 2020)	dynamic	stochastic
Gaussian (Smilkov et al. 2017; Sturmfels, Lundberg, and Lee 2020)	dynamic	stochastic
Uniform (Sturmfels, Lundberg, and Lee 2020)	dynamic	stochastic
Expectation (Lundberg and Lee 2017)	static	stochastic
Neutral (Izzo et al. 2020)	static	deterministic

knowledge, there is no generally accepted scheme for categorising and comparing these methods. To this end, we propose a novel taxonomy.

Let $x_i, x_j \sim X$ be two arbitrary observations. Suppose we apply a baseline method B to obtain a baseline corresponding to each observation, i.e. $B(x_i, x_j) = b_{x_i}, b_{x_j}$. We then define

Definition 1 (Static or Dynamic Baseline) *A baseline method B is static, if $\forall i, j : b_{x_i} = b_{x_j}$. Otherwise, B is dynamic, i.e. $\exists i, j : b_{x_i} \neq b_{x_j}$.*

Intuitively, a static baseline method provides the same baseline value for every observation, whereas a dynamic baseline method may provide different values.

Next, suppose that we run the baseline method $B(x_i)$ two times for the observation x_i, which corresponds to $B^1(x_i) = b^1_{x_i}$ (first run) and $B^2(x_i) = b^2_{x_i}$ (second run). We then define

Definition 2 (Deterministic or Stochastic Baseline) *A baseline method B is deterministic, if the probability $Pr(b^1_{x_i} = b^2_{x_i}) = 1$. Otherwise, B is stochastic, i.e. $Pr(b^1_{x_i} = b^2_{x_i}) < 1$.*

Intuitively, a deterministic baseline method always produces the same baseline with respect to an observation x_i whenever it is called. On the other hand, a stochastic baseline method may be subject to variation.

Categorizing Common Baseline Methods

Next, we briefly introduce some popular and recent baseline methods. Table 1 shows the categorization of all methods according to the taxonomy defined above (Definition 1-2).

The *constant* baseline is a prominent static and deterministic baseline. As the name suggests, the constant baseline is a fixed value that is specified once. Note that the zero baseline (i.e. black image) mentioned earlier is an instantiation of the constant baseline.

The *maximum distance* baseline corresponds to an observation that is furthest away from the observation in question by the $\ell 1$-norm (Sturmfels, Lundberg, and Lee 2020). Notably, the maximum distance baseline is dynamic and deterministic.

The *blurred* baseline was originally introduced for image data (Sturmfels, Lundberg, and Lee 2020). Specifically, this baseline method applies a Gaussian blur filter to the observation in question (Fong and Vedaldi 2017). Note that the filter blurs each input feature with respect to its adjacent features. Accordingly, the blurred baseline requires an inherent sense of neighbourhood among input features, which might not always be evident in tabular data. In general, however, we may apply the blurred baseline to (numeric) tabular data as well. The blurred baseline method is dynamic and stochastic.

Similar to the blurred baseline, the *Gaussian* baseline introduces noise to the original observation. To this end, one specifies a Gaussian distribution per input feature, which is centered at the original input value (Smilkov et al. 2017; Sturmfels, Lundberg, and Lee 2020). One then draws random samples from the generated distributions. Hence, the Gaussian baseline is dynamic and stochastic.

Likewise, we may draw random samples from uniform distributions per input feature. The uniform distributions are defined in the valid range of the original features (Sturmfels, Lundberg, and Lee 2020). Again, the *uniform* baseline is dynamic and stochastic.

Lundberg and Lee (2017) specified the baseline as a function of the expectation of a reference sample. We call this the *expectation* baseline, which is static and stochastic as the reference sample is usually drawn randomly from the training data.

Finally, Izzo et al. (2020) argue that a baseline should lie on the decision boundary of the predictive model. Given that the decision boundary does not shift, this *neutral* baseline is static and deterministic. At the time of writing this paper, the neutral baseline was only specified for certain neural network architectures and no open source implementation was available. For this reason, we did not consider the neutral baseline in our experiments.

Experiments

Next, we evaluated the baseline methods shown in Table 1. As mentioned above, we did not consider the recently proposed neutral baseline (Izzo et al. 2020) in this evaluation. All experiments were conducted on an NVIDIA GeForce GTX 1050 TI GPU with Intel i5 7500 CPU and 16Gb RAM. Our machine ran Linux Fedora 32 and Python 3.7.6.

We selected four state-of-the-art local attribution methods to illustrate the effect of different baselines. Specifically, we used KernelSHAP (Lundberg and Lee 2017), DeepSHAP (Lundberg and Lee 2017), DeepLift (Shrikumar, Greenside, and Kundaje 2017) and Integrated Gradients (IG) (Sundararajan, Taly, and Yan 2017). Note that the authors of SHAP, Lundberg and Lee (2017), substitute missing features based on the training distribution, which we called expecta-

Table 2: **Data Sets.** "Feature Types" describes the data types included in the data set (cont. = continuous, cat. = categorical). "Class Imbalance" indicates whether the target distribution is imbalanced. For imbalanced data sets, we indicate the proportion of observations that belong to the positive class in parentheses. Note that in the Communities data, we have removed features with a high proportion of missing values to ensure valid evaluations. Besides, we have obtained a random sample of 50,000 observations from the Fraud Detection data set.

Dataset	# Observations	# Features	Feature Types	# Classes	Class Imbalance
Human Activity Recognition (Dua and Graff 2017)	10,299	561	cont.	6	no
Fraud Detection (Dal Pozzolo et al. 2014)	50,000	30	cont.	2	yes (99.8% pos. class)
Communities (Dua and Graff 2017)	1,993	100	cat./cont.	2	yes (72% pos. class)
Spambase (Dua and Graff 2017)	4,601	57	cont.	2	no
COMPAS (Angwin et al. 2016)	7,214	11	cat./cont.	2	no

tion baseline above. Lundberg and Lee (2017) thereby aim to align their work with earlier methods that approximate the Shapley value. Still, it is worth considering the SHAP framework in our evaluation, as some applications may require different baselines. In contrast, DeepLift and IG leave the choice of a baseline to the user, apart from suggesting the zero vector as a meaningful baseline for image recognition tasks. Note that all open source packages readily allow the user to set a baseline.

We used a top-K ablation test to quantify the discriminative power of the generated feature attributions regarding the different baselines. Accordingly, we masked K percent of the most highly attributed input features with random noise and measured the effect on the generated F1 score. The F1 score is the harmonic mean of precision and recall and provides valid results, even if the target class is imbalanced. Ablation tests are a popular and intuitive evaluation technique for feature attribution methods. Still, ablation tests should always be considered with care, since they do not consider feature interactions. In general, the evaluation of explanation models is subject to ongoing discussions in the research community.

For DeepSHAP, DeepLIFT and IG we trained a simple neural network with one hidden layer and a ReLu activation, using a sigmoid activation function at the output layer. Note that we deliberately chose a shallow architecture, since complex nets tend to mitigate noise at the input. In this way, we wanted to maintain the effect that masking certain input features had on predictive performance, thus providing an unbiased view of the effect of the different baselines. Finally, since KernelSHAP is model agnostic, we applied it to a Support Vector Machine.

As discussed above, the blurred baseline assumes some form of neighbourhood among input features. Since tabular data is usually ordered arbitrarily, we computed the blurred baseline for 1,000 feature permutations and averaged the results. Accordingly, we examined every feature with respect to different neighbouring features, thereby mitigating the effect of the initial ordering of the features.

All experiments are also available on our GitHub page.[1]

[1]https://github.com/ITZuern/On-Baselines-for-Local-Feature-Attributions

Data Sets

A summary of all data sets can be found in Table 2. We standardized (zero mean, unit variance) the continuous features of each data set. Besides, we encoded every non-numeric categorical feature in integers. Finally, we split all data sets, with 80% of the observations used for training and 20% for testing.

The Human Activity Recognition (HAR) data set contains sensor signals of a waist-mounted accelerometer and gyroscope from different participants who performed six distinct physical activities. Since KernelSHAP's computation time grows exponentially with the number of features, we performed the ablation test of KernelSHAP on HAR with a stratified sample of 400 observations to enable reproducibility of our results with limited hardware.

The Fraud Detection (Dal Pozzolo et al. 2014) data set contains benign and fraudulent credit card transactions. We used a processed version[2] of the data set containing the timestamp and amount of each transaction, along with 28 features generated by a Principal Component Analysis (PCA) of the original transaction details. Since Fraud Detection is a very large data set, we used a random sample of 50,000 observations to compute the experiment in reasonable time. Note that the random sample has approximately maintained the class imbalance of the full data set.

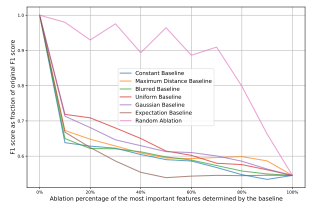
The Communities and Crime (Dua and Graff 2017) data set (in the following referred to as Communities) contains socio-economic and law enforcement data on communities in the USA. Our goal was to predict the amount of violent crimes per 100,000 inhabitants. To be precise, we have considered all communities with a crime rate of $>= 30\%$ as high risk and others as low risk. We then classified the observations according to these two labels. Note that we removed any feature that had more than 1,000 ($\approx 50\%$) missing values, in order to guarantee valid results.

The Spambase (Dua and Graff 2017) data set consists of information about genuine and spam emails.

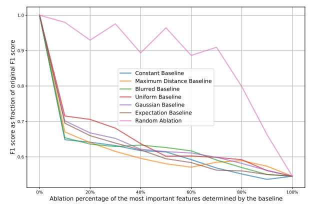
Finally, COMPAS (Angwin et al. 2016) is an algorithm that evaluates the risk of recidivism and is known to be biased against black defendants. The COMPAS data set contains the personal information of the defendants. For our experiments, we used a preprocessed version[3] of COMPAS.

[2]https://www.kaggle.com/mlg-ulb/creditcardfraud

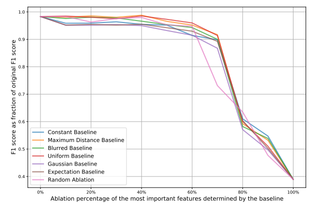
[3]https://www.kaggle.com/danofer/compass



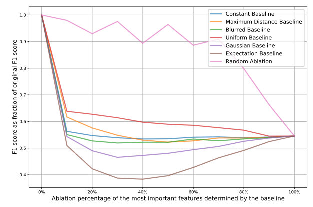
(a) DeepLift (Shrikumar, Greenside, and Kundaje 2017) on Human Activity Recognition



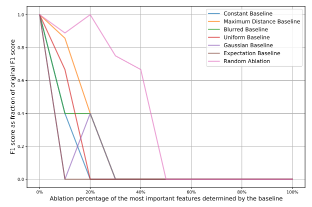
(b) IG (Sundararajan, Taly, and Yan 2017) on Human Activity Recognition



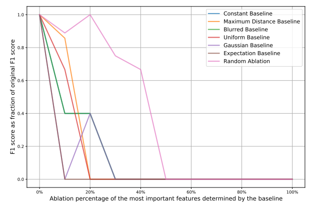
(c) KernelSHAP (Lundberg and Lee 2017) on Human Activity Recognition



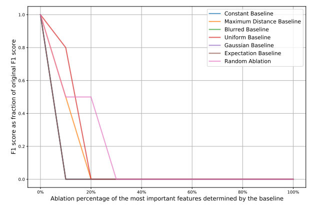
(d) DeepSHAP (Lundberg and Lee 2017) on Human Activity Recognition



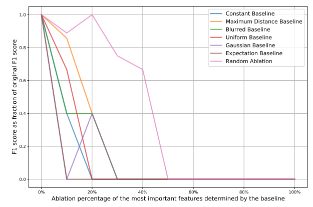
(e) DeepLift (Shrikumar, Greenside, and Kundaje 2017) on Fraud Detection



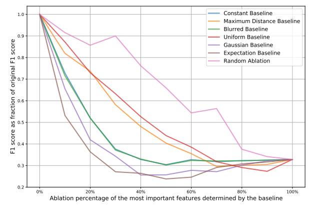
(f) IG (Sundararajan, Taly, and Yan 2017) on Fraud Detection



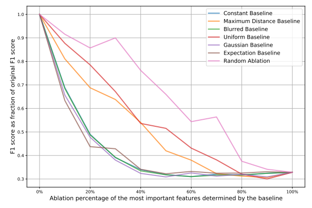
(g) KernelSHAP (Lundberg and Lee 2017) on Fraud Detection



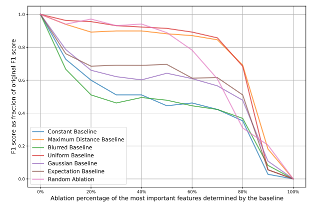
(h) DeepSHAP (Lundberg and Lee 2017) on Fraud Detection



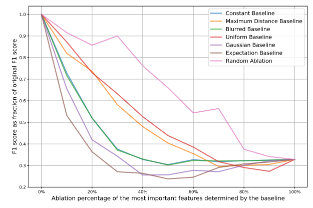
(i) DeepLift (Shrikumar, Greenside, and Kundaje 2017) on Communities



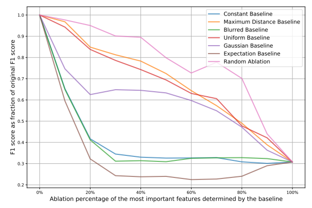
(j) IG (Sundararajan, Taly, and Yan 2017) on Communities



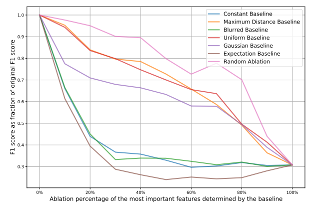
(k) KernelSHAP (Lundberg and Lee 2017) on Communities



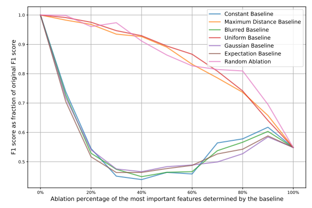
(l) DeepSHAP (Lundberg and Lee 2017) on Communities



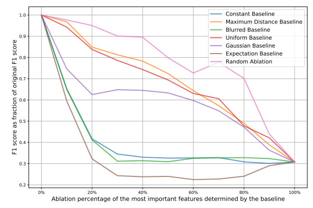
(m) DeepLift (Shrikumar, Greenside, and Kundaje 2017) on Spambase



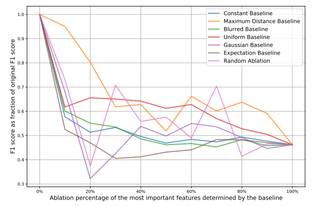
(n) IG (Sundararajan, Taly, and Yan 2017) on Spambase



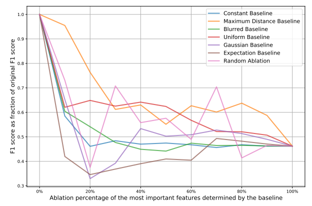
(o) KernelSHAP (Lundberg and Lee 2017) on Spambase



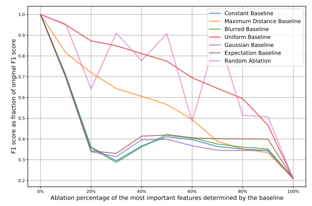
(p) DeepSHAP (Lundberg and Lee 2017) on Spambase



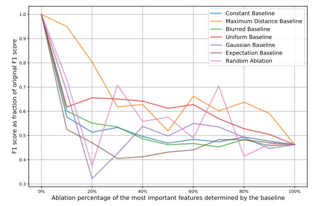
(q) DeepLift (Shrikumar, Greenside, and Kundaje 2017) on Compas



(r) IG (Sundararajan, Taly, and Yan 2017) on Compas

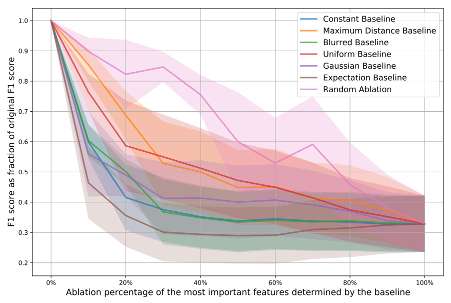


(s) KernelSHAP (Lundberg and Lee 2017) on Compas

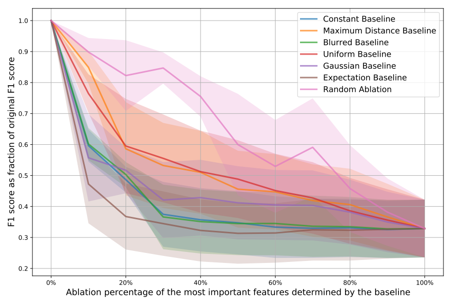


(t) DeepSHAP (Lundberg and Lee 2017) on Compas

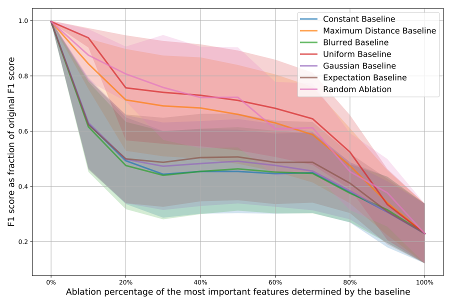
Figure 1: **Top K ablation tests:** We conducted ablation tests for every data set, attribution model and baseline method (subplots). One might have to zoom into the subplots. Specifically, we masked K percent (x-axis) of the most highly attributed features with random noise and monitored the change in the F1 score (y-axis). A large decrease in F1 indicates discriminative feature attributions with respect to the predictive model. Notably, there are significant differences between the baseline methods.



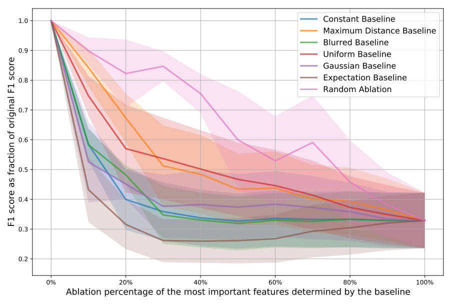
(a) DeepLift (Shrikumar, Greenside, and Kundaje 2017) on average over all data sets



(b) IG (Sundararajan, Taly, and Yan 2017) on average over all data sets



(c) KernelSHAP (Lundberg and Lee 2017) on average over all data sets



(d) DeepSHAP (Lundberg and Lee 2017) on average over all data sets

Figure 2: **Average ablation tests:** Here we depict the ablation test results averaged over all tabular data sets. The lines correspond to the mean decrease in F1 and the shaded areas are the respective standard deviation. Strikingly, we observe that the constant, blurred, Gaussian and expectation baseline produce competitive results, whereas the uniform and maximum distance baseline tend to generate less discriminative feature attributions.

Note that the blurred, Gaussian and uniform baseline assume continuity of features. Nevertheless, we decided to keep the categorical features of the COMPAS and Communities data set, since we did not perceive a significant change of the ablation tests compared to an evaluation without categorical features.

Results and Discussion

As described above, we performed ablation tests to measure the quality of the attributions on different baselines. The computation times of every baseline and attribution model can be found on our Github page.

Figure 1 exhibits the detailed results of every ablation test. Note that if a baseline were to produce discriminative attributions, we would see a sharp decline in F1 scores (since we mask the most important features). In general, we find that all attribution models were very sensitive to the different baselines. In the following, we discuss some of the most important findings. Please note again, that this experimental study is intended as a basis for a more in-depth analysis of baselines on tabular data. Accordingly, the following observations and hypotheses apply in the context of our experiment, but do not necessarily apply in general.

The maximum distance and uniform baselines often struggled to outperform the random baseline. This is in contrast to the results of Sturmfels, Lundberg, and Lee (2020), where the uniform baseline achieved competitive results in the ablation test on image data. A possible explanation is as follows: The uniform distribution might not be complex enough to approximate the data generating distribution of heterogeneous tabular data sufficiently well. As a result, the generated baseline values might lie outside the data generating distribution and therefore produce non-discriminative attributions. Likewise, the maximum distance baseline may not be representative of the data. In fact, the maximum distance baseline may return an outlier value.

On the other hand, the expectation and Gaussian baselines performed well in general. The expectation baseline closely approximates the data generating distribution, since it computes the expectation over a sample of training observations. Since many natural phenomena follow a normal distribution, we expect that the Gaussian baseline also approximates the data generating distribution sufficiently well. These results suggest that feature attributions become more discriminative, the closer a baseline follows the data generating distribution. We leave a detailed analysis for future work.

The constant and blurred baseline showed almost identical performance. Note that by applying a blur filter, we reduce the variance among input features. Accordingly, the blurred baseline approaches a constant value (low variance) the stronger we set the blur effect.

For the extremely imbalanced Fraud Detection data, all baselines performed equally well. Note that Fraud Detection comprises features which were generated in a Principal Component Analysis. Hence, by removing the features that correspond to the first principal components, we remove much of the discriminative information, which would explain the sharp drop in the predictive performance. However, the ablation test does not explicitly consider class imbalances. Hence, these results should be considered with care.

KernelSHAP values are theoretically optimal, but require an exponentially increasing number of feature coalition samples. As Lundberg, Erion, and Lee (2018) acknowledge, SHAP values can thus be challenging to compute. Accordingly, the performance of KernelSHAP decreases as the data dimensionality increases, if we can not adjust the number of coalition samples accordingly. This effect can be observed across all baseline methods both in the Communities data set and, to a greater extent, in the HAR data set. Note that the performance of KernelSHAP may be improved by running our experiments on more-advanced hardware.

In summary, our experiments illustrated that the expectation, blurred, constant and Gaussian baselines frequently produce discriminative attributions on tabular data. The uniform and maximum distance baselines generally perform worse in the ablation test, suggesting that they may not be a sensible choice in practice. The average ablation test results in Figure 2 support these findings. Still, other evaluation methods, such as randomization tests (Adebayo et al. 2018), could be considered in the future to substantiate our findings. As indicated in previous work (Sturmfels, Lundberg, and Lee 2020; Shrikumar, Greenside, and Kundaje 2017) and supported by our results, the appropriateness of a baseline method depends strongly on the data distribution at hand. Therefore, it might also be interesting to consider dynamic baseline methods that can be used even if the data generating distribution changes (Haug and Kasneci 2020). In general, we argue that a conceptual comparison of baseline methods is needed to enable more principled decisions in practice. In this context, the proposed taxonomy of baselines can be an important guideline.

Conclusion

In this work, we provided a first empirical comparison of common baseline methods for local attributions on tabular data sets. Additionally, we proposed a novel taxonomy of baseline methods. In this way, we complemented existing studies on image data. Our results show that the baseline can have a dramatic impact on the quality of generated feature attributions. In general, we argue that the selection and development of sensible baseline methods should receive more attention in research and practice.

References

Adebayo, J.; Gilmer, J.; Muelly, M.; Goodfellow, I.; Hardt, M.; and Kim, B. 2018. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, 9505–9515.

Angwin, J.; Larson, J.; Mattu, S.; and Kirchner, L. 2016. Machine Bias: There's software used across the country to predict future criminals. And it's biased against blacks. *ProPublica* .

Dal Pozzolo, A.; Caelen, O.; Le Borgne, Y.-A.; Waterschoot, S.; and Bontempi, G. 2014. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert systems with applications* 41(10): 4915–4928.

Dua, D.; and Graff, C. 2017. UCI Machine Learning Repository. URL http://archive.ics.uci.edu/ml.

Fong, R. C.; and Vedaldi, A. 2017. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, 3429–3437.

Haug, J.; and Kasneci, G. 2020. Learning Parameter Distributions to Detect Concept Drift in Data Streams. *arXiv preprint arXiv:2010.09388* .

Haug, J.; Pawelczyk, M.; Broelemann, K.; and Kasneci, G. 2020. Leveraging Model Inherent Variable Importance for Stable Online Feature Selection. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1478–1502.

Izzo, C.; Lipani, A.; Okhrati, R.; and Medda, F. 2020. A Baseline for Shapely Values in MLPs: from Missingness to Neutrality. *arXiv preprint arXiv:2006.04896* .

Kasneci, G.; and Gottron, T. 2016. Licon: A linear weighting scheme for the contribution of input variables in deep artificial neural networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 45–54.

Lundberg, S. M.; Erion, G. G.; and Lee, S.-I. 2018. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888* .

Lundberg, S. M.; and Lee, S.-I. 2017. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, 4765–4774.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. " Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 1135–1144.

Shrikumar, A.; Greenside, P.; and Kundaje, A. 2017. Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685* .

Smilkov, D.; Thorat, N.; Kim, B.; Viégas, F.; and Wattenberg, M. 2017. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825* .

Sturmfels, P.; Lundberg, S.; and Lee, S.-I. 2020. Visualizing the impact of feature attribution baselines. *Distill* 5(1): e22.

Sundararajan, M.; and Najmi, A. 2019. The many Shapley values for model explanation. *arXiv preprint arXiv:1908.08474* .

Sundararajan, M.; Taly, A.; and Yan, Q. 2017. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365* .

A.4 Dynamic Model Tree for Interpretable Data Stream Learning

Publication: Published in the proceedings of the 38th IEEE International Conference on Data Engineering (ICDE 2022).

Contribution: Inspired by an earlier work from Klaus Broelemann and Gjergji Kasneci, we had the idea of using Model Trees as an online learning method and alternative to Hoeffing Trees. I then formalized, implemented, and evaluated the Dynamic Model Tree. In addition, I wrote substantial parts of the paper. Klaus Broelemann contributed to a cleaner formalization, and Gjergji Kasneci helped me to clarify the two interpretability properties and polish the writing.

Dynamic Model Tree for Interpretable Data Stream Learning

Johannes Haug
University of Tuebingen
Tuebingen, Germany
johannes-christian.haug@uni-tuebingen.de

Klaus Broelemann
Schufa Holding AG
Wiesbaden, Germany
klaus.broelemann@schufa.de

Gjergji Kasneci
University of Tuebingen
Tuebingen, Germany
gjergji.kasneci@uni-tuebingen.de

***Abstract*—Data streams are ubiquitous in modern business and society. In practice, data streams may evolve over time and cannot be stored indefinitely. Effective and transparent machine learning on data streams is thus often challenging. Hoeffding Trees have emerged as a state-of-the art for online predictive modelling. They are easy to train and provide meaningful convergence guarantees under a stationary process. Yet, at the same time, Hoeffding Trees often require heuristic and costly extensions to adjust to distributional change, which may considerably impair their interpretability. In this work, we revisit Model Trees for machine learning in evolving data streams. Model Trees are able to maintain more flexible and locally robust representations of the active data concept, making them a natural fit for data stream applications. Our novel framework, called Dynamic Model Tree, satisfies desirable consistency and minimality properties. In experiments with synthetic and real-world tabular streaming data sets, we show that the proposed framework can drastically reduce the number of splits required by existing incremental decision trees. At the same time, our framework often outperforms state-of-the-art models in terms of predictive quality – especially when concept drift is involved. Dynamic Model Trees are thus a powerful online learning framework that contributes to more lightweight and interpretable machine learning in data streams.**

***Index Terms*—machine learning, data stream, model tree, concept drift, interpretability**

I. Introduction

Large-scale data streams are integral to most modern web-based applications such as online credit scoring, e-commerce or social media. Accordingly, the demand for powerful streaming machine learning models has increased. In practice, streaming or online learning models have to cope with limited hardware capacity and drifts of the data generating concept. Efficient, accurate and interpretable machine learning for evolving data streams is thus a major challenge.

Unlike traditional batch learning models, online learning models are updated incrementally. In this way, online learning models can be trained without the entire data set being available in main memory. Consequently, online learning models enable machine learning in practical applications that generate a potentially unlimited amount of data, e.g. large sensor systems or credit card transactions.

Online learning models usually have to cope with limited hardware capacity and drifts of the data generating concept. Changing customer preferences or emerging social media trends are prominent examples of such concept drift. In the worst case, concept drift may render previously learned concepts obsolete.

Accordingly, online learning models must provide discriminative predictions and adjust to concept drift, while reducing overall resource consumption. In addition, much attention has recently been paid to the interpretability of machine learning models [1], [2]. In particular, high-stakes applications and regulations (e.g. the EU General Data Protection Regulation GDPR) require models to be interpretable. For example, if a model is used to predict the risk of recidivism or the probability of a loan default, it can be crucial to be able to describe the model in understandable terms. However, compared to other domains such as image recognition [3], relatively little attention has been paid to the interpretability of machine learning models in evolving data streams. As one of the first works, we therefore briefly outline important aspects of interpretable online learning below.

A. On "Interpretability" in Evolving Data Streams

In general, we distinguish between post-hoc explainability and intrinsic interpretability [4]. The former concerns dedicated methods, e.g. local feature attributions [5]–[8], that allow to explain complex (black-box) models. Conversely, we speak of intrinsic interpretability when the internal mechanics of the predictive model are inherently understandable to a human.

Interpretability has varying domain-specific definitions [2] and cannot be measured in a standardized way [4]. Hence, interpretability is often represented by heuristic measures such as model size or complexity [9]. Intuitively, it is easier for humans to attribute meaning to individual model parameters when complexity is low. That is, the less complex a model is, the easier it is to interpret. For example, linear models and decision trees are typically considered highly interpretable. Specifically, the interpretability of linear models can be linked to their sparsity, i.e., the number of nonzero parameters. Similarly, the interpretability of decision trees can be quantified by the number of split nodes or the depth of the tree [10].

Since online learning models are incrementally updated, the parameters and model complexity can change between time steps. Therefore, in order to achieve interpretable online learning, we argue that it is not sufficient to deliver low complexity at each individual time step. Rather, changes in model complexity must also be comprehensible to humans.

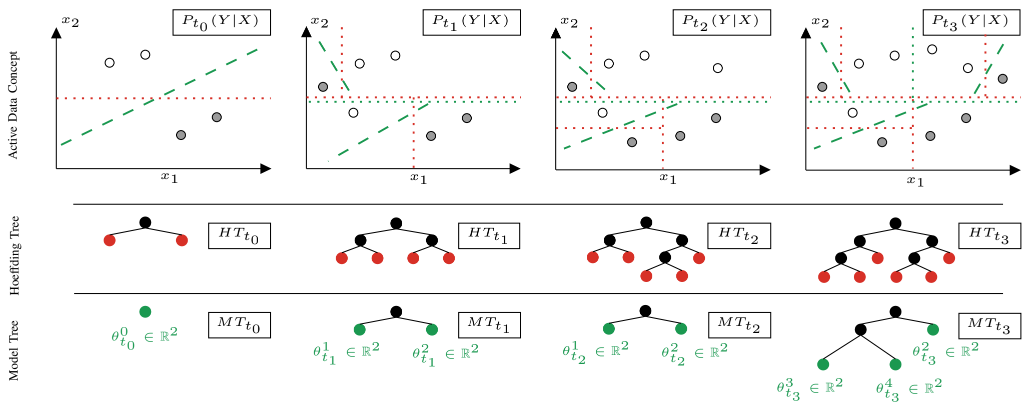

Active Data Concept
Hoeffding Tree
Model Tree
x_2
x_1
$P_{t_0}(Y|X)$
$P_{t_1}(Y|X)$
$P_{t_2}(Y|X)$
$P_{t_3}(Y|X)$
HT_{t_0}
HT_{t_1}
HT_{t_2}
HT_{t_3}
MT_{t_0}
MT_{t_1}
MT_{t_2}
MT_{t_3}
$\theta^0_{t_0} \in \mathbb{R}^2$
$\theta^1_{t_1} \in \mathbb{R}^2$
$\theta^2_{t_1} \in \mathbb{R}^2$
$\theta^1_{t_2} \in \mathbb{R}^2$
$\theta^2_{t_2} \in \mathbb{R}^2$
$\theta^2_{t_3} \in \mathbb{R}^2$
$\theta^3_{t_3} \in \mathbb{R}^2$
$\theta^4_{t_3} \in \mathbb{R}^2$

Fig. 1: **Hoeffding Tree vs. Model Tree:** A Hoeffding Tree (HT, red) aims for maximum node purity regarding the target (here we assume a binary target, i.e. white, grey circles). A Model Tree (MT, green), on the other hand, trains simple (linear) models at every leaf node (with parameters θ_t), and aims to minimize the estimated loss. Above, the dotted lines represent binary splits of the corresponding trees, and the dashed, green lines indicate the linear functions of the Model Tree. Model Trees are capable to maintain more flexible representations of the active data concept ($P_t(Y|X)$), while being more robust to small local variations. As a consequence, Model Trees tend to make more principled split decisions and remain shallower than Hoeffding Trees of similar predictive quality, as illustrated by this simplified example.

Ultimately, this requires that all updates to an interpretable online model are understandable. For example, the model should be able to answer questions like “Why have you removed this ensemble component at time step t?” or “Why have you split this node at time step u?”. In this sense, online interpretability is closely related to the robustness to noise and adaptability to concept drift. For example, model adaptations could be made understandable by linking them to changes of the (approximate) data concept or, ideally, corresponding events in the real world.

Although this discussion is certainly not exhaustive, it serves as a first guide for the development of inherently interpretable online machine learning methods. Note that a more formal definition of online interpretability is beyond the scope of this paper and is left for future work.

B. The State-Of-The-Art in Online Machine Learning

Incremental decision trees have emerged as the state-of-the-art for online machine learning. The Hoeffding Tree is one of the most prominent frameworks. Hoeffding Trees use Hoeffding’s inequality to decide at which time step, i.e. after how many observations, a leaf node will be split [11]–[14]. A Hoeffding Tree comes asymptotically arbitrarily close to a hypothetical, batch-trained decision tree, given that the data generating process is stationary. Similar to batch-trained decision trees, Hoeffding Trees benefit from high efficiency and transparency.

However, the basic Hoeffding Tree algorithm, VFDT [11], may grow indefinitely. This behaviour can considerably impair the performance of the VFDT and – in the above sense – its interpretability. In general, such infinite growth can be avoided, e.g. by extending the Hoeffding Tree with dedicated drift detection strategies [13]. However, such extensions often increase the complexity and make split or prune decisions less intuitive. Moreover, Hoeffding Trees suffer practical limitations. For example, the way in which Hoeffding’s inequality and heuristic purity measures are used within the framework has been repeatedly questioned [15]–[17].

C. Model Trees As An Alternative to Hoeffding Trees

In this work, we revisit Model Trees as an alternative to Hoeffding Trees [18]–[21]. Model Trees have much in common with regular decision trees, but contain simple predictive models in place of each (leaf) node. Hence, similar to an ensemble, Model Trees are a collection of weak learners that are combined in a structured way through a set of binary decisions. However, unlike Hierarchical Mixtures of Experts, Model Trees use only a single feature to split at each inner node. Accordingly, Model Trees preserve much of the simplicity of a regular decision tree.

Owing to the simple models, Model Trees are able to apply a less rigid separation of observations in the leaf nodes. In this way, Model Trees are generally more flexible regarding the active data concept than existing frameworks like the Hoeffding Tree (see Figure 1). In particular, Model Trees can represent linear relationships with only a few splits. Hence, Model Trees can usually achieve high predictive quality while using a simple and robust representation.

Replacing regular leaf nodes with simple models in an otherwise unmodified tree increases complexity. However, Model Trees often remain extremely shallow and thus interpretable, as we show in experiments. In addition, unlike Hoeffding Trees, Model Trees allow feature weights for different subgroups to be extracted directly from the simple models. In comparison to majority weighting schemes, this can be an advantage for local feature-based explanations.

D. Our Contribution

In this paper, we introduce a novel online learning framework called *Dynamic Model Tree*. We show that the simple models of a Model Tree can be leveraged to define node-specific gain functions. These gain functions guarantee sensible consistency and minimality properties, which contribute to more intuitive and interpretable online learning. Compared to existing state-of-the-art methods such as Hoeffding Trees or earlier incremental Model Trees [20], [21], Dynamic Model Trees adapt to concept drift by design. In particular, the proposed framework does not rely on Hoeffding's inequality, heuristic purity measures or explicit concept drift detection mechanisms. Consequently, the Dynamic Model Tree eliminates some of the most fundamental weaknesses of existing online decision trees.

In summary, the contributions of this work are as follows:

- We specify valuable properties related to the consistency and minimality of incremental decision trees (Section III). Combined, these properties lead to more interpretable online learning as described above.
- We introduce the *Dynamic Model Tree* framework (Section IV). In particular, we define generic gain functions that guarantee the above-mentioned properties and can be efficiently approximated via gradients [19].
- We propose an effective implementation of the Dynamic Model Tree that uses Generalized Linear Models and the negative log-likelihood loss (Section V).
- We evaluate the Dynamic Model Tree on multiple synthetic and real-world tabular data sets with different types of concept drift (Section VI). While maintaining high efficiency, our implementation often outperforms existing classifiers in terms of predictive quality and complexity.

II. Related Work

Incremental decision trees are a powerful class of online predictors. In the following, we briefly outline state-of-the-art algorithms based on the Hoeffding Tree, along with their limitations. Moreover, we discuss previous works on incremental Model Trees. For more information about online learning, we refer to recent surveys [22]–[24].

A. Variations of the Hoeffding Tree

The Very Fast Decision Tree (VFDT) is the first and basic implementation of a Hoeffding Tree [11]. As mentioned above, it has practical limitations. In particular, the VFDT assumes that a relatively small set of past and current observations is representative of all future observations – a misconception under realistic streaming conditions. Accordingly, the VFDT grows indefinitely and does not revisit old split decisions, which can impair its interpretability.

Most of the limitations of the basic VFDT can be overcome, e.g. by using regularization [25], different probabilistic inequalities [15], gain measures [16], [17] or tricks in the implementation [26]. To increase the predictive performance under concept drift, the Hoeffding Tree may also be augmented with adaptation strategies like alternate tree growth [12], sliding windows [13] or a dynamic replacement of inner nodes [14]. In addition, ensembles of Hoeffding Trees, e.g. Bagging or Boosting [27], [28], can increase the predictive performance of the basic models at the cost of higher overall complexity.

The Hoeffding Tree has gained popularity due to its rigorous convergence guarantees, efficiency, extensibility and accessibility via packages like MOA [29] or scikit-multiflow [30]. However, the inherent limitations of the basic architecture may ultimately leave users in doubt about the reliability of the Hoeffding Tree. Hence, we argue that a different framework is needed, which offers a similar level of efficiency and extensibility, but is more flexible and interpretable in a dynamic online environment.

B. Incremental Model Trees

Hoeffding Trees have been augmented with simple models, such as Naïve Bayes [31] and Perceptrons [32]. Such extensions often provide considerable improvements in predictive performance compared to majority-weighted leaves. Surprisingly, however, the more general family of Model Trees has received only little attention in online learning scenarios. Notable exceptions include the work by [20], which is aimed at stationary applications, and the FIMT-DD model [21]. FIMT-DD was introduced as a solution for online regression tasks. Similar to Hoeffding Trees, FIMT-DD applies Hoeffding's inequality to split at the inner nodes. Specifically, FIMT-DD aims to find the split that gives the largest reduction in the standard deviation of the target variable. To avoid infinite growth, FIMT-DD employs explicit concept drift detection via the Page-Hinkley test and offers various adaptation strategies. The FIMT-DD model and the Dynamic Model Tree proposed in this work have fundamental differences, which we outline in Section V.

III. Preliminaries and Properties for Online Decision Tree Learning

A data stream can be represented by a potentially infinite series of time steps $1,..,t,..,T$. Let $X_t \in \mathbb{R}^{n_t \times m}$ be the matrix of observations at time step t, where $n_t \geq 1$ is the number of observations and $m \geq 1$ is the number of features. We denote $Y_t \in \mathbb{R}^{n_t}$ the corresponding labels at time step t. The observations and labels are drawn from a distribution $P_t(X, Y)$, which we call the active concept at time step t. Concept drift is defined as a change in the active concept between two time steps, i.e. $P_{t_1}(X, Y) \neq P_{t_2}(X, Y)$.

Suppose that an incremental decision tree is parameterized by Θ_t at time step t. We assume that the parameters Θ_t are given by the context and therefore leave them unspecified. As described in the introduction, we generally aim for models that are discriminative and interpretable. Given our understanding of interpretable online learning and the example in Figure 1, we argue that for equal predictive power, the smaller tree should be preferred. In this context, we identify two crucial properties for training incremental decision trees.

Let Ω_t be a set of time indices up to time step t. Let $X_{\Omega_t}, Y_{\Omega_t}$ be sets of corresponding observations and labels and, for simplicity, let $L(\Omega_t)$ be the shorthand notation for $L(\Theta_{\Omega_t}, Y_{\Omega_t}, X_{\Omega_t})$, which denotes the estimated loss of an incremental decision tree with respect to Ω_t. As before, we assume that the parameters Θ_{Ω_t} are given by the context.

Property 1 (Consistency with Parent Splits)**.** Suppose we perform a split at time step t. Let $L_C(\Omega_t)$ be the new estimated loss after the split. An incremental decision tree algorithm is *consistent with parent splits* regarding the set Ω_t, if $L_C(\Omega_t) \leq L(\Omega_t)$.

Accordingly, we must avoid splits that would increase the estimated loss. This property primarily concerns the predictive quality of the obtained tree and is a common objective. Additionally, by choosing an adequate loss function that approximates the active data concept (which we discuss in Section V), Property 1 enables interpretable split decisions.

With the goal of low model complexity, i.e. high interpretability, we add a second property:

Property 2 (Model Minimality)**.** Suppose there exists a subtree of the incremental decision tree at time step t, whose loss is denoted by $L_{alt}(\Omega_t)$. An incremental decision tree algorithm preserves *model minimality* regarding the set Ω_t, if for $L(\Omega_t) = L_{alt}(\Omega_t)$ it retains the tree with fewer number of parameters.

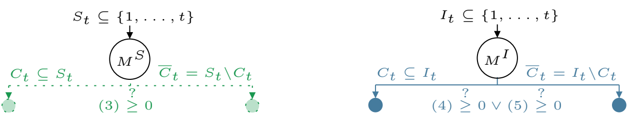
Hence, we are bound to replace a complex tree, whenever it contains a simpler subtree that has equal predictive quality regarding Ω_t. For practical purposes, this means that we have to prune or replace nodes or branches of the tree that no longer improve the estimated loss (since the number of parameters per node is usually fixed). Consequently, Property 2 also implies a mechanism to adapt to concept drift.

IV. Dynamic Model Tree

In this paper, we extend Model Trees to a novel framework for adaptive predictive modelling in dynamic data streams that adheres to the aforementioned properties.

A Dynamic Model Tree is constructed in a similar fashion as regular decision trees. That is, we begin with a single root node and gradually grow and prune the tree over time. Each node of a Dynamic Model Tree can be represented by a set of time indices $S_t \subseteq \{1, \dots, t\}$ corresponding to the observations that have reached the node up to time step t. Other than existing Model Trees, a Dynamic Model Tree maintains simple predictive models at both leaf and inner nodes (see Figure 2). These models are used to identify optimal split candidates (i.e., feature-value combinations) and make predictions. Let X_{S_t}, Y_{S_t} be the observations and labels, and Θ_{S_t} the parameters of the simple model at a node corresponding to the time indices in S_t. We aim to find the parameters that minimize a loss function $L(\cdot) \geq 0$:

$$\Theta^*_{S_t} = \underset{\Theta_{S_t}}{\arg\min}\ L(\Theta_{S_t}, Y_{S_t}, X_{S_t}) \\ = \underset{\Theta_{S_t}}{\arg\min}\ \sum_{t \in S_t} L(\theta_t, Y_t, X_t) \qquad (1)$$


$S_t \subseteq \{1, \dots, t\}$
M^S
$C_t \subseteq S_t$
$\overline{C}_t = S_t \backslash C_t$
$(3) \overset{?}{\geq} 0$
$I_t \subseteq \{1, \dots, t\}$
M^I
$C_t \subseteq I_t$
$\overline{C}_t = I_t \backslash C_t$
$(4) \overset{?}{\geq} 0 \vee (5) \overset{?}{\geq} 0$

(a) Leaf Node (b) Inner Node

Fig. 2: **DMT Nodes:** Both inner and leaf nodes of a Dynamic Model Tree contain simple models M that are incrementally trained during a subset of time steps S_t and I_t, respectively. At every time step t, we check at the leaf nodes whether there is a new split candidate with positive gain (3) (green, see also Algorithm 1). Similarly, we check at the inner nodes whether the gains (4) or (5) are positive, i.e., whether we must replace the current split (blue) and thus prune the old branch.

We assume independence between time steps; a simplifying, yet common assumption in data stream learning that has been shown to work well in practice. Accordingly, we can update the parameters θ_t independently at every time step using gradient descent. The optimal parameters from the previous time step can be used as prior parameters at time step t. Accordingly, at every time step, we forward incoming observations to a corresponding leaf node, updating each simple model along the path. Once we have updated all relevant simple models, we attempt to grow or prune the Dynamic Model Tree. To this end, we require gain measures that account for the aspired *consistency with parent splits* and *model minimality*.

A. Loss-Based Gain Functions

Typically, decision tree algorithms aim for maximum node purity with respect to the target variable. For this purpose, split decisions are usually based on heuristic purity measures such as the Information Gain or the Gini index. However, the simple models of a Dynamic Model Tree offer a fundamental advantage in terms of the proposed properties. Instead of relying on heuristic measures, we may directly select the split candidate that reduces the overall loss of our tree. Consequently, any update of the model complexity can be directly linked to a change in the loss, providing better interpretability as described in Section I-A.

Suppose we are at a leaf node of the tree. Let S_t be the corresponding set of time indices observed at this leaf node. Our goal now is to find a new split candidate, i.e., a feature-value pair, to further split the observations. We can represent each split candidate by a set of time indices that would have been passed to the left child $C_t \subseteq S_t$ and the right child $\bar{C}_t = S_t \backslash C_t$. For the sake of illustration, we assume binary splits. However, our exposition can readily be extended to non-binary trees. Our goal is to select the split candidate that maximizes the improvement of the current loss:

$$C^*_t = \underset{C_t}{\arg\max}\ G_{S_t, C_t}, \text{ with} \qquad (2)$$

$$G_{S_t, C_t} = L(\Theta_{S_t}, Y_{S_t}, X_{S_t}) \\ - L(\Theta_{C_t}, Y_{C_t}, X_{C_t}) - L(\Theta_{\bar{C}_t}, Y_{\bar{C}_t}, X_{\bar{C}_t}) \qquad (3)$$

With (3), the proof of *consistency to parent splits* is almost trivial:

Lemma 1. Every new split with a gain $G_{S_t,C_t} \geq 0$ due to (3) implies *consistency with parent splits* (Property 1).

Proof. The loss of a Dynamic Model Tree at time step t corresponds to the sum of losses at each leaf node, i.e. $L(\Omega_t) = \sum_{J_t \subseteq \Omega_t} L(\Theta_{J_t}, Y_{J_t}, X_{J_t})$, where every set J_t represents a leaf node. Suppose there exists a leaf node S_t, such that $G_{S_t,C_t} \geq 0$ for some split candidate C_t. A split on C_t corresponds to a new loss $L_C(\Omega_t) = L(\Omega_t) - G_{S_t,C_t}$, which implies $L_C(\Omega_t) \leq L(\Omega_t)$. □

To satisfy *model minimality* (Property 2), we also need to evaluate existing splits of the Dynamic Model Tree. Specifically, we may replace an existing inner node with either a new split candidate or a leaf node. In both cases, we would prune the old branch (subtree). Suppose there is a subtree whose root corresponds to an inner node of the original tree. As before, we represent this inner node by a set of time indices I_t. Likewise, each leaf node of the subtree is represented by a set J_t, such that the union of all J_t is equal to I_t. We then try to find an alternate split candidate (represented by $C_t \subseteq I_t$, $\bar{C}_t = I_t \backslash C_t$), i.e. a substitute for the inner node I_t, which offers an improvement in terms of the loss:

$$G_{I_t,C_t} = \sum_{J_t \subseteq I_t} L(\Theta_{J_t}, Y_{J_t}, X_{J_t}) - L(\Theta_{C_t}, Y_{C_t}, X_{C_t}) - L(\Theta_{\bar{C}_t}, Y_{\bar{C}_t}, X_{\bar{C}_t}) \quad (4)$$

If the gain (4) is positive, we can prune the old subtree and add a new inner node with two new leaf nodes in its place. Alternatively, we may make the current inner node a leaf. To this end, we need to compare the loss at the inner node with the loss of the current subtree. The corresponding gain is

$$G_{I_t} = \sum_{J_t \subseteq I_t} L(\Theta_{J_t}, Y_{J_t}, X_{J_t}) - L(\Theta_{I_t}, Y_{I_t}, X_{I_t}). \quad (5)$$

If both gains (4) and (5) are positive and $G_{I_t} \geq G_{I_t,C_t}$, we apply the second option, replacing the inner node with a leaf node, to obtain the overall smaller tree. Notably, (4) and (5) allow us to maintain the minimality of a Dynamic Model Tree:

Lemma 2. Greedy replacement of inner nodes, wherever $G_{I_t,C_t} \geq 0$ due to (4) or $G_{I_t} \geq 0$ due to (5), implies *model minimality* (Property 2).

Proof. Let I_t represent an inner node of the Dynamic Model Tree. There exists a subtree whose root is the inner node I_t. We may prune this subtree by replacing the inner node I_t with a different split candidate or a leaf. The gain G corresponds to (4) or (5) respectively. Accordingly, $L_{alt}(\Omega_t) = L(\Omega_t) - G$ is the loss of the potential alternate tree with the subtree replaced. Note that the alternate tree is guaranteed to have an equal or lower number of nodes and, since the number of parameters per node is fixed, an equal or lower number of parameters. Since $L_{alt}(\Omega_t) = L(\Omega_t)$ implies that $G = 0$, by assumption we would replace the Dynamic Model Tree by the alternate tree with the smaller number of parameters. This procedure may be repeated from the bottom to the root of the tree. □

B. Candidate Loss Approximation

To compare the gains (3) or (4) of different split candidates, we require loss estimates $L(\Theta_{C_t}, Y_{C_t}, X_{C_t})$ and $L(\Theta_{\bar{C}_t}, Y_{\bar{C}_t}, X_{\bar{C}_t})$ for each candidate. However, due to limited resources, we usually cannot train the simple models corresponding to every potential split candidate. For this purpose, we adopt an efficient gradient-based approximation.

The authors in [19] argue that we may warm-start optimizing the parameters of a split candidate Θ_{C_t} with a single gradient step on the parameters of the current node Θ_{S_t}:

$$\Theta_{C_t} \approx \Theta_{S_t} - \frac{\lambda}{|C_t|} \nabla_{\Theta_{S_t}} L(\Theta_{S_t}, Y_{C_t}, X_{C_t}) \quad (6)$$

The first order Taylor polynomial at the point Θ_{S_t} then gives a good approximation to the loss of the split candidate $L(\Theta_{C_t}, Y_{C_t}, X_{C_t})$. Accordingly, we write

$$\begin{aligned} L(\Theta, Y_{C_t}, X_{C_t}) &\approx L(\Theta_{S_t}, Y_{C_t}, X_{C_t}) \\ &\quad + (\Theta - \Theta_{S_t})^T \nabla_{\Theta_{S_t}} L(\Theta_{S_t}, Y_{C_t}, X_{C_t}) \\ \overset{(6)}{\Rightarrow} L(\Theta_{C_t}, Y_{C_t}, X_{C_t}) &\approx L(\Theta_{S_t}, Y_{C_t}, X_{C_t}) \\ &\quad - \frac{\lambda}{|C_t|} \|\nabla_{\Theta_{S_t}} L(\Theta_{S_t}, Y_{C_t}, X_{C_t})\|_2^2. \end{aligned} \quad (7)$$

With (7) we can approximate the loss of different split candidates without maintaining corresponding simple models. Moreover, we can reuse the gradient calculated during the optimisation of the parent model, which further increases efficiency. Finally, note that other work has successfully used gradient-based split finding [33].

C. Basic Algorithm And Complexity

Algorithm 1 depicts the general procedure at a leaf node of the Dynamic Model Tree. For inner nodes, we compute the gain functions (4) and (5) in line 12. In line 19, we then replace the inner node with a new split or a leaf (depending on which gain is greater). Otherwise, the general update procedure is equivalent for both types of nodes. We update the nodes of the tree in a bottom-up fashion.

The time complexity of Algorithm 1 for updating one node without fitting the simple model is $\mathcal{O}(mn_tc + m^2vc)$, where c is the number of classes, m is the number of features, n_t is the sample size at time step t and v is the maximal number of unique values of a feature. Depending on the choice of simple model, the time complexity might increase. If the maximal number of unique values is large, i.e. $v \gg n_t$, then the first term becomes negligible, leading to a complexity of $\mathcal{O}(m^2vc)$. In practice, decision tree algorithms often reduce computation time by limiting the number of eligible split candidates. This can be particularly important when we deal with large numbers of (continuous) features. We propose a simple method in Section V.

The memory complexity per node of the Dynamic Model Tree is $\mathcal{O}(m^2vc)$. As before, the memory requirements of the

Algorithm 1 Dynamic Model Tree - General Update Procedure at a Leaf Node at Time Step t

```
Input: Observations and labels $X_t, Y_t$; Simple model $M_{\theta_t}$;
    Likelihoods, gradients and counts of time step $t-1$.
Output: Updated likelihoods, gradients and counts.
    *** Increment the loss, gradient and count at the node. ***
 1: $L(\Theta_{S_t}, Y_{S_t}, X_{S_t}) \leftarrow$
        $L(\Theta_{S_{t-1}}, Y_{S_{t-1}}, X_{S_{t-1}}) + L(\theta_t, Y_t, X_t)$
 2: $\nabla_{\Theta_{S_t}} L(\Theta_{S_t}, Y_{S_t}, X_{S_t}) \leftarrow$
        $\nabla_{\Theta_{S_{t-1}}} L(\Theta_{S_{t-1}}, Y_{S_{t-1}}, X_{S_{t-1}}) + \nabla_{\theta_t} L(\theta_t, Y_t, X_t)$
 3: $n_{S_t} \leftarrow n_{S_{t-1}} + len(Y_t)$
    *** Update the statistics of split candidates and compute the gains (NOTE: The right child statistics corresponding to the set $\bar{C}_t$ can be obtained as the difference between the statistics of the left child ($C_t$) and the parent node ($S_t$). They therefore do not need to be stored separately.)- ***
 4: $G_{\max} \leftarrow -1$
 5: $C_{\text{top}} \leftarrow$ None
 6: for all split candidates $C$ do
 7:     $Y_t^C \subseteq Y_t$; $X_t^C \subseteq X_t$
 8:     $L(\Theta_{S_t}, Y_{C_t}, X_{C_t}) \leftarrow$
            $L(\Theta_{S_{t-1}}, Y_{C_{t-1}}, X_{C_{t-1}}) + L(\theta_t, Y_t^C, X_t^C)$
 9:     $\nabla_{\Theta_{S_t}} L(\Theta_{S_t}, Y_{C_t}, X_{C_t}) \leftarrow$
            $\nabla_{\Theta_{S_{t-1}}} L(\Theta_{S_{t-1}}, Y_{C_{t-1}}, X_{C_{t-1}}) + \nabla_{\theta_t} L(\theta_t, Y_t^C, X_t^C)$
10:     $n_{C_t} \leftarrow n_{C_{t-1}} + len(Y_t^C)$

11:     $L(\Theta_{C_t}, Y_{C_t}, X_{C_t}) \leftarrow$ (7)
12:     $G_{S_t, C_t} \leftarrow$ (3)
13:     if ($G_{S_t, C_t} > G_{\max}$) then
14:         $G_{\max} \leftarrow G_{S_t, C_t}$
15:         $C_{\text{top}} \leftarrow C$
16:     end if
17: end for
    *** Split or retain the leaf node. ***
18: if $G_{\max} \geq 0$ then
19:     Split on candidate $C_{\text{top}}$
20: end if
```

Dynamic Model Tree scale with the number of split candidates considered.

D. Differences Between DMT and Earlier Methods

Dynamic Model Trees differ clearly from earlier work. A major difference lies in the way Dynamic Model Trees handle concept drift. While purity-based adaptation strategies usually require dedicated drift detection models to identify concept drift [34], a Dynamic Model Tree does not. In fact, adaptation to concept drift is automatically handled via the proposed gain functions. As a consequence, Dynamic Model Trees only have few hyperparameters that need to be optimized, while providing a similar level of flexibility as earlier works.

FIMT-DD is one of the most popular existing Model Tree frameworks for data streams [21]. In the following, we briefly highlight key differences between FIMT-DD and the Dynamic Model Tree. Like a Hoeffding Tree, FIMT-DD relies on a purity measure (Standard Deviation Reduction) and Hoeffding's inequality to compare split candidates. That is, in FIMT-DD, "the process of learning linear models in the leaves will not explicitly reduce the size of the (...) tree" [21]. In addition, FIMT-DD requires a dedicated concept drift detection method (Page Hinkley) to adapt to change. As mentioned before, the Dynamic Model Tree neither requires a heuristic measure nor a separate concept drift detection model.

Other than FIMT-DD, the Dynamic Model Tree continues to update the simple models at the inner nodes even after splitting. This may increase the computation time, but allows us to compute the loss concerning the active concept on different hierarchies. In this way, the proposed framework can effectively identify and adjust to global and local concept drift.

E. Limitations

Typically, incremental decision trees like VFDT [11] or FIMT-DD [21] primarily occupy memory for saving statistics in the leaf nodes. Dynamic Model Trees also require memory to store statistics for every inner node. For example, while VFDT occupies $\mathcal{O}(lmvc)$ memory, a Dynamic Model Tree requires $\mathcal{O}((l+i)m^2vc)$, where l and i are the number of leaf and inner nodes, m is the number of features, v is the maximal number of unique values per feature and c is the number of classes. However, Dynamic Model Trees usually remain shallow due to the *model minimality* property, which reduces the overall computational gap to other methods.

Likewise, Dynamic Model Trees can have a longer training time per node, depending on the selected simple model type. The choice of appropriate simple models also affects the general performance of the tree. With random initial weights, a simple model may take some time to achieve good predictive quality. However, this mainly affects the root node of the Dynamic Model Tree, since all other simple models are warm-started with the optimized parameters of the parent node. In addition, if the simple models are non-robust or biased, the split and prune decisions of the proposed framework will suffer. In general, however, inadequate model types can be quickly identified by comparing the predictive error to benchmarks (e.g. the VFDT).

V. Implementation

The Dynamic Model Tree offers a large degree of flexibility. In particular, our framework may be implemented with different simple models and loss functions to account for different applications. For illustration, we propose an effective implementation of the Dynamic Model Tree for binary and multi-class classification.

A. Simple Models

We use logit and multinomial logit models (softmax) to represent binary and categorical target variables, respectively. Both models belong to the family of Generalized Linear Models (GLM) and are widely used in practice due to their efficiency and transparency. We train the simple models by stochastic gradient descent with a constant learning rate. In the future, one might experiment with different base models, optimization strategies or online feature selection [35].

B. Loss Function

Owing to the proposed gain functions, changes in a Dynamic Model Tree are directly linked to changes in the empirical loss. Although purity-based splits usually also lead to a reduction in error, splits based on a change in loss can be very powerful in terms of interpretability.

To this end, we recall that concept drift between two time steps t_1 and t_2 corresponds to a change in the active concept, i.e. $P_{t_1}(X,Y) \neq P_{t_2}(X,Y)$. Online learning models need to adjust to concept drift in order to maintain high predictive performance. Accordingly, we are mainly interested in concept drift that shifts the optimal decision boundary. This form of concept drift is called real concept drift and is defined as $P_{t_1}(Y|X) \neq P_{t_2}(Y|X)$ [36]. Since the true distribution $P_t(Y|X)$ is generally unknown, our best approximation of the active data concept is the likelihood $P(Y_t|X_t,\theta_t)$ [34]. In this context, the negative log-likelihood $L(\Theta_{S_t}, Y_{S_t}, X_{S_t}) = -\sum_{t\in S_t} \log P(Y_t|X_t,\theta_t)$ is a straight-forward choice for the loss function.

If a simple model performs well, we can generally assume that the likelihood is a good approximation of the data-generating concept. Accordingly, we may assume that the negative log-likelihood loss changes as a consequence of concept drift. For this reason, the negative log-likelihood loss allows us to associate any (major) change in the gains (3)-(5) with a local change in the approximate data concept. Compared to popular purity measures, this enables a much higher degree of online interpretability, as discussed in the introduction.

C. Threshold for Robust Model Updates

In practice, an online learning model will be subject to small variations and noise. It may therefore be useful to specify a threshold on the gain functions to avoid excessive updates.

If we set a threshold for the gains defined in (4) and (5), we need to relax the *model minimality* (Property 2): We recall that the loss of a minimal alternate model is given by $L_{alt}(\Omega_t) = L(\Omega_t) - G$, where $L(\Omega_t)$ is the loss of the current tree and G corresponds to (4) or (5) (see Lemma 2 and Proof). Consequently, if we prune the inner node whenever $G \geq threshold \geq 0$, we retain the minimal model for $L_{alt}(\Omega_t) \leq L(\Omega_t) - threshold$. This relaxation can sometimes be sensible, since a non-robust tree may be equally undesirable than an overly complex tree. Besides, if the threshold is reasonably small, changes of the loss due to concept drift will usually trigger model updates after a few iterations. To set a threshold, one only needs to adjust line 18 of the basic procedure shown in Algorithm 1.

By using the negative log-likelihood loss, we enable a natural threshold in terms of the Akaike Information Criterion:

$$AIC = 2k - 2\ell(\Theta), \tag{8}$$

where ℓ is the log-likelihood and k is the number of free (estimated) parameters. The AIC is a popular test statistic for model selection problems. It estimates the relative amount of information lost among competing models. Given two models i and j where $AIC_i \leq AIC_j$, the quantity $\exp([AIC_i - AIC_j]/2)$ is proportional to the relative probability that model j minimizes the estimated information loss. Therefore, if we set a threshold for this quantity, we can control the tolerated probability that model j actually has the minimum AIC instead of model i.

We can apply this methodology to our split and prune strategy. For example, when attempting to split, we compare the simple models representing the current node (S_t) and the potential split (C_t, $\bar{C}_t$). The corresponding AICs are

$$\begin{aligned} AIC_{S_t} &= 2k_{S_t} - 2\ell(\Theta_{S_t}, Y_{S_t}, X_{S_t}), && (9)\\ AIC_{C_t} &= 2(k_{C_t} + k_{\bar{C}_t}) \\ &\quad - 2\big(\ell(\Theta_{C_t}, Y_{C_t}, X_{C_t}) + \ell(\Theta_{\bar{C}_t}, Y_{\bar{C}_t}, X_{\bar{C}_t})\big), && (10) \end{aligned}$$

where k_{S_t}, k_{C_t} and $k_{\bar{C}_t}$ denote the numbers of free parameters of the corresponding models. Let $\epsilon \in [0,1]$ be a user-specified hyperparameter. We apply the following test:

$$\begin{aligned} & \exp([AIC_{C_t} - AIC_{S_t}]/2) \leq \epsilon \\ \Leftrightarrow\ & \exp\big(k_{C_t} + k_{\bar{C}_t} - \ell(\Theta_{C_t}, Y_{C_t}, X_{C_t}) - \ell(\Theta_{\bar{C}_t}, Y_{\bar{C}_t}, X_{\bar{C}_t}) \\ & \qquad - k_{S_t} + \ell(\Theta_{S_t}, Y_{S_t}, X_{S_t})\big) \leq \epsilon \\ \overset{(3)}{\Leftrightarrow}\ & \exp\big(k_{C_t} + k_{\bar{C}_t} - k_{S_t} - G_{S_t,C_t}\big) \leq \epsilon \\ \Leftrightarrow\ & \exp(-G_{S_t,C_t}) \leq \frac{\epsilon}{\exp(k_{C_t} + k_{\bar{C}_t} - k_{S_t})} \\ \Leftrightarrow\ & -G_{S_t,C_t} \leq \log(\epsilon) - k_{C_t} - k_{\bar{C}_t} + k_{S_t} \\ \Leftrightarrow\ & G_{S_t,C_t} \geq k_{C_t} + k_{\bar{C}_t} - k_{S_t} - \log(\epsilon) && (11) \end{aligned}$$

If we use the same simple model type at every node (e.g. logit models as proposed earlier), then (11) simplifies to $G_{S_t,C_t} \geq k - \log(\epsilon)$. Similarly, we can calculate thresholds for the remaining gain functions, which we omit for brevity. Note that the hyperparameter ϵ controls the trade-off between quick and robust updates. In this way, we can adjust the sensitivity of the Dynamic Model Tree.

D. Algorithmic Considerations

We implemented the Dynamic Model Tree in Python.[1] Note that the Dynamic Model Tree is able to handle both batch-incremental and instance-incremental online learning. In the following, we discuss important algorithmic details and propose a sensible hyperparameter configuration.

In practice, the number of unique split candidates may grow quickly – in particular for continuous variables. This is a problem that most incremental decision trees have in common. To overcome potential memory overload, our framework may be extended with advanced strategies like Binary Search Trees (see their application in FIMT-DD [21], for example). For illustration, however, we have chosen a simpler technique.

[1]https://github.com/haugjo/dynamic-model-tree

TABLE I: *Data sets*. We used state-of-the-art tabular streaming data sets with different types of concept drift. TüEyeQ [37], as well as Insects-Abrupt and Insects-Incremental [38] have been obtained from the sources referenced in the respective papers. The remaining real-world data sets have been obtained from *https://www.openml.org*. We included the original reference wherever available. The synthetic data sets have been generated with *scikit-multiflow* [30]. Here we also indicate the type of concept drift generated (abrupt or incremental).

Name	**#Samples**	**#Features**	**#Classes (#Majority)**
Electricity	45,312	8	2 – (26,075)
Airlines	539,383	7	2 – (299,119)
Bank [39]	45,211	16	2 – (39,922)
TüEyeQ [37]	15,762	76	2 – (12,975)
Poker-Hand [40]	1,025,000	10	9 – (513,701)
KDDCup	494,020	41	23 – (280,790)
Covertype [40]	581,012	54	7 – (283,301)
Gas [41]	13,910	128	6 – (3,009)
Insects-Abrupt [38]	355,275	33	6 – (101,256)
Insects-Incremental [38]	452,044	33	6 – (134,717)
SEA (synthetic, abrupt)	1,000,000	3	2
Agrawal (synthetic, incremental)	1,000,000	9	2
Hyperplane (synthetic, incremental)	500,000	50	2

Specifically, we store only a fixed number of statistics corresponding to the candidates with largest estimated gain (we recommend a default value of three times the number of features). At every time step, we allow a fixed percentage of the saved candidate statistics to be replaced by newly observed candidates. This is similar to the VFDT algorithm [11], which drops split candidates that diverge too far from the current maximal gain. We recommend a default replacement rate of 50%, which provided good results throughout all our experiments.

Since we limit the number of split candidates in main memory, we need to approximate the gain of newly observed candidates from the current sample. Note that the initial approximation can be biased if the current batch is not representative of the active concept. Specifically, such initial bias might occur if the batch size is small or the data is very noisy. Once stored, however, the statistics are updated at each successive time step, mitigating any initial bias over time. In addition, a split candidate that was rejected or deleted in the past can be added again in the future, e.g. if its importance has changed after concept drift. In experiments, we obtained good results for this simple approximation scheme.

Additionally, we propose a learning rate of 0.05 to train the binary and multinomial logit models and a threshold of $\epsilon = 10e - 8$ for the AIC-based confidence test.

Finally, note that we might be able to improve the efficiency of the Dynamic Model Tree by using parallelization or distributed computation. We leave a detailed discussion of more advanced implementation techniques for future work.

VI. Experiments

We evaluated the Dynamic Model Tree in multiple experiments on synthetic and real-world streaming classification data sets. Specifically, we compared the proposed framework to the related Model Tree architecture FIMT-DD [21] and different versions of the Hoeffding Tree. We begin with a description of the experimental setup, including the data sets, related methods and performance measures. Afterwards, we summarize our most important findings.

A. Environment and Evaluation Strategy

All models and experiments were implemented in Python (3.8.5) and run on an AMD Ryzen Threadripper 3960X (24x 3.8GHz) CPU with 128Gb RAM under Ubuntu 18.04. In addition, we used the following packages: numpy (1.20.1), pandas (1.2.4), matplotlib (3.4.2), scikit-learn (0.24.2) and scikit-multiflow (0.5.3). We specified a random state to guarantee the reproducibility of all results.

We performed a prequential (test-then-train) evaluation [43], which is the most common evaluation strategy for data stream learning. A disadvantage of data stream evaluations compared to regular batch evaluations is the lack of statistical significance. To be precise, since we cannot alter the order of observations without introducing artificial concept drift, we cannot obtain results for different permutations or samples of the data set. There are approaches where multiple instances of a classifier are trained in parallel [44]. However, they are very computationally intensive. Accordingly, we ask readers to be aware that statistical significance, although being standard in other areas of machine learning, is uncommon in the data stream literature.

At each iteration of the prequential evaluation, we processed a batch of 0.1% of the data. We also examined other batch sizes to ensure that the reported results are representative.

B. Data Sets

Typically, online classifiers are evaluated on tabular data sets. Machine learning with heterogeneous and evolving tabular data is challenging and has recently attracted attention in other areas such as deep learning [45]. In our experiments, we used state-of-the-art tabular streaming data sets, which we briefly describe in the following. We obtained most real-world data sets from *https://www.openml.org*. A summary of the data sets and their properties can also be found in Table I.

The Electricity data set describes price changes in the Australian New South Wales Electricity Market. The prices are not fixed, but adjust over time to the varying supply and demand. In the Airlines data set, the goal is to predict whether a flight will be delayed, given information about its scheduled departure. The Bank Marketing data set incorporates information about a marketing campaign of a Portuguese bank institute [39]. Here, the goal is to predict whether a customer will subscribe a deposit. Poker-Hand is a popular multiclass classification data set that consists of variables describing different poker hands [40]. Covertype contains information about several forest cover types that need to be distinguished [40]. The Gas data set contains drifting measurements of chemical sensors that are used to classify different types of gas [41]. The KDD Cup 1999 data set was introduced as part of a data mining competition. The data set contains features about network connections that are used for intrusion detection. We

TABLE II: *F1 Measure (higher is better).* We show the mean and standard deviation of the F1 measures observed over time in all data sets. For reference, we also provide the results of two ensemble classifiers separated from the stand-alone models by horizontal lines. We highlight the top result of each data set in bold letters. The average performance across all data sets is shown in the rightmost column. Note that the standard deviation also captures the variation caused by concept drift. It should therefore not be taken as an indication of the robustness to noise. The proposed Dynamic Model Tree frequently outperforms the remaining classifiers in terms of the predictive power and performs best on average.

Model \ Data Set	Electricity	Airlines	Bank	TüEyeQ	Poker	KDD	Covertype	Gas	Insects-Abr.	Insects-Inc.	SEA	Agrawal	Hyperplane	Mean
DMT (ours)	0.76 ± 0.20	0.63 ± 0.05	**0.88 ± 0.11**	**0.79 ± 0.20**	0.44 ± 0.05	**0.99 ± 0.01**	0.80 ± 0.09	**0.82 ± 0.27**	**0.73 ± 0.10**	**0.73 ± 0.08**	0.88 ± 0.02	0.82 ± 0.08	**0.84 ± 0.04**	**0.78 ± 0.10**
FIMT-DD [21]	0.78 ± 0.20	0.55 ± 0.12	**0.88 ± 0.14**	0.76 ± 0.22	0.41 ± 0.08	**0.99 ± 0.01**	0.81 ± 0.10	0.79 ± 0.28	**0.73 ± 0.08**	0.72 ± 0.08	0.78 ± 0.10	0.64 ± 0.13	0.76 ± 0.05	0.74 ± 0.12
VFDT (MC) [11]	0.76 ± 0.20	0.64 ± 0.06	0.87 ± 0.15	0.77 ± 0.22	0.47 ± 0.05	0.96 ± 0.10	0.72 ± 0.13	0.29 ± 0.37	0.64 ± 0.14	0.67 ± 0.10	0.86 ± 0.03	0.77 ± 0.11	0.65 ± 0.03	0.70 ± 0.13
VFDT (NBA) [31]	**0.80 ± 0.15**	**0.65 ± 0.05**	**0.88 ± 0.13**	0.77 ± 0.21	**0.50 ± 0.03**	**0.99 ± 0.01**	**0.85 ± 0.09**	0.77 ± 0.27	0.71 ± 0.10	0.72 ± 0.07	0.86 ± 0.04	0.79 ± 0.10	0.73 ± 0.02	0.77 ± 0.10
HT-ADA [13]	0.77 ± 0.21	0.62 ± 0.07	**0.88 ± 0.13**	0.77 ± 0.23	0.47 ± 0.05	0.96 ± 0.10	0.67 ± 0.19	0.22 ± 0.35	0.59 ± 0.15	0.64 ± 0.13	**0.89 ± 0.02**	**0.84 ± 0.08**	0.66 ± 0.03	0.69 ± 0.13
EFDT [14]	0.77 ± 0.20	0.60 ± 0.09	**0.88 ± 0.14**	0.77 ± 0.23	0.47 ± 0.05	**0.99 ± 0.01**	0.74 ± 0.14	0.55 ± 0.39	0.68 ± 0.11	0.65 ± 0.10	0.87 ± 0.04	0.82 ± 0.09	0.69 ± 0.03	0.73 ± 0.12
Forest Ens. [42]	**0.81 ± 0.14**	0.64 ± 0.05	**0.89 ± 0.13**	**0.78 ± 0.20**	0.50 ± 0.02	**0.99 ± 0.01**	**0.74 ± 0.19**	**0.80 ± 0.33**	0.72 ± 0.09	0.72 ± 0.08	**0.90 ± 0.02**	0.80 ± 0.08	0.64 ± 0.03	0.76 ± 0.10
Bagging Ens. [27]	**0.81 ± 0.17**	**0.65 ± 0.05**	**0.89 ± 0.13**	**0.78 ± 0.21**	**0.53 ± 0.03**	**0.99 ± 0.04**	0.72 ± 0.23	0.67 ± 0.40	**0.74 ± 0.10**	**0.75 ± 0.07**	**0.90 ± 0.02**	**0.84 ± 0.08**	**0.72 ± 0.04**	**0.77 ± 0.12**

TABLE III: *No. of Splits (lower is better).* Complexity – quantified here by the mean and standard deviation of the number of splits (as described in Section VI-D2) – is often used as an indicator of the interpretability of a model. Model Trees (FIMT-DD and DMT) tend to remain shallower than Hoeffding Trees, due to the flexibility provided by the linear leaf models.

Model \ Data Set	Electricity	Airlines	Bank	TüEyeQ	Poker	KDD	Covertype	Gas	Insects-Abr.	Insects-Inc.	SEA	Agrawal	Hyperplane	Mean
DMT (ours)	6.5 ± 3.1	35.7 ± 16.7	**2.3 ± 1.0**	1.4 ± 0.8	**9.0 ± 0.0**	24.8 ± 6.3	10.7 ± 4.0	9.3 ± 3.5	9.1 ± 3.5	**9.1 ± 3.5**	35.1 ± 25.3	75.4 ± 34.4	**2.2 ± 1.3**	**17.7 ± 8.0**
FIMT-DD [21]	52.0 ± 30.1	**4.9 ± 3.9**	75.5 ± 47.3	**1.0 ± 0.0**	17.7 ± 10.2	24.8 ± 6.4	13.7 ± 8.2	6.0 ± 0.0	**7.4 ± 3.1**	10.6 ± 5.9	**1.0 ± 0.0**	**65.8 ± 71.5**	8.0 ± 10.3	22.2 ± 15.1
VFDT (MC) [11]	37.8 ± 22.3	323.3 ± 182.4	21.9 ± 13.9	10.6 ± 6.8	84.7 ± 50.6	25.6 ± 13.0	356.8 ± 201.7	0.7 ± 0.7	41.3 ± 23.7	53.5 ± 32.5	588.4 ± 339.8	628.3 ± 371.0	277.9 ± 162.4	188.5 ± 109.3
VFDT (NBA) [31]	76.7 ± 44.6	647.6 ± 364.7	44.8 ± 27.7	22.3 ± 13.7	856.3 ± 506.0	637.3 ± 310.8	2861.1 ± 1613.4	11.1 ± 5.1	295.2 ± 165.7	380.3 ± 227.6	1177.8 ± 679.7	1257.6 ± 742.1	556.8 ± 324.9	678.8 ± 386.6
HT-ADA [13]	**3.4 ± 2.1**	12.7 ± 6.8	5.6 ± 3.4	2.3 ± 1.6	58.0 ± 28.1	25.4 ± 12.8	**3.1 ± 2.9**	**0.2 ± 0.4**	8.0 ± 5.0	21.5 ± 12.9	131.4 ± 69.8	158.2 ± 79.2	188.7 ± 101.4	47.6 ± 25.1
EFDT [14]	10.9 ± 4.5	15.2 ± 7.5	9.5 ± 3.4	2.8 ± 1.4	10.0 ± 6.6	**24.7 ± 9.2**	9.4 ± 4.3	4.7 ± 2.7	17.3 ± 7.8	15.9 ± 10.4	109.9 ± 70.3	89.7 ± 66.2	31.0 ± 17.4	27.0 ± 16.3

TABLE IV: *No. of Parameters (lower is better).* For the sake of completeness and to account for the difference between majority weighting and linear leaf models, we depict the number of parameters (mean ± standard deviation) as another measure of complexity (as described in Section VI-D2). In general, heuristic measures like the number of splits or parameters do not always give a clear indication of the interpretability of a model and should thus be considered with care. A more reliable indication of interpretability is provided by theoretical properties such as Property 1 and 2.

Model \ Data Set	Electricity	Airlines	Bank	TüEyeQ	Poker	KDD	Covertype	Gas	Insects-Abr.	Insects-Inc.	SEA	Agrawal	Hyperplane	Mean
DMT (ours)	33 ± 14	146 ± 67	27 ± 8	92 ± 31	80 ± 0	970 ± 238	474 ± 162	939 ± 320	237 ± 82	238 ± 82	71 ± 51	381 ± 172	80 ± 33	290 ± 97
FIMT-DD [21]	238 ± 136	**22 ± 15**	649 ± 402	76 ± 0	150 ± 83	971 ± 239	597 ± 332	640 ± 0	198 ± 74	275 ± 140	**3 ± 0**	333 ± 358	229 ± 262	337 ± 157
VFDT (MC) [11]	77 ± 45	648 ± 365	45 ± 28	22 ± 14	170 ± 101	52 ± 26	715 ± 403	2 ± 1	84 ± 47	108 ± 65	1178 ± 680	1258 ± 742	557 ± 325	378 ± 219
VFDT (NBA) [31]	349 ± 201	2,594 ± 1,459	388 ± 236	896 ± 526	6,943 ± 4,099	24,016 ± 11,695	116,270 ± 65,543	1,105 ± 470	7,023 ± 3,930	9,042 ± 5,397	2,357 ± 1,359	6,292 ± 3,710	14,224 ± 8,285	14,731 ± 8,224
HT-ADA [13]	**8 ± 4**	27 ± 14	**12 ± 7**	**6 ± 3**	144 ± 78	52 ± 26	**7 ± 6**	**1 ± 1**	**17 ± 10**	44 ± 26	264 ± 140	377 ± 193	378 ± 203	103 ± 55
EFDT [14]	23 ± 9	31 ± 15	20 ± 7	7 ± 3	**21 ± 13**	**50 ± 18**	20 ± 9	10 ± 5	36 ± 16	**33 ± 21**	221 ± 141	**180 ± 132**	**63 ± 35**	**55 ± 33**

shuffled the KDD data set, because it was initially grouped by class labels. Since KDD does not involve known concept drift, shuffling the data is required to obtain an even distribution of classes over time and enable a fair evaluation.

It is usually difficult to determine the exact period of concept drift in a real-world streaming process. In fact, we cannot access such information for any of the above-mentioned data sets. Two recent exceptions are the TüEyeQ [37] and Insects [38] data collections. From TüEyeQ, we used the sociodemographic data about all subjects participating in an IQ test. The classification task is to decide whether a subject fails or passes an IQ-related task. The data set is divided in four task blocks with increasing difficulty within each block, resembling a natural concept drift. The Insects data comprises sensor information from monitoring of flying insect species. The measurements were obtained in a non-stationary but controllable environment. That is, by changing the temperature and humidity, the authors in [38] were able to generate different types of concept drift. We used the imbalanced Insects data sets with abrupt and incremental drift.

In addition, we created synthetic data streams with *scikit-multiflow* [30]. Specifically, we used the AGRAWALGenerator, HyperplaneGenerator and SEAGenerator to obtain synthetic data with different types of concept drift. For detailed information about each data generator, we refer to the corresponding documentation. Each synthetic data stream was sampled with 0.1 probability of noisy inputs (this corresponds to the "perturbation" parameter of the scikit-multiflow classes).

The resulting Hyperplane data set is subject to a continuous incremental concept drift over all observations. The Agrawal data set contains incremental drift between the observations 100,000-200,000, 300,000-500,000 and 800,000-900,000, but is otherwise stable. The SEA data set has three abrupt concept drifts at the observations 200,000, 400,000, 600,000 and 800,000.

Finally, we factorised the categorical string variables of all data sets. In addition, we normalized the features before use (range $[0, 1]$). Otherwise, we did not pre-process the data sets.

C. Related Algorithms and Hyperparameters

As mentioned before, we compared the Dynamic Model Tree to different versions of the Hoeffding Tree. Specifically, we obtained results for the basic VFDT [11] and two of its extensions, the adaptive Hoeffding Tree (HT-Ada) [13] and the Extremely Fast Decision Tree (EFDT) [14]. Unlike VFDT, both extensions contain a mechanism to adapt to concept drift.

Since it is generally not possible to optimize hyperparameters in a data stream, we applied the default configurations suggested by the corresponding scikit-multiflow implementations. These implementations have been heavily optimized over the years. Since our goal was to compare the originally proposed models, and in order to allow a fairer comparison with our implementation, we disabled some of the optimizations of scikit-multiflow. In particular, we did not use bootstrap sampling in the leaves of the HT-Ada algorithm. Moreover, we used majority voting in the leaf nodes of the Hoeffding Trees. However, to give an indication of the possible improvement introduced by simple predictive models in the leaves of a Hoeffding Tree, we also report the results of a VFDT augmented with adaptive Naïve Bayes models [31]. Finally, to improve the efficiency of the EFDT algorithm, we set the minimum number of observations between re-evaluations to 1,000.

For the sake of completeness, we also looked at two state-of-the-art ensembles of the Hoeffding Tree, an Adaptive Random Forest [42] and a Leveraging Bagging Ensemble [27]. Both ensembles were trained with 3 basic Hoeffding Tree classifiers as weak learners. We configured the weak learners in the same way as the stand-alone VFDT model. Otherwise, we used the default parameters of the ensembles specified in scikit-multiflow.

In addition, we evaluated FIMT-DD [21]. To the best of our knowledge, there is no publicly available Python implementation of a FIMT-DD classification model. Therefore, we implemented the classifier based on the description in the paper.[2] Our implementation uses the second drift adjustment strategy proposed by the authors, i.e., it deletes branches where the Page Hinkley test issues an alert. We used a default learning rate of 0.01 for the simple models and a threshold of 0.01 for the significance test based on Hoeffding's inequality. Besides, we specified a threshold of 0.05 to break ties between split candidates with similar gain.

Finally, note that we only allowed binary splits in all incremental decision trees. The Dynamic Model Tree was configured in the way described in Section V.

D. Performance Measures

1) Predictive Performance: Classification error and accuracy are common measures for evaluating online classifiers. However, both measures might produce biased results for imbalanced data. As our evaluation incorporates many imbalanced data sets, we report the F1 measure instead. The F1 measure is the harmonic mean of precision and recall and provides reliable results even for strong imbalances.

2) Interpretability/Complexity: Since there is no common measure of interpretability, one usually resorts to heuristics. For example, one can compare the number of parameters in linear models or the number of nodes in decision trees. Unfortunately, in our case there is no clear separation between model families. In particular, a comparison between the complexities of Hoeffding and Model Trees is difficult, as their leaf nodes offer different degrees of expressiveness. Hence, we consider the *number of splits* in our evaluation, which we calculated as follows: Each inner node counted as one split. Majority-weighted leaf nodes did not contribute to the total number of splits. Conversely, the leaf classifiers can be considered as another final split of the observations. Accordingly, we counted one more split for binary classifiers and c more splits for multiclass classifiers, where c is the number of classes. Compared to measuring the total number of nodes, the *number of splits* accounts for the different leaf types of Hoeffding Trees and Model Trees. For completeness, we also report the *number of parameters*. Specifically, we counted one parameter per inner node corresponding to the split value. We counted leaf nodes as either one (majority class) or m (linear model weights; Naïve Bayes conditional probabilities) additional parameters, where m is the number of features.[3]

In practice, it depends on a given application whether the simple leaf models should be considered as limiting interpretability or not. That is, as mentioned earlier, simple models can offer significant advantages in terms of local feature-based explainability. Accordingly, we generally consider the *number of splits* to be a more reliable indication of the interpretability of incremental decision trees. Still, instead of giving too much importance to heuristics, one should aim for online learning models that have meaningful interpretability properties, such as those proposed in this paper.

3) Computational Efficiency: Computational efficiency depends on the respective implementation and hardware configuration. As we used both scikit-multiflow and custom implementations, we did not focus on computational efficiency in the experiments. However, for the sake of completeness, we provide the average computation time for one train/test-iteration of each model in Table V.

TABLE V: *Computation Time in Seconds (lower is better).* We show the mean and standard deviation of the computation time for one test/train iteration over all data sets.

DMT (ours)	FIMT-DD	VFDT (MC)	VFDT (NBA)	HT-ADA	EFDT
0.53 ± 0.21	1.12 ± 0.57	**0.06 ± 0.03**	0.14 ± 0.03	0.34 ± 0.08	17.23 ± 6.33

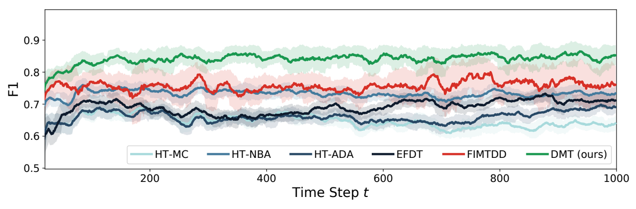
E. Results

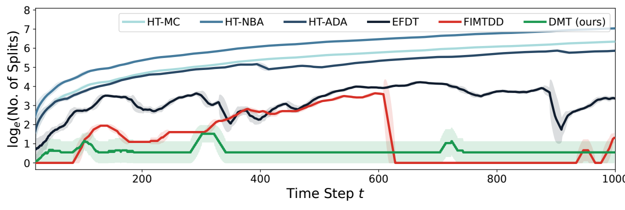
In the following, we discuss our most important findings.

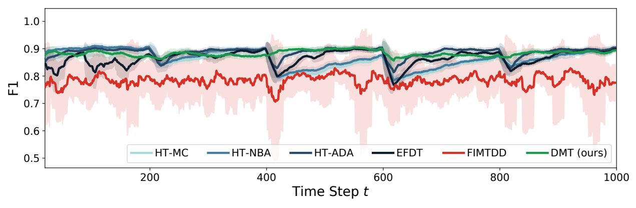
1) Predictive Performance: Table II shows the average F1 measure of all models and data sets. Using simple leaf models instead of majority voting has generally improved the obtained F1 score (see DMT, FIMT-DD and VFDT (NBA)). This advantage is most evident in the Hyperplane data set. The Hyperplane data was generated by rotating a decision hyperplane in multidimensional space. Thus, after a few splits, the observations can be linearly separated sufficiently well by the simple models. Although no model achieved good predictive quality on Poker, the VFDT with Naïve Bayes has a higher average score than the Model Trees, suggesting that a different simple model type may improve the results.

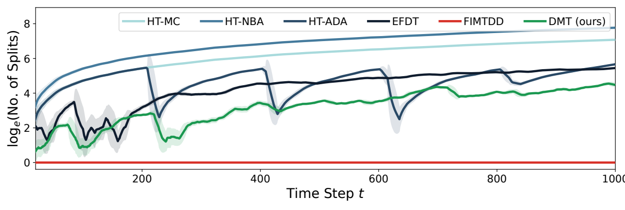
[2]The FIMT-DD implementation can also be accessed via Github at https://github.com/haugjo/dynamic-model-tree

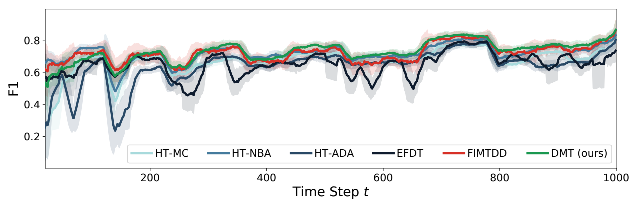
[3]For multinomial classification, we counted the parameters corresponding to each class.

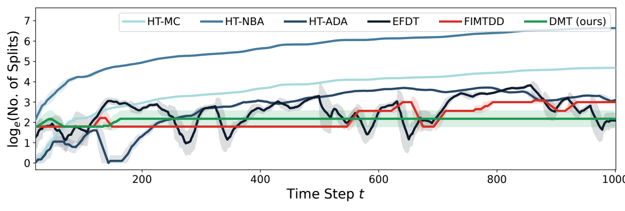

(a) Hyperplane (Incremental Drift), F1 Measure

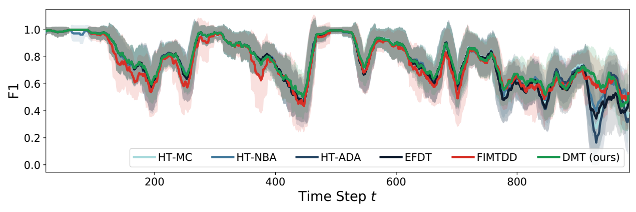

(b) Hyperplane (Incremental Drift), Log Number of Splits


(c) SEA (Abrupt Drifts), F1 Measure


(d) SEA (Abrupt Drifts), Log Number of Splits


(e) Insects-Inc. (Incremental Drift), F1 Measure


(f) Insects-Inc. (Incremental Drift), Log Number of Splits


(g) TüEyeQ (Abrupt Drifts), F1 Measure

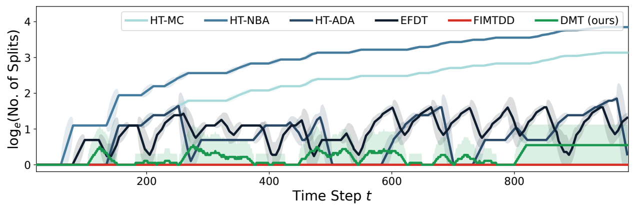

(h) TüEyeQ (Abrupt Drifts), Log Number of Splits

Fig. 3: *Performance and Complexity Under Concept Drift.* We show the F1 scores and the log number of splits for four data sets with known concept drift. We indicate the types of concept drift in parentheses. Specifically, we show the mean and standard deviation (shaded area) for a sliding window aggregation with a window size of 20. The Dynamic Model Tree has less performance degradation and recovers faster after a concept drift, while often remaining shallower than existing models.

In general, FIMT-DD and the Dynamic Model Tree obtained similar F1 scores. However, our framework outperformed FIMT-DD for Airlines and the synthetic data sets. Looking at the behavior of FIMT-DD over time, its disadvantage can often be attributed to slow growth (e.g., Airlines, Agrawal, SEA) or aggressive pruning (e.g., Agrawal, Hyperplane). Accordingly, a less strict split threshold, different purity measures, and alternative pruning strategies could be explored in the future. Similarly, the Dynamic Model Tree may perform poorly in the first time steps if the random initial weights have not yet converged. This effect is noticeable in the averaged result of Electricity, since it is a relatively small data set. To speed up the initial training of the simple models, one may experiment with dynamic learning rates. As can be seen from the Gas data set, the VFDT and HT-Ada implementations may have difficulty finding optimal split candidates for high-dimensional and continuous feature sets. Both models remained extremely shallow and obtained poor predictive performance. In such cases, where it is difficult or infeasible to find a good split value among all possible candidates, the simple leaf models can provide an advantage. Besides, HT-Ada was not competitive for Covertype and Insects-Abrupt. Here, pruning near the root caused temporary declines of the F1 score. Such behaviour might be avoided by a less aggressive pruning strategy or a more robust drift detection scheme.

Our framework obtained either the best or second best average F1 score for all data sets with known concept drift (TüEyeQ, Insects-Abrupt, Insects-Incremental, SEA, Agrawal and Hyperplane). We depict the detailed results of four data sets in Figure 3. The Dynamic Model Tree often suffers

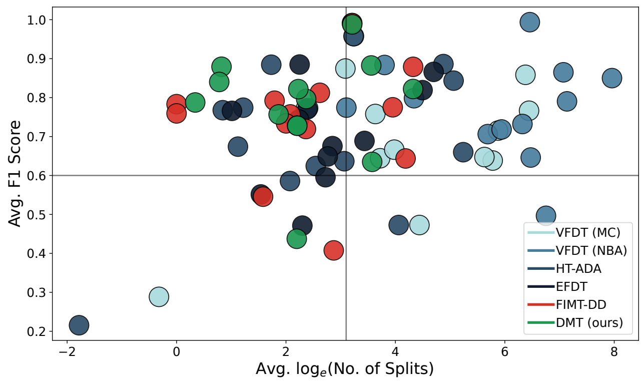


Fig. 4: *Predictive Performance vs. Model Complexity.* Above, we compare the F1 measure and the logarithm of the number of splits of each incremental decision tree. The number of splits is an indication of model complexity, which in turn is a common proxy for the interpretability. That is, fewer splits can usually be associated with higher interpretability. Each point corresponds to the average measure of one data set. Detailed results can be found in the Tables II and III. Ideally, we aim for a large F1 score and a small number of splits, corresponding to a value in the top left quadrant. While achieving competitive F1 scores, the Dynamic Model Tree generally manages to reduce the number of splits compared to the Hoeffding Trees.

TABLE VI: *Experiment Summary.* We provide a concise summary of our experiments. For more detailed results, please see the remaining tables and plots. We ranked all methods according to four categories. Both predictive performance categories are based on the results in Table II. The second category reflects the average performance for the data sets with known concept drift. The complexity and efficiency scores are based on the average results in the Tables III and V. We used the following methodology: The best and worst models per category have received a score of **++** and **– –** respectively. The other methods have received a score of **+** or **–** depending on whether they were above or below the median.

Model \ Category	Overall Pred. Performance	Pred. Performance For Known Drift	Complexity/ Interpretability	Computational Efficiency
DMT (ours)	++	++	++	–
FIMT-DD [21]	+	–	+	–
VFDT (MC) [11]	–	– –	–	++
VFDT (NBA) [31]	+	+	– –	+
HT-Ada [13]	– –	–	–	+
EFDT [14]	–	+	+	– –

only minor performance deterioration after a concept drift. Compared to the other models, our framework usually recovers faster from both abrupt and incremental concept drift. The effect is most notable in the SEA and Insects data set.

In summary, the proposed Dynamic Model Tree (DMT) is among the best performing models for most data sets. In fact, our framework ranks first place on average, even when the more powerful ensemble models are taken into account.

2) Complexity and Interpretability: As described above, we report the *number of splits* (Table III) and the *number of parameters* (Table IV) as indicators for the interpretability of a model. Model Trees often maintain a shallower tree structure than Hoeffding Trees. This effect can be attributed to the additional flexibility provided by the simple models. For example, the synthetic Hyperplane and SEA data sets can both be separated by a hyperplane. The Dynamic Model Tree was able to represent these linear relationships with fewer splits than the Hoeffding Trees, while achieving similar or higher predictive quality. The complexities of the Dynamic Model Tree and FIMT-DD often varied. We attribute this effect to the loss-based gains that allow our framework to meet the *consistency with parent splits* and *model minimality* properties. Specifically, while the Dynamic Model Tree will only retain a split, if it is beneficial in terms of the loss, FIMT-DD retains a split as long as the Page Hinkley test does not detect a concept drift. This may lead to overly complex trees that offer only slight or no improvements in terms of the F1 score (see Electricity and Bank). Likewise, if there is no significant difference according to the Hoeffding bound, FIMT-DD does not split a node, even though this might reduce the expected loss (see Airlines and SEA). In addition, FIMT-DD aims to reduce the standard deviation of the target and can therefore obtain leaf nodes that are extremely imbalanced towards one class. While this would be beneficial for majority weighting, it could make training simple (linear) models more difficult. Ultimately, this may reduce the predictive performance of FIMT-DD compared to a Dynamic Model Tree, even though both models have similar complexity (see Agrawal).

The Dynamic Model Tree ranks first for the average *number of splits* and third for the more conservative *number of parameters*. Indeed, Figure 3 shows that the complexity of the Dynamic Model Tree typically remains low over time, while other methods such as VFDT produce increasingly larger trees. Besides, the Dynamic Model Tree can adapt to different types of concept drift without drastically changing its complexity.

In general, our results demonstrate that high predictive performance and low complexity need not be mutually exclusive in an evolving data stream. The relationship of predictive performance and complexity is also shown in Figure 4. A summary of our experiments is depicted in Table VI.

VII. Conclusion

In this paper, we introduced the Dynamic Model Tree, a flexible and interpretable framework for machine learning on large-scale evolving data streams. A Dynamic Model Tree adheres to sensible properties that make it a reliable choice even in highly challenging streaming scenarios. Our experiments show that the proposed framework can achieve state-of-the-art performance with a fraction of the complexity of many previous methods. In particular, the Dynamic Model Tree automatically adapts to different types of concept drift, without the need for complex model extensions common in existing frameworks. Accordingly, we hope that our work will support the current trend towards more efficient and interpretable machine learning.

REFERENCES

[1] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 267, pp. 1–38, feb 2019.

[2] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019.

[3] C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, and C. Rudin, "This looks like that: Deep learning for interpretable image recognition," *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, Jun. 2018.

[4] M. Du, N. Liu, and X. Hu, "Techniques for interpretable machine learning," *Communications of the ACM*, vol. 63, no. 1, pp. 68–77, dec 2019.

[5] G. Kasneci and T. Gottron, "Licon: A linear weighting scheme for the contribution ofinput variables in deep artificial neural networks," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016, pp. 45–54.

[6] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st international conference on neural information processing systems*, 2017, pp. 4768–4777.

[7] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[8] J. Haug, S. Zürn, P. El-Jiz, and G. Kasneci, "On baselines for local feature attributions," *AAAI-21 Explainable Agency in Artificial Intelligence Workshop*, 2021.

[9] A. Bibal and B. Frénay, "Interpretability of machine learning models and representations: an introduction," in *ESANN*, 2016.

[10] M. Moshkovitz, Y.-Y. Yang, and K. Chaudhuri, "Connecting interpretability and robustness in decision trees through separation," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 7839–7849.

[11] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 71–80.

[12] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 97–106.

[13] A. Bifet and R. Gavalda, "Adaptive learning from evolving data streams," in *International Symposium on Intelligent Data Analysis*. Springer, 2009, pp. 249–260.

[14] C. Manapragada, G. I. Webb, and M. Salehi, "Extremely fast decision tree," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1953–1962.

[15] L. Rutkowski, L. Pietruczuk, P. Duda, and M. Jaworski, "Decision trees for mining data streams based on the mcdiarmid's bound," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1272–1279, 2012.

[16] P. Matuszyk, G. Krempl, and M. Spiliopoulou, "Correcting the usage of the hoeffding inequality in stream mining," in *International Symposium on Intelligent Data Analysis*. Springer, 2013, pp. 298–309.

[17] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, "The cart decision tree for mining data streams," *Information Sciences*, vol. 266, pp. 1–15, 2014.

[18] J. R. Quinlan *et al.*, "Learning with continuous classes," in *5th Australian joint conference on artificial intelligence*, vol. 92. World Scientific, 1992, pp. 343–348.

[19] K. Broelemann and G. Kasneci, "A gradient-based split criterion for highly accurate and transparent model trees," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI*, 2019.

[20] D. Potts and C. Sammut, "Incremental learning of linear model trees," *Machine Learning*, vol. 61, no. 1-3, pp. 5–48, 2005.

[21] E. Ikonomovska, J. Gama, and S. Džeroski, "Learning model trees from evolving data streams," *Data mining and knowledge discovery*, vol. 23, no. 1, pp. 128–168, 2011.

[22] J. Gama, "A survey on learning from data streams: current and future trends," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 45–55, 2012.

[23] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Computational Intelligence Magazine*, vol. 10, no. 4, pp. 12–25, 2015.

[24] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, "A survey on ensemble learning for data stream classification," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–36, 2017.

[25] J. P. Barddal and F. Enembreck, "Regularized and incremental decision trees for data streams," *Annals of Telecommunications*, pp. 1–11, 2020.

[26] C. Manapragada, G. I. Webb, M. Salehi, and A. Bifet, "Emergent and unspecified behaviors in streaming decision trees," *arXiv preprint arXiv:2010.08199*, 2020.

[27] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2010, pp. 135–150.

[28] J. Montiel, R. Mitchell, E. Frank, B. Pfahringer, T. Abdessalem, and A. Bifet, "Adaptive xgboost for evolving data streams," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

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

[30] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 2915–2914, 2018.

[31] J. Gama, R. Rocha, and P. Medas, "Accurate decision trees for mining high-speed data streams," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 523–528.

[32] A. Bifet, G. Holmes, B. Pfahringer, and E. Frank, "Fast perceptron decision tree learning from evolving data streams," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2010, pp. 299–310.

[33] H. Gouk, B. Pfahringer, and E. Frank, "Stochastic gradient trees," in *Asian Conference on Machine Learning*. PMLR, 2019, pp. 1094–1109.

[34] J. Haug and G. Kasneci, "Learning parameter distributions to detect concept drift in data streams," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 9452–9459.

[35] J. Haug, M. Pawelczyk, K. Broelemann, and G. Kasneci, "Leveraging model inherent variable importance for stable online feature selection," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1478–1502.

[36] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[37] E. Kasneci, G. Kasneci, T. Appel, J. Haug, F. Wortha, M. Tibus, U. Trautwein, and P. Gerjets, "Tüeyeq, a rich iq test performance data set with eye movement, educational and socio-demographic information," *Scientific Data*, vol. 8, no. 1, pp. 1–14, 2021.

[38] V. M. Souza, D. M. dos Reis, A. G. Maletzke, and G. E. Batista, "Challenges in benchmarking stream learning algorithms with real-world data," *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1805–1858, 2020.

[39] S. Moro, R. Laureano, and P. Cortez, "Using data mining for bank direct marketing: An application of the crisp-dm methodology," *Proceedings of the European Simulation and Modelling Conference - ESM'2011*, 2011.

[40] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[41] A. Vergara, S. Vembu, T. Ayhan, M. A. Ryan, M. L. Homer, and R. Huerta, "Chemical gas sensor drift compensation using classifier ensembles," *Sensors and Actuators B: Chemical*, vol. 166, pp. 320–329, 2012.

[42] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, vol. 106, no. 9, pp. 1469–1495, 2017.

[43] J. Gama, R. Sebastião, and P. P. Rodrigues, "Issues in evaluation of stream learning algorithms," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*. ACM Press, 2009.

[44] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, and B. Pfahringer, "Efficient online evaluation of big data stream classifiers," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2015.

[45] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *arXiv preprint arXiv:2110.01889*, 2021.

A.5 Change Detection for Local Explainability in Evolving Data Streams

Publication: Accepted at the 31st ACM International Conference on Information and Knowledge Management (CIKM 2022) and published on ArXiv. At the time of publication of this thesis, the official version of record of the paper was not yet available.

Contribution: I had the idea to analyze local attributions under the influence of concept drift. With the help of Gjergji Kasneci, I then concretized the early idea and produced a first draft of the paper. As part of a student project, Alexander Braun and Stefan Zürn implemented early versions of the change detection method and experiments, which I later revised and extended. Gjergji Kasneci made valuable suggestions for improving the experiments and helped refine the text and formalization.

Change Detection for Local Explainability in Evolving Data Streams

Johannes Haug
johannes-christian.haug@uni-tuebingen.de
University of Tuebingen
Tuebingen, Germany

Alexander Braun
al.braun@student.uni-tuebingen.de
University of Tuebingen
Tuebingen, Germany

Stefan Zürn
stefan.zuern@student.uni-tuebingen.de
University of Tuebingen
Tuebingen, Germany

Gjergji Kasneci
gjergji.kasneci@uni-tuebingen.de
University of Tuebingen
Tuebingen, Germany

ABSTRACT

As complex machine learning models are increasingly used in sensitive applications like banking, trading or credit scoring, there is a growing demand for reliable explanation mechanisms. Local feature attribution methods have become a popular technique for post-hoc and model-agnostic explanations. However, attribution methods typically assume a stationary environment in which the predictive model has been trained and remains stable. As a result, it is often unclear how local attributions behave in realistic, constantly evolving settings such as streaming and online applications. In this paper, we discuss the impact of temporal change on local feature attributions. In particular, we show that local attributions can become obsolete each time the predictive model is updated or concept drift alters the data generating distribution. Consequently, local feature attributions in data streams provide high explanatory power only when combined with a mechanism that allows us to detect and respond to local changes over time. To this end, we present CDLEEDS, a flexible and model-agnostic framework for detecting local change and concept drift. CDLEEDS serves as an intuitive extension of attribution-based explanation techniques to identify outdated local attributions and enable more targeted recalculations. In experiments, we also show that the proposed framework can reliably detect both local and global concept drift. Accordingly, our work contributes to a more meaningful and robust explainability in online machine learning.

CCS CONCEPTS

• **Computing methodologies → Online learning settings**.

KEYWORDS

online machine learning; explainable machine learning; concept drift detection; local feature attributions

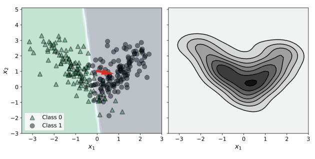
ACM Reference Format:
Johannes Haug, Alexander Braun, Stefan Zürn, and Gjergji Kasneci. 2022. Change Detection for Local Explainability in Evolving Data Streams. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3511808.3557257

1 INTRODUCTION

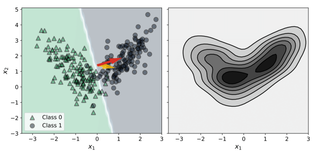
Data streams are abundant in modern applications such as financial trading, social media, online retail, sensor-driven production or urban infrastructure [19]. To perform machine learning on large amounts of streaming data, we require powerful and efficient online learning models. Likewise, if we are to use online machine learning for high-stakes decisions, e.g. in online credit scoring or healthcare [49], we need reliable mechanisms to explain the model and its predictions. However, the explainability of online machine learning models has received only little attention in the past.

Online machine learning is generally more challenging than its offline counterpart. Aside from limited resources and real-time demands, online learning models must deal with changing environments and, in particular, concept drift, i.e., a shift in the data generating distribution [24]. Concept drift can manifest itself in most practical applications. For example, an online retailer must adapt product recommendations to changing customer preferences. Similarly, social media platforms need to consider the shifting interests of their users. If we do not account for concept drift, the performance and reliability of online learning methods can suffer.

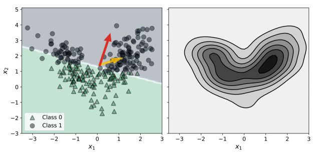
Due to stricter regulations and increased public awareness, interest in mechanisms for explainable machine learning has gained momentum in recent years. In this context, local feature attribution methods have become one of the most popular families of post-hoc explanation models [31, 34, 41, 42]. Local attribution methods aim to quantify the local importance of input features in the prediction. Traditionally, local attribution methods are used to explain the complex predictive model once it is trained and stationary. However, in data streams, concept drift requires that we continue updating the predictive model; accordingly, its explanation must also be updated. For example, we need to ensure that the explanations we give to a credit applicant are still meaningful after we update the predictive model with new customer data. However, although local attribution methods are commonly used, it is usually unclear how they behave in a realistic and dynamic online environment.



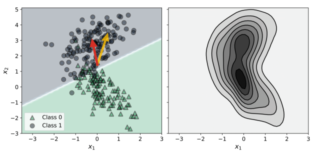
(a) t=1: The classes are separated with an accuracy ≈ 85%.



(b) t=2: Virtual concept drift of the positive class (the conditional target distribution remains unaffected).



(c) t=3: Real concept drift (the conditional target distribution has changed).



(d) t=4: Virtual and real concept drift.

Figure 1: *Local Attributions in a Synthetic Data Stream.* Concept drift can cause drastic changes of the local feature attributions obtained in a streaming application, which we illustrate above with synthetic data. We used a logistic regression classifier and the SHAP attribution framework [34]. Each plot depicts one of four time steps. The left subplots show the current data batch (250 observations), the decision boundary (between the green and blue areas) and the mean SHAP attribution of the current (red arrow) and previous time step (yellow arrow). The right subplots show the kernel density estimate of the observations. *Both real and virtual concept drift change the decision boundary and thus the expected attribution between time steps.*

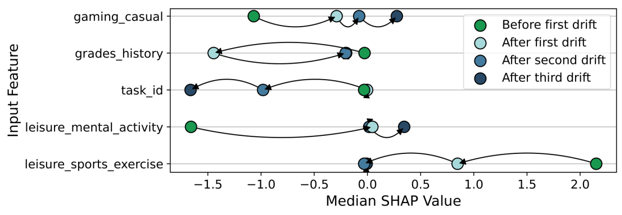


Figure 2: *Local Attributions in a Real-World Setting.* We trained a logistic regression classifier on the TüEyeQ data set [30], which comprises sociodemographic information of 315 subjects in an IQ test. The data set contains natural concept drifts by switching between 4 blocks of IQ-related tasks. Above, we show the median SHAP attributions [34] of the 5 input features with the highest variation over time. Online training and concept drift lead to drastic changes in the attributions of this real-world stream of tasks (i.e., solving IQ test items). For example, the *task-id* has a greater (negative) importance in later task blocks that are more difficult to solve (for more information about the features see [30]). *To achieve better feature-based explanations in online scenarios, we need to identify such (local) changes of the attributions.*

1.1 Local Attributions Under Concept Drift

Indeed, a simple example shows that local attributions for an incrementally trained machine learning model can change significantly over time. For illustration, we generated an artificial two-dimensional data set that underlies different types of concept drift (we discuss the two fundamental types of concept drift more formally in Section 3). Figure 1 illustrates four time steps from the training procedure of a logistic regression model. Strikingly, the expected feature attributions (SHAP values [34], red and yellow arrows) changed drastically due to shifts in the decision boundary of the classifier (green and blue areas), which in turn were caused by concept drift. This example shows that local attributions generated at a certain point in time may lose their validity after a single update of the predictive model. In fact, we observe such behavior in real-world settings. For example, the SHAP attributions of a recent IQ study [30] underlie considerable change over time (Figure 2). For the long-term acceptance of machine learning models in sensitive applications, one has to address such changes in the data.

Ideally, local feature attribution methods should account for changes in a data stream by design, e.g., through robustness to small model variations or efficient incremental update procedures. However, most existing attribution methods produce point estimates and rely on costly sampling- or permutation-based approximations [34, 42]. Thus, without knowledge of the hidden dynamics underlying a data stream – and in particular concept drift – we would have to recalculate the local attributions over and over again to ensure their validity. Given the scale of most data streams, however, it is typically infeasible to simply recompute all past attribution vectors at every time step. In fact, since incremental updates, e.g., via stochastic gradient descent, often only alter parts of the model, recalculation of all previous attributions would often be unnecessary. Therefore, in order to efficiently establish local feature attributions in data streams, enable more informed decisions, and provide an overall higher degree of explainability, we require an effective mechanism to detect local changes over time. Indeed, we argue that *(local) change detection should be part of any sensible strategy for explainable machine learning in evolving data streams.*

1.2 Our Contribution

In this paper, we introduce a novel framework for *Change Detection for Local Explainability in Evolving Data Streams (CDLEEDS).* CDLEEDS serves as a generic and model-agnostic extension of local attribution methods in data streams. The proposed change detection strategy arises naturally from the behavior of local attribution methods in the presence of incremental model updates and concept drift. In fact, we show that due to a fundamental property of many attribution methods, local change can be reliably detected without computing a single attribution score. We propose an effective implementation of CDLEEDS via adaptive hierarchical clustering. In experiments, we show that our approach can help to significantly reduce the number of recalculations of a local attribution over time. CDLEEDS is also one of the first drift detection methods capable of detecting concept drift at different levels of granularity. Thus, CDLEEDS can have general value for online machine learning applications – even outside the context of explainability. For illustration, we compared our model with several state-of-the-art drift detection methods. Notably, CDLEEDS was able to outperform existing methods for both real-world and synthetic data streams.

In summary, this paper is one of the first to discuss local explainability and, in particular, local feature attributions in evolving data streams. We propose a powerful and flexible framework capable of recognizing local and global changes in the online learning and attribution model. In this way, our work enables more meaningful and robust use of local feature attribution methods in data streams and is thus an important step towards better explainability in the practical domain of online machine learning.

In Section 2, we introduce related work. In Section 3, we formally examine the behavior of local attributions under concept drift. We then present CDLEEDS and an effective implementation in Section 4. Finally, in Section 5, we demonstrate CDLEEDS in several experiments on binary and multiclass tabular streaming data sets.

2 RELATED WORK

In the following, we briefly outline related work on local attribution methods, explainability in data streams and concept drift detection.

Local Feature Attributions. Local attribution methods are one of the most popular and widely used explanation techniques. Most frameworks are based on a similar intuition: a complex (black-box) predictive model can be locally approximated sufficiently well by a much simpler, usually linear, explanation model. Model-agnostic frameworks like LIME [42] and SHAP [34] belong to the most popular attribution methods and have inspired a variety of follow-up work [1, 13, 29, 33, 47]. Additionally, there are model-specific attribution methods that exploit the inner mechanics of the complex model. In particular, a large literature has formed on gradient-based attribution techniques for Deep Neural Networks [3, 31, 45, 48]. For more detailed information, we refer to recent surveys [2, 4, 12, 22].

Explainability in Data Streams. Compared to the explanation of offline (black-box) models, relatively little attention has been paid to the explainability of predictions in dynamic data streams. Bosnić et al. [10] were among the first to describe that explanations in a data stream must actually consist of a series of individual explanations that can change over time. Demšar and Bosnić [14] later argued that the dissimilarity of periodically generated feature attributions may be used to detect concept drift. Finally, Tai et al. [50] briefly discuss feature-based explainability in the context of sketching. Still, given the abundance of streaming applications in practice, explainable machine learning for data streams should receive more attention.

Concept Drift Detection. Concept drift detection has traditionally served as a tool to prevent deterioration in predictor performance following distributional changes. As such, global concept drift detection methods have been integrated into state-of-the-art online learning frameworks, such as the Hoeffding Tree [8]. Most modern drift detection methods are based on changes in the observed predictive error of the online learning model [5, 6, 18]. In this context, many approaches use sliding windows for more robust or statistically significant drift detection [7, 39, 43]. Alternatively, a more recent approach monitors changes in the inherent uncertainty of model parameters to detect global and feature-specific (partial) concept drift [25]. For more information about global concept drift detection, we refer to the comprehensive surveys of Gama et al. [20], Gonçalves Jr et al. [21], Webb et al. [52], Zliobaite [54].

In contrast, the potential explanatory power of concept drift detection has been largely neglected. Accordingly, there are only few methods capable of local concept drift detection. For example, Gama and Castillo [17] integrate an error-based concept drift detection scheme into the inner nodes of an incremental decision tree. In this way, they are able to detect concept drifts in specific input regions, represented by the branches of the tree. However, as mentioned earlier, we need a mechanism that is able to detect instance-level changes to enable better explainability in a data stream.

3 ONLINE LEARNING AND LOCAL ATTRIBUTION – FORMAL INTRODUCTION

Data streams are defined by a series of time steps $1, \dots, t, \dots, T$. At each time step, we obtain an observation $x_t \in \mathbb{R}^m$ and a corresponding label $y_t \in \mathbb{R}$, where m is the number of features. Our goal is to incrementally train an online predictive model $f_{\theta_t}(x_t)$. That is, we aim to optimize the model parameters θ_t at every time step t given the new training observation (we may also use batches of training data). Since the parameters θ_t are defined by the selected model, we write $f_{\theta_t}(x_t) = f_t(x)$ to simplify the exposition.

We may represent the observations and labels by two corresponding random variables X and Y. The data generating concept at time step t is defined by the joint probability distribution $P_t(Y, X)$. Typically, we assume that the observations are drawn independently from the data generating distribution. Although this independence assumption can be violated in practice, it has proven effective in many applications [26]. Concept drift describes a change of the joint probability between two time steps t_1 and t_2 [52], i.e.,

$$P_{t_1}(Y, X) \neq P_{t_2}(Y, X) \Leftrightarrow P_{t_1}(Y|X)P_{t_1}(X) \neq P_{t_2}(Y|X)P_{t_2}(X). \quad (1)$$

In general, we distinguish two fundamental types of concept drift. *Real* concept drift corresponds to a change in the conditional probability distribution $P_t(Y|X)$, while $P_t(X)$ remains stable. Conversely, *virtual* concept drift describes a shift in $P_t(X)$, while $P_t(Y|X)$ remains unchanged. Other than real concept drift, virtual concept drift does not change the optimal decision boundary. In practice, we are therefore mostly interested in real concept drift, i.e. $P_{t_1}(Y|X) \neq P_{t_2}(Y|X)$. Nevertheless, virtual concept drift may affect the (learned) decision boundary of our online learning model [37]. This effect can be seen in our introductory example in Figure 1b. Moreover, we can distinguish between local and global concept drift. While global concept drift affects the entire (or large regions) of the input space, local concept drift is locally bounded. Hence, it is often more difficult to detect local concept drift.

In practice, the true data generating distribution is usually unknown. Hence, the online predictive model is often our best approximation of the active concept [25]. That is, we typically assume that the predictive model at time step t approximates the conditional target probability well, i.e. $P_t(Y|X) \approx f_t(x)$. This simplifying assumption is the fundamental basis of most existing concept drift detection methods [5, 7, 25]. Therefore, instead of explicitly learning the true data generating distribution, we can detect concept drift directly from a change in the predictive model:

$$f_{t_1}(x) \neq f_{t_2}(x) \tag{2}$$

Notably, since Eq. (2) allows us to detect concept drift based on changes in the decision boundary, we can also detect the changes caused by virtual concept drift as described above.

3.1 Local Attribution Accuracy and Its Implication for Online Learning

Local attribution methods allow us to explain complex predictive models by quantifying the local importance of input features in the prediction. Let $\phi_{x_i,f} \in \mathbb{R}^m$ be the local attribution vector corresponding to an observation x_i and a model f. Typically, the generated feature attribution vector $\phi_{x_i,f}$ adheres to a set of sensible properties. A fundamental property shared by most state-of-the-art attribution methods is *local accuracy* [34], also known as *local fidelity* [42] or *summation to delta* [45].

Local accuracy describes that the attribution vector must account for the difference between the local model outcome and a baseline value. We can adopt the generic definition of Lundberg and Lee [34] for the online case and define local accuracy accordingly as

$$f_t(x_i) = \phi_t^0 + \sum_{j=1}^{m} \phi_{x_i,f_t}^j, \tag{3}$$

where $\phi_t^0 \in \mathbb{R}$ is the baseline outcome at time step t and ϕ_{x_i,f_t}^j is the attribution of feature j. Note that Lundberg and Lee [34] used an additional vector representation of missing features. In general, however, we can assume that the observation to be explained has no missing features.

The baseline value ϕ_t^0 is set to represent missing discriminative information, i.e., ideally it is a value for which the prediction is neutral. For example, in image recognition, the zero vector is often used as a baseline [48]. Alternatively, we might use the expectation $\phi^0 = \mathbb{E}_{X^0}[f(x)]$ over a static sample of training observations X^0 as our baseline [34]. The choice of the baseline can drastically alter the generated attributions and should thus be selected with care [28]. In particular, for data streams where our understanding of missingness may change over time, we might need to update ϕ_t^0 between time steps. We propose an effective baseline in Section 4.3.

In the introductory experiments, we have shown that local feature attributions may lose their validity due to incremental model updates and concept drift. With the above definitions in place, we can now express this behavior in more formal terms. Suppose the predictive model has changed between two time steps t_1 and t_2 according to (2). We know that there must exist at least one data point x_i such that $f_{t_1}(x_i) \neq f_{t_2}(x_i)$. By definition of local accuracy, a shift of the local model outcome $f_t(x_i)$ implies a shift of the baseline ϕ_t^0 and/or the local attribution vector ϕ_{x_i,f_t} and vice versa:

$$f_{t_1}(x_i) \neq f_{t_2}(x_i) \overset{(3)}{\Leftrightarrow} \phi_{t_1}^0 + \sum_{j=1}^{m} \phi_{x_i,f_{t_1}}^j \neq \phi_{t_2}^0 + \sum_{j=1}^{m} \phi_{x_i,f_{t_2}}^j \tag{4}$$

In other words, any change in the decision boundary of the model, e.g., due to concept drift or incremental updates, is guaranteed to change either the baseline and/or at least one local attribution score. Therefore, as before, we argue that local attribution methods in data streams need a mechanism to detect such local changes in order to provide meaningful explanations over time.

4 THE CDLEEDS FRAMEWORK

We present CDLEEDS, a novel framework for local change detection that allows us to identify outdated attributions and enable more efficient and targeted recalculations for temporally adjusted explanations in data streams. In general, our goal is to identify whether a local attribution vector ϕ_{x_i,f_t} has changed between two time steps t_1 and t_2. Based on the local accuracy property, we can immediately formulate a naïve scheme for local change detection:

$$\sum_{j=1}^{m} \phi_{x_i,f_{t_1}}^j \neq \sum_{j=1}^{m} \phi_{x_i,f_{t_2}}^j \overset{(3)}{\Leftrightarrow} f_{t_1}(x_i) - \phi_{t_1}^0 \neq f_{t_2}(x_i) - \phi_{t_2}^0. \tag{5}$$

By calculating the right part of Eq. (5) for a given observation x_i in all successive time steps, we are able to detect local change over time. Indeed, since we only require the baseline ϕ_t^0 and model outcome $f_t(x_i)$, *this simple method allows us to detect local changes without calculating a single attribution vector.* However, this approach may be too costly if we want to detect changes for a large number of observations (because we would have to repeatedly obtain predictions $f_t(x_i)$, e.g., for an entire user base). Moreover, since we are comparing snapshots at individual time steps, this naïve approach might be prone to noise. Therefore, we need to modify this basic change detection method to make it more reliable and efficient.

4.1 Spatiotemporal Neighborhoods

In practice, it may often be sufficient to detect changes in the close proximity of a given observation. Specifically, if we can detect concept drift in the neighborhood of an observation x_i with high confidence, it is likely that the attribution of x_i has changed. To this end, we need a meaningful understanding of neighborhood in data streams. Intuitively, we would like a neighborhood to include close previous observations. In this context, we introduce the notion of *spatiotemporal neighborhood:*

Definition 1 (Spatiotemporal γ-Neighborhood (STN)). *Let $sim(\cdot)$ be a sensible similarity measure (e.g., cosine similarity or RBF kernel). A spatiotemporal γ-neighbourhood with respect to an observation x_i is defined by a set of time steps $\Omega^{(x_i,\gamma)} = \{t \in \{1,\ldots,T\} \mid sim(x_t, x_i) \geq \gamma\}$.*

More intuitively, a spatiotemporal γ-neighborhood, STN for short, is a set of time steps corresponding to previous observations similar to the observation in question. With the parameter γ we can control the minimal similarity and thus the boundedness of the STN. If we are able to detect changes in the STN of an observation x_i with reasonably large γ, we can assume that the attribution of that observation has changed. Accordingly, we can rephrase the naïve change detection method from Eq. (5) in a more robust way:

$$\mathbb{E}_{u \in \Omega_{<t}^{(x_i,\gamma)}}[f_u(x_u) - \phi_u^0] \neq \mathbb{E}_{v \in \Omega_{\geq t}^{(x_i,\gamma)}}[f_v(x_v) - \phi_v^0], \tag{6}$$

where $\Omega_{<t}^{(x_i,\gamma)} = \{u \in \{1,\ldots,t-1\} \mid \mathrm{sim}(x_u, x_i) \geq \gamma\}$ and $\Omega_{\geq t}^{(x_i,\gamma)} = \{v \in \{t,\ldots,T\} \mid \mathrm{sim}(x_v, x_i) \geq \gamma\}$ denote the STNs of x_i for different intervals before and after the time step t. With Eq. (6), we can now compare time intervals instead of individual snapshots, which usually leads to more robust and reliable detections. Note that we

can scale the time intervals, and hence the size of the STNs, by limiting the set of relevant time steps. For example, to obtain the STN in an interval of size w before time step t, we can specify $\Omega_{<t}^{(x_i,\gamma)} = \{u \in \{t-w,\dots,t-1\} \mid \text{sim}(x_u,x_i) \geq \gamma\}$. Similar to existing concept drift detection methods that use sliding windows (see Section 2), the size of the specified time intervals affects the performance. If the interval is chosen too small, the method may not be robust and produce false alarms. On the other hand, if the interval is chosen too large, certain changes may be missed. In order to achieve a higher degree of flexibility, we therefore only limit the maximum size of an STN in our implementation, but not the eligible time intervals.

In order to detect local change over time, we can incrementally update the STNs $\Omega_{<t}^{(x_i,\gamma)}$ and $\Omega_{\geq t}^{(x_i,\gamma)}$. As a result, we avoid having to consider (predict) old observations repeatedly, which considerably reduces the resource consumption compared to the initial naïve scheme. In fact, since we can process observations in a single pass, we fulfill a central requirement of online machine learning [15].

Instead of comparing expectations directly, as shown in Eq. (6), we may also use a hypothesis test to detect significant changes over time. Note that we have assumed independent streaming observations (see Section 3). Moreover, the expectations in Eq. (6) tend to be normally distributed for large sample sizes, i.e., for large STNs, according to the central limit theorem. Therefore, if we specify reasonably large STNs, we may apply the unpaired two-sample t-test (which we did in our implementation).

4.2 Finding Representative Neighborhoods With Adaptive Hierarchical Clustering

Data streams produce a large number of observations for which we may need to detect changes in the explanation. Although Eq. (6) provides an efficient mechanism for detecting changes at a point x_i, the construction of STNs for all observations to be explained can lead to a high computational cost. For practical reasons, we may instead select a representative set of observations for which we maintain STNs over time, which in turn serve as an approximation to the STNs of all observations. Specifically, since we are interested in grouping similar data points according to Definition 1, we aim to identify a set of representative observations $C_t = \{c_1,\dots,c_n,\dots,c_N\}$, such that each c_n is similar to a large group of current observations. This problem is very similar to online clustering [11, 53], where each c_n denotes the centroid of a cluster $\Gamma_{c_n} = \{x_i \mid \text{sim}(x_i,c_n) \geq \gamma\}$. Accordingly, if we obtain an STN with respect to c_n in a given interval, e.g. $\Omega_{<t}^{(c_n,\gamma)}$, we can assume that it is also representative of all observations in the cluster, and hence

$$\forall x_i \in \Gamma_{c_n} : \mathbb{E}_{u\in\Omega_{<t}^{(x_i,\gamma)}}[f_u(x_u)-\phi_u^0] \approx \mathbb{E}_{u\in\Omega_{<t}^{(c_n,\gamma)}}[f_u(x_u)-\phi_u^0]. \quad (7)$$

Note that Eq. (7) holds equivalently for $\Omega_{\geq t}^{(c_n,\gamma)}$.

On this basis, we propose a simple hierarchical and dynamic clustering of observations in a binary tree. The root of the clustering tree contains all observations from a specified interval, implemented as a sliding window. The centroid corresponds to the mean value of these observations. If the similarity radius of the current node is smaller than γ, we split the node by choosing the two most dissimilar data points as the new children (i.e., a binary split). The observations of the parent node are then assigned to the most similar child. We continue the procedure for the children recursively until the similarity radius for each leaf node is greater or equal γ.

Virtual concept drift can shift high-density regions in the input space (see Figure 1). Therefore, the clustering should adjust accordingly. For this purpose, we maintain an internal age counter for each node, which is updated as soon as the node receives a new

Algorithm 1 update() - General update procedure at a node n of the CDLEEDS hierarchical clustering approach.

```
Require: Observation $x_t$; Prediction-baseline difference $\hat{y}_t - \phi_t^0$.
    *** A node comprises an age counter, a sliding window of observations used for clustering, a sliding window of prediction-baseline differences used for change detection, and a centroid. ***
    *** The sliding windows $W_n$, $V_n$ have a user-defined size and correspond to an STN at the centroid, i.e. $\Omega^{(c_n,\gamma)} = \{u \in \{t-w,\dots,t\} \mid sim(x_u,c_n) \geq \gamma\}$. ***
1:  $\text{age}_n \leftarrow \text{age}_n + 1$
2:  $W_n \leftarrow$ Remove oldest entry and append $x_t$.
3:  $V_n \leftarrow$ Remove oldest entry and append $\hat{y}_t - \phi_t^0$.
4:  $c_n \leftarrow \text{mean}(W_n)$

5:  if $n$ is a leaf node then
6:      if $\exists x_u \in W_n : \text{sim}(x_u,c_n) < \gamma$ then
7:          *** Split the node by using the most dissimilar points in $W_n$ as the centroids of the new children. ***
8:          $n_{\text{left}}, n_{\text{right}} \leftarrow$ Split the node $n$.
9:          *** Assign each observation to the closest child node. ***
10:         for $x_u \in W_n$ do
11:             $n_{\text{child}} \leftarrow \underset{[n_{\text{left}},n_{\text{right}}]}{\arg\max}\left(\text{sim}(x_u,c_{n_{\text{left}}}),\text{sim}(x_u,c_{n_{\text{right}}})\right)$
12:             $n_{\text{child}}.\text{update}(x_u,\ \hat{y}_u - \phi_u^0)$
13:             $\text{age}_{n_{\text{child}}} \leftarrow \text{age}_n$
14:         end for
15:     else
16:         *** Identify change at the node by testing for a significant difference in $V_n$. According to Eq. (6), we compare the means of the first and second (equally sized) halves of $V_n$.***
17:         $\bar{V}_n^* = \text{mean}(V_n[:|V_n|/2])$
18:         $\bar{V}_n^{**} = \text{mean}(V_n[|V_n|/2:])$
19:         if $h_0 : \bar{V}_n^* = \bar{V}_n^{**}$ can be rejected for significance $\alpha$ then
20:             Alert local change at $n$.
21:         end if
22:     end if
23: else
24:     *** Forward $x_t$ to the closest child. ***
25:     $n_{\text{child}} \leftarrow$ see line 11.
26:     $n_{\text{child}}.\text{update}(x_t,\ \hat{y}_t - \phi_t^0)$
27:     $\text{age}_{n_{\text{child}}} \leftarrow \text{age}_n$
        *** Check if the split is outdated and should be pruned. ***
28:     if $\text{age}_n - \min(\text{age}_{n_{\text{left}}},\ \text{age}_{n_{\text{right}}}) \geq$ threshold then
29:         Prune the branch at $n$ and make $n$ a leaf node.
30:         Test for change at $n$ as in line 16 - 21.
31:     end if
32: end if
```

observation. If a child node has not received any observations for a while, its age differs from the age of the parent node, indicating an outdated split that can be pruned.

To identify change for a given observation, we then only need to retrieve the most similar leaf node of the current tree and test for change as specified in Eq. (6) using the STNs of the corresponding centroid (see Eq. (7)). The general procedure at a node of the tree is described in Algorithm 1. A corresponding implementation is available at https://github.com/haugjo/cdleeds.

The proposed hierarchical clustering provides clusters with increasing granularity. In the context of explainable online learning, this is an advantage as we are able to detect change at different hierarchies. For example, to detect global change, we can combine the test results of leaf nodes with Fisher's method [16]. In this context, we would correct the significance level α for multiple hypothesis testing, using the mean false discovery rate $\alpha_{\text{corr}} = \alpha(N+1)/(2N)$, where N is the number of independent tests at the leaf nodes.

4.3 Further Algorithmic Decisions

The generic clustering method proposed above requires us to make choices during implementation. A central component of the clustering is the similarity measure. The cosine similarity and the (negative) Euclidean distance are commonly used to measure the similarity of vectors. However, the cosine similarity is a measure of orientation and does not take into account the magnitude of the input features, which are relevant for local attributions. Conversely, the Euclidean distance is very sensitive to the dimensionality and magnitude of a vector. This can make it difficult to establish a meaningful threshold γ - especially since dimensionality and magnitude can change in practice due to concept drift. For this reason, we use the Radial Basis Function (RBF) kernel in our implementation (with variance parameter $1/m$, where m is the number of features). The RBF kernel ranges from zero to one (when the vectors are equal) and is frequently used as a measure of similarity in machine learning. Due to the boundedness of the RBF kernel, it is generally much easier to specify and interpret the parameter γ.

Moreover, we use the exponentially weighted moving average to obtain our baseline, i.e., $\phi_t^0 = f_t(\text{EWMA}_t)$ with $\text{EWMA}_t = \beta x_t + (1-\beta)\text{EWMA}_{t-1}$, where $\beta \in [0,1]$ is the decay factor. Compared to using a static sample of observations [28, 34], the EWMA has the advantage of reducing the weight of old observations over time. In this way, our baseline automatically adjusts to concept drift.

4.4 Complexity and Limitations

The memory complexity of the proposed hierarchical clustering is $O(K_t w)$, where w is the size of the sliding windows and K_t is the number of nodes at time step t. Moreover, the computational complexity of constructing the hierarchical clustering for T data points is $O(T \log T)$. Accordingly, CDLEEDS has a higher resource consumption than existing methods for global concept drift detection. However, the proposed framework is much more powerful because it can detect both global and local change.

The selection of an appropriate similarity threshold γ is not trivial. If we set γ too small, we get large neighborhoods that do not capture local behavior. If we set γ too high, the cluster tree may become too deep to be maintained in a real-time application. To address the latter problem, decision tree algorithms often specify a maximum depth. Limiting the depth may result in STNs at leaf nodes that have a lower similarity than originally specified by γ (because we cannot further partition the observations). However, to enable more efficient computations, it can often be useful to limit the maximum size of the cluster tree. For example, if we want to use CDLEEDS for global change detection, we do not need the same local granularity as for local change detection.

Similarly, it can be difficult to set a reasonable significance level for hypothesis testing. If the significance level is too small, we might miss certain concept drifts. On the other hand, if the significance level is too high, we might produce many false alarms. However, in our experiments, we obtained good results for common significance levels such as 0.01 and 0.05.

Concept drift detection methods are sensitive to hyperparameter settings, and CDLEEDS is no exception. Therefore, it is generally advisable to perform hyperparameter optimization on an initial, stationary training set to learn what degree of variation to expect under a reasonably stable data concept. In addition, it can be useful to re-evaluate the initial hyperparameters at regular intervals.

Table 1: *Data Sets.* We used popular and open-sourced classification data sets in our experiments (obtained from openml.org, original sources are included where available). TüEyeQ [30] and Insects [46] comprise natural concept drift. We induced the remaining real-world streaming data sets with artificial concept drift [44]. Finally, we generated synthetic data streams with scikit-multiflow [35] (indicated by "(s.)").

Name	#Samples	#Features	# Classes	Data Types	Drift Types
TüEyeQ [30]	15,762	77	2	cont., cat.	abrupt
Bank-Marketing [36]	45,211	16	2	cont., cat.	abrupt
Electricity [23]	45,312	8	2	cont., cat.	abrupt
Adult [32]	48,840	54	2	cont., cat.	abrupt
Airlines	539,383	7	2	cont., cat.	abrupt
KDD Cup 1999	494,020	41	23	cont., cat.	abrupt
Covertype [9]	581,012	54	7	cont., cat.	abrupt
Insects [46]	355,275	33	6	cont.	abrupt
SEA (s.)	500,000	3	2	cont.	abrupt
Agrawal-Gradual (s.)	500,000	9	2	cont.	gradual
Agrawal-Mixed (s.)	500,000	9	2	cont.	abrupt, gradual

5 EXPERIMENTS

We evaluated the proposed framework in three experiments. In Section 5.3, we show that the proposed hierarchical clustering algorithm is able to identify meaningful spatiotemporal neighborhoods and adapt to local virtual concept drift. In Section 5.4, we demonstrate that CDLEEDS can be used to reduce the number of recalculations of local attributions over time. In this context, we also illustrate the local change detection of CDLEEDS. Finally, in Section 5.5, we compare CDLEEDS to state-of-the-art methods for global concept drift detection. For illustration, we used a binary and multi-class classification setting, which is well handled by most drift detectors. If not mentioned otherwise, we trained a Hoeffding Tree with adaptive Naïve Bayes models at the leaf nodes in the default configuration of scikit-multiflow [35]. All models and experiments were implemented in Python (3.8.5) and run on an AMD Ryzen Threadripper 3960X CPU with 128GB RAM under Ubuntu 18.04.

5.1 Data Sets

Unfortunately, there are few real-world data sets with known concept drift. Therefore, one usually has to rely on synthetically generated streaming data to evaluate concept drift detection approaches [27]. In our experiments, we used a mixture of popular real-world data sets with both natural and synthetic concept drift, as well as synthetic streaming data sets. We normalized all data sets before use. A list of the data sets and their properties is shown in Table 1.

One of the few real-world data sets with natural and known concept drift is TüEyeQ [30], which we already mentioned in the introduction. Recently, Souza et al. [46] presented several data sets with sensor measurements of flying insect species. The classification task is to identify the correct insect. By changing environmental parameters such as humidity and temperature, Souza et al. [46] produced different types of concept drift. In our experiment, we used the unbalanced Insect data set with abrupt concept drift.

Moreover, we imputed popular real-world streaming data sets obtained from openml.org with synthetic concept drift. In particular, we adopted the method due to Sethi and Kantardzic [44] based on the Mutual Information. Specifically, in order to simulate concept drift, we randomly permuted the values of the top 50% of features with highest Mutual Information with the target. We repeated the procedure to generate multiple synthetic drifts per data set.

We also generated synthetic data streams using scikit-multiflow [35]. In particular, we generated data streams with abrupt concept drift (SEA), gradual concept drift (Agrawal-Gradual), and mixed, i.e., abrupt and gradual, concept drift (Agrawal-Mixed). We did not balance the classes of the generated data sets and specified *perturbation=0.1* for all generators. Otherwise, we used the default configuration of scikit-multiflow.

5.2 Hyperparameters for CDLEEDS

We performed a grid search on the Bank-Marketing data set to identify hyperparameters for CDLEEDS. To obtain unbiased results, we used the same set of hyperparameters in all experiments. Specifically, we set the similarity threshold to $\gamma = 0.95$, the significance level of the t-test to $\alpha = 0.01$, the decay factor of the EWMA-baseline to $\beta = 0.001$, the maximum size of the sliding windows (STNs) to 200 observations, and the maximum age of a node to 100 observations before pruning. If not mentioned otherwise, we limited the depth of the hierarchical clustering to 5.

5.3 1st Experiment - CDLEEDS Clustering Under Local Virtual Concept Drift

In a first experiment, we investigated the ability of the hierarchical clustering method to adapt to local virtual concept drift. Figure 3 shows the TSNE representation of the clustering for two exemplary data sets. Specifically, we collected samples over two time intervals (upper/lower plots). After the first time interval, we simulated a local virtual concept drift. That is, we ignored all new observations that would have been assigned to the red cluster and continued the online training without these observations. In this way, we tested the ability of CDLEEDS to identify and prune obsolete leaves and branches. We limited the maximum depth of the cluster tree to 3 for this experiment. Notably, Figure 3 shows that CDLEEDS managed to form meaningful clusters over time. Moreover, our age-based pruning strategy was able to correctly identify the obsolete (red) cluster. We observed similar results for all remaining data sets.

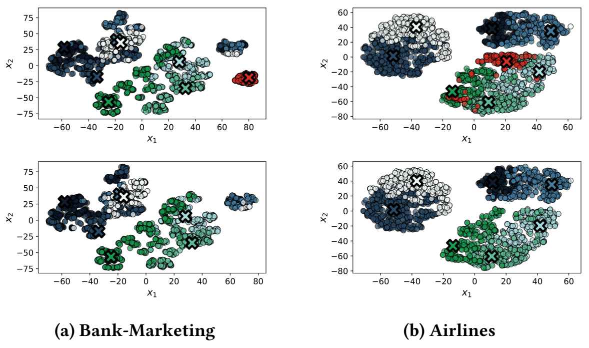
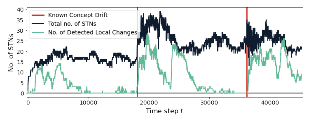


Figure 3: *CDLEEDS Clustering - Adjusting to Local Virtual Concept Drift.* We trained CDLEEDS on 4,000 observations from two real-world data streams. After processing the first 2,000 observations, we simulated a local virtual concept drift by ignoring all observations that would have fallen into the red cluster during the rest of the training process. Above, we depict the TSNE representation [51] of observations before (upper) and after (lower) the concept drift. The learned clusters are indicated by different colors. The centroids are shown as corresponding "x" markers. Notably, the proposed clustering method was able to learn meaningful (i.e., spatially coherent) clusters for both data sets over time.

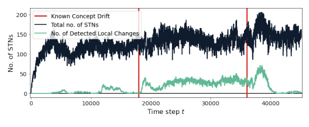
5.4 2nd Experiment - Local Change Detection for More Efficient Feature Attributions

CDLEEDS is a local change detection framework that can help make local attribution methods in data streams more feasible. In this experiment, we demonstrate the ability of the proposed framework to detect local changes, in particular those caused by concept drift. Figure 4 illustrates the number of spatiotemporal γ-neighborhoods, i.e., leaf nodes, maintained by CDLEEDS for four exemplary data sets. We also show the number of detected local changes over time. We did not limit the maximum depth of the hierarchical clustering for this experiment. Consequently, at each time step, all leaf nodes corresponded to valid STNs with $\gamma = 0.95$ as defined in Def. 1.

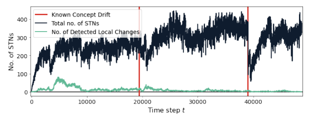
Based on Figure 4, we can make several interesting observations. As in the previous experiment, we find that the hierarchical clustering method is able to adapt to concept drift by pruning obsolete leaves and branches or creating new ones. This adaptation is most evident after the last concept drift in the Adult data set. Notably, during this period, CDLEEDS issued only few local change alerts, suggesting that the last concept drift in Adult is a virtual rather than a real concept drift. Early change detections can generally be attributed to the initial training of the predictive model and cluster tree and would thus be ignored in practice. In general, the known concept drifts are accompanied by a substantial increase in the detected local drifts. Moreover, Figure 4 shows that there is usually relatively little local change when the data generating distribution is stable.



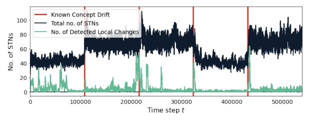
(a) Electricity



(b) Bank-Marketing

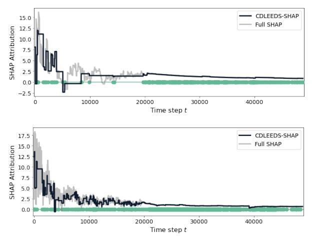


(c) Adult

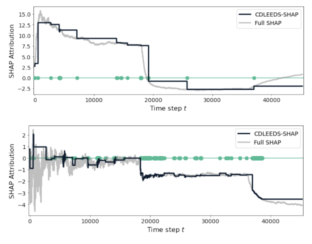


(d) Airlines

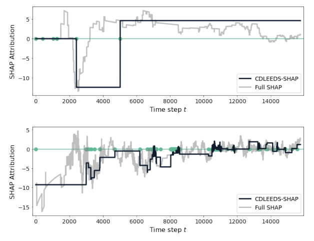
Figure 4: *Local Change Detection with CDLEEDS.* We show the CDLEEDS clustering on 4 data sets. In particular, we depict the total number of STNs over time (blue) and the number of STNs for which we detected local change (green). The red vertical lines indicate known global concept drifts. In all cases, the CDLEEDS clustering changed in complexity after a concept drift and/or detected an increasing number of local changes. Yet, concept drift often only affected a subset of the STNs. Moreover, in times of stable data concepts, there were usually only a few STNs underlying changes (caused by the continued incremental model updates). With this insight, we could considerably reduce the number of local attributions that have to be recalculated after each update.



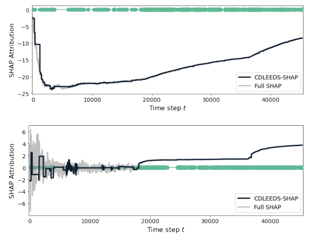
(a) Adult - Reduction of recalc.: 92.82% upper plot, 62.40% lower plot, 80.05% ± 0.53% on average (100 obs.).



(b) Bank-Mark. - Reduction of recalc.: 99.85% upper plot, 88.77% lower plot, 95.71% ± 0.05% on average (100 obs.).



(c) TüEyeQ - Reduction of recalc.: 99.96% upper plot, 89.90% lower plot, 95.97% ± 0.12% on average (100 obs.).



(d) Electricity - Reduction of recalc.: 70.26% upper plot, 51.23% lower plot, 56.11% ± 0.11% on average (100 obs.).

Figure 5: *Feasible Local Attributions in Data Streams with CDLEEDS.* From a random sample of 100 observations per data set, we show the SHAP attributions [34] of the two observations that required the fewest (top) and most (bottom) recalculations over time. For reasons of readability, we only display the attribution of the feature with the largest average value. However, note that we observed similar results for all observations and features. The blue line indicates the SHAP attributions that we obtained by using CDLEEDS to trigger recalculations (detected local changes are indicated by green markers). The grey background pattern corresponds to the actual SHAP attribution at each time step. By using CDLEEDS, the average number of recalculations can be significantly reduced without affecting the explanatory power compared to the actual feature attributions.

We can use this insight to make the recomputation of local attributions more efficient. In particular, we argue that it is usually sufficient to recompute old attributions when there has been a corresponding local change. To support our argument, we computed SHAP attributions [34] for four different data streams. This time we used a logistic regression model, as the SHAP implementation for linear models is much more efficient. Also, we only performed this experiment on the small data sets, as calculating and storing SHAP values for large data streams is not feasible on most machines.

We conducted the experiment as follows: At time step $t = 0$, we computed the SHAP attributions for a random sample of 100 observations. We then recomputed the SHAP attribution of an observation in subsequent time steps, only if the observation had been assigned to a new leaf in the hierarchical clustering, or if a corresponding local change had been detected. Figure 5 shows SHAP attributions for two observations in each data stream. Strikingly, the CDLEEDS-based recomputations approximate the actual SHAP attributions well. Indeed, for the entire sample of 100 observations, we observed an average deviation from the true SHAP attribution of only 0.10 ± 0.15 for Adult, 0.16 ± 0.10 for Bank-Marketing, 0.45 ± 0.29 for TüEyeQ, and 0.11 ± 0.11 for Electricity. Given attributions of up to 17.5 (see Adult) or -24 (see Electricity), these deviations become negligible. At the same time, CDLEEDS was able to considerably reduce the number of recalculations. We observed an average reduction in recalculations of 80.05% ± 0.53% for Adult, 95.71% ± 0.05% for Bank-Marketing, 95.97% ± 0.12% for TüEyeQ, and 56.11% ± 0.11% for Electricity (in % of all time steps).

Our experiments show that periodic recalculations of local attributions are generally necessary, since attributions can change considerably in the streaming setting. However, the number of recalculations actually performed can be significantly reduced through CDLEEDS. In addition, the detected changes, along with the recalculated attributions, may carry valuable explanatory information. For example, in Figure 5(c), drastic changes in the local attributions and corresponding alerts by CDLEEDS indicate a sudden concept drift around $t = 2500$ (which might be due to a new, e.g., more difficult, type of IQ task [30]). In general, as claimed above and shown in our experiments, CDLEEDS can make local attribution-based explainability in data streams more efficient and expressive.

5.5 3rd Experiment - Using CDLEEDS for Global Concept Drift Detection

CDLEEDS is designed as a framework for detecting local change. However, as suggested in Section 4.2, we might also use CDLEEDS to detect global concept drift by applying a simple strategy based on Fisher's method for combining p-values. For the sake of completeness, we thus compared CDLEEDS with several state-of-the-art global concept drift detection models. In particular, we compared our approach to ADWIN [7], DDM [18], ECDD [43], MDDM-A [40] and RDDM [6], all of which attempt to detect concept drift through changes in the error rate. Moreover, we compared CDLEEDS to ERICS [25], which detects concept drift by monitoring changes in the distributions of model parameters. We used the original ERICS implementation provided by the authors [25]. The remaining implementations are openly available via the tornado package [38]. We applied the same hyperparameter search as for CDLEEDS. Accordingly, we specified *delta* = 0.1 for ADWIN [7], as well as *window_mvg_average* = 90 and *beta* = 0.001 for ERICS [25]. Other than that, however, we could use the default hyperparameters.

5.5.1 Evaluation Measures. We examined the delay, recall, and false discovery rate (FDR) of each drift detection method [27]. The delay corresponds to the time steps until a known concept drift is first detected. The recall quantifies the proportion of known concept drifts that the model detected. And the FDR is the proportion of false positives among all detected drifts. Note that only the combination of recall and FDR provides a meaningful evaluation, as each measure can be easily optimized on its own. Therefore, we report the mean of recall and (1-FDR) in Table 2. To compute the recall and FDR, we need to define a time interval after known concept drift in which we count a drift alert as a true positive. In the experiments, we used several intervals with lengths between 1% and 10% of the original data set size. Table 2 shows the means and standard deviations for the different interval sizes. In addition, to give the classifier time for initial training, we did not include drift alerts that occurred within the first 1,000 observations.

5.5.2 Results. In general, we find that there are considerable differences between all concept drift detection methods and data sets, both regarding the combined recall and FDR in Table 2 and the delay in Table 3. As described above, this effect might be mitigated by (periodically) optimizing the hyperparameter configurations for each data set. However, such performance differences can often be observed in practice because concept drift detection methods are usually sensitive to the predictive model and data distribution at hand. For this reason, it is generally advisable to use multiple methods in parallel for more robust global concept drift detection.

Naturally, the more elaborate drift detectors CDLEEDS (2.32 milliseconds) and ERICS [25] (3.68 ms) had a larger average update time than the error rate-based drift detectors (ADWIN [7] = 0.14 ms, DDM [18] = 0.12 ms, ECDD [43] = 0.12 ms, MDDM-A [40] = 0.14 ms, and RDDM [6] = 0.12 ms). However, the computation times should be treated with care, as they generally depend on the implementation and hardware configuration at hand.

Despite its relative simplicity, the CDLEEDS-based approach to global concept drift detection competes with powerful existing methods such as ADWIN [7], ECDD [43] or ERICS [25]. In particular, CDLEEDS received the best average score and the second best average ranking on the combined recall and FDR measure. At the same time, CDLEEDS was usually also able to achieve a short delay. In summary, our results suggest that CDLEEDS, although being originally designed for local change detection, might also be a valuable alternative to popular global drift detection methods.

Table 2: *Global Concept Drift Detection – Part 1.* CDLEEDS was developed for local change detection, in particular to detect obsolete local attributions. However, as a byproduct, CDLEEDS can also be used to detect global concept drift. The results of this additional experiment are shown in this and the following table. The missing values correspond to test runs in which a method did not raise an alert. ERICS [25] can only process binary target variables, so no results are available for the multi-class data sets. Here we show the mean of recall + (1 - false discovery rate) (%; mean ± standard deviation; higher is better). Strikingly, CDLEEDS can compete with state-of-the-art methods for concept drift detection.

	CDLEEDS	ERICS	ADWIN	DDM	ECDD	MDDM-A	RDDM
TüEyeQ	0.57 ± 0.04	0.61 ± 0.04	0.50 ± 0.17	0.38 ± 0.16	0.57 ± 0.18	0.28 ± 0.17	0.36 ± 0.26
Bank-Mark.	0.58 ± 0.01	0.56 ± 0.03	0.45 ± 0.00	-	0.56 ± 0.03	0.75 ± 0.00	0.63 ± 0.00
Electricity	0.40 ± 0.00	0.56 ± 0.02	0.45 ± 0.06	-	0.26 ± 0.00	0.00 ± 0.00	0.25 ± 0.00
Adult	0.66 ± 0.04	0.57 ± 0.03	0.41 ± 0.24	0.25 ± 0.00	0.56 ± 0.02	0.00 ± 0.00	0.07 ± 0.00
Airlines	0.62 ± 0.06	0.61 ± 0.06	0.70 ± 0.07	0.63 ± 0.00	0.57 ± 0.09	0.37 ± 0.05	0.35 ± 0.00
SEA	0.60 ± 0.09	0.19 ± 0.01	0.65 ± 0.05	0.85 ± 0.09	0.60 ± 0.05	0.78 ± 0.08	0.83 ± 0.03
Agrawal-Grad.	0.68 ± 0.02	0.70 ± 0.03	0.75 ± 0.02	0.42 ± 0.00	0.70 ± 0.02	0.67 ± 0.00	0.32 ± 0.01
Agrawal-Mix.	0,55 ± 0.12	0.62 ± 0.04	0.69 ± 0.04	0.55 ± 0.00	0.63 ± 0.04	0.65 ± 0.06	0.40 ± 0.00
KDD Cup	0.59 ± 0.13	-	0.45 ± 0.21	0.12 ± 0.00	0.61 ± 0.05	-	0.40 ± 0.15
Covertype	0.49 ± 0.08	-	0.49 ± 0.02	0.33 ± 0.00	0.61 ± 0.06	0.28 ± 0.00	0.47 ± 0.05
Insects	0.62 ± 0.06	-	0.68 ± 0.11	-	0.51 ± 0.08	0.62 ± 0.08	0.52 ± 0.08
Mean Measure	0.58 ± 0.06	0.55 ± 0.03	0.56 ± 0.09	0.44 ± 0.03	0.56 ± 0.06	0.44 ± 0.05	0.42 ± 0.05
Mean Ranking	2.9	3.1	2.6	4.1	3.2	4.4	4.8

Table 3: *Global Concept Drift Detection – Part 2.* Below we depict the drift detection delay (no. of observations; lower is better). As in the previous table, missing values correspond to test runs in which a method did not issue a single alert.

	CDLEEDS	ERICS	ADWIN	DDM	ECDD	MDDM-A	RDDM
TüEyeQ	940	39	357	1,049	282	973	819
Bank-Mark.	238	7	4,586	-	181	4,582	4721
Electricity	5,859	98	4,656	-	9,097	13,643	13,643
Adult	204	2	1,851	14,701	60	14,701	14,702
Airlines	819	20	144	81,262	4,482	54,043	54,035
SEA	2,517	75,032	675	27,959	76	25,897	1,410
Agrawal-Grad.	13,251	30	5,987	100,415	328	12,869	101,306
Agrawal-Mix.	10,163	27	1,315	36,515	173	2,417	33,691
KDD Cup	5,412	-	20,177	98,829	550	-	25,312
Covertype	11,316	-	27,817	116,227	366	58,408	14,878
Insects	152	-	4,160	-	6,091	2,050	17,552
Mean Ranking	3.3	1.8	3.1	6.1	2.3	4.6	5.0

6 CONCLUSION

To the best of our knowledge, this is the first work to formally investigate the mechanisms underlying changes in local feature attributions in data streams. It turns out that for sensible attribution methods that respect the local accuracy criterion, attribution changes are a direct consequence of incremental model updates and concept drift. CDLEEDS, the framework proposed in this work, can reliably detect such changes both locally and globally, thereby enabling more efficient and reliable use of attribution methods in online machine learning. Indeed, the proposed framework can compete with state-of-the-art concept drift detection methods, which we have demonstrated in extensive experiments on publicly available data sets. Accordingly, CDLEEDS is a flexible tool that enables more efficient, robust, and meaningful explainability in data stream applications.

REFERENCES

[1] Kjersti Aas, Martin Jullum, and Anders Løland. 2021. Explaining individual predictions when features are dependent: More accurate approximations to Shapley values. *Artificial Intelligence* 298 (2021), 103502. https://doi.org/10.1016/j.artint.2021.103502

[2] Amina Adadi and Mohammed Berrada. 2018. Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI). *IEEE Access* 6 (2018), 52138–52160.

[3] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. 2018. Towards Better Understanding of Gradient-Based Attribution Methods for Deep Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Sy21R9JAW

[4] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion* 58 (2020), 82–115.

[5] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and R Morales-Bueno. 2006. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, Vol. 6. 77–86.

[6] Roberto SM Barros, Danilo RL Cabral, Paulo M Gonçalves Jr, and Silas GTC Santos. 2017. RDDM: Reactive drift detection method. *Expert Systems with Applications* 90 (2017), 344–355.

[7] Albert Bifet and Ricard Gavalda. 2007. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 443–448.

[8] Albert Bifet and Ricard Gavalda. 2009. Adaptive learning from evolving data streams. In *International Symposium on Intelligent Data Analysis*. Springer, 249–260.

[9] Jock A. Blackard and Denis J. Dean. 1999. Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Types from Cartographic Variables. *Computers and Electronics in Agriculture* 24, 3 (dec 1999), 131–151. https://doi.org/10.1016/s0168-1699(99)00046-0

[10] Zoran Bosnić, Jaka Demšar, Grega Kešpret, Pedro Pereira Rodrigues, Joao Gama, and Igor Kononenko. 2014. Enhancing data stream predictions with reliability estimators and explanation. *Engineering Applications of Artificial Intelligence* 34 (2014), 178–192.

[11] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. 2006. Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM international conference on data mining*. SIAM, 328–339.

[12] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. 2019. Machine learning interpretability: A survey on methods and metrics. *Electronics* 8, 8 (2019), 832.

[13] Hugh Chen, Joseph D Janizek, Scott Lundberg, and Su-In Lee. 2020. True to the Model or True to the Data? *arXiv preprint arXiv:2006.16234* (2020).

[14] Jaka Demšar and Zoran Bosnić. 2018. Detecting concept drift in data streams using model explanation. *Expert Systems with Applications* 92 (2018), 546–559.

[15] Pedro M Domingos and Geoff Hulten. 2001. Catching up with the Data: Research Issues in Mining Data Streams.. In *DMKD*.

[16] Ronald Aylmer Fisher. 1992. Statistical methods for research workers. In *Breakthroughs in statistics*. Springer, 66–70.

[17] Joao Gama and Gladys Castillo. 2006. Learning with local drift detection. In *International conference on advanced data mining and applications*. Springer, 42–55.

[18] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. Learning with drift detection. In *Brazilian symposium on artificial intelligence*. Springer, 286–295.

[19] João Gama and Pedro Pereira Rodrigues. 2009. An overview on mining data streams. *Foundations of Computational, IntelligenceVolume 6* (2009), 29–45.

[20] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 44.

[21] Paulo M Gonçalves Jr, Silas GT de Carvalho Santos, Roberto SM Barros, and Davi CL Vieira. 2014. A comparative study on concept drift detectors. *Expert Systems with Applications* 41, 18 (2014), 8144–8156.

[22] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)* 51, 5 (2018), 1–42.

[23] Michael Harries and New South Wales. 1999. Splice-2 comparative evaluation: Electricity pricing. (1999).

[24] Johannes Haug, Klaus Broelemann, and Gjergji Kasneci. 2022. Dynamic Model Tree for Interpretable Data Stream Learning. *IEEE 38th International Conference on Data Engineering* (2022).

[25] Johannes Haug and Gjergji Kasneci. 2021. Learning Parameter Distributions to Detect Concept Drift in Data Streams. In *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 9452–9459.

[26] Johannes Haug, Martin Pawelczyk, Klaus Broelemann, and Gjergji Kasneci. 2020. Leveraging Model Inherent Variable Importance for Stable Online Feature Selection. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1478–1502.

[27] Johannes Haug, Effi Tramountani, and Gjergji Kasneci. 2022. Standardized Evaluation of Machine Learning Methods for Evolving Data Streams. *arXiv preprint arXiv:2204.13625* (2022).

[28] Johannes Haug, Stefan Zürn, Peter El-Jiz, and Gjergji Kasneci. 2021. On Baselines for Local Feature Attributions. *AAAI-21 Explainable Agency in Artificial Intelligence Workshop* (2021).

[29] Sérgio Jesus, Catarina Belém, Vladimir Balayan, João Bento, Pedro Saleiro, Pedro Bizarro, and João Gama. 2021. How can I choose an explainer? An Application-grounded Evaluation of Post-hoc Explanations. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. 805–815.

[30] Enkelejda Kasneci, Gjergji Kasneci, Tobias Appel, Johannes Haug, Franz Wortha, Maike Tibus, Ulrich Trautwein, and Peter Gerjets. 2021. TüEyeQ, a rich IQ test performance data set with eye movement, educational and socio-demographic information. *Scientific Data* 8, 1 (2021), 1–14. https://www.nature.com/articles/s41597-021-00938-3

[31] Gjergji Kasneci and Thomas Gottron. 2016. Licon: A linear weighting scheme for the contribution ofinput variables in deep artificial neural networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 45–54.

[32] Ron Kohavi. 1996. Scaling up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (Portland, Oregon) *(KDD'96)*. AAAI Press, 202–207.

[33] Scott M Lundberg, Gabriel G Erion, and Su-In Lee. 2018. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888* (2018).

[34] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*. 4765–4774.

[35] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. 2018. Scikit-multiflow: A multi-output streaming framework. *The Journal of Machine Learning Research* 19, 1 (2018), 2915–2914.

[36] Sergio Moro, Raul Laureano, and Paulo Cortez. 2011. Using Data Mining for Bank Direct Marketing: An Application of the CRISP-DM Methodology. *Proceedings of the European Simulation and ModellingConference - ESM'2011* (2011).

[37] Gustavo Oliveira, Leandro L Minku, and Adriano LI Oliveira. 2021. Tackling Virtual and Real Concept Drifts: An Adaptive Gaussian Mixture Model Approach. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[38] Ali Pesaranghader, Herna Viktor, and Eric Paquet. 2018. Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams. *Machine Learning* 107, 11 (2018), 1711–1743.

[39] Ali Pesaranghader and Herna L Viktor. 2016. Fast hoeffding drift detection method for evolving data streams. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 96–111.

[40] Ali Pesaranghader, Herna L Viktor, and Eric Paquet. 2018. McDiarmid drift detection methods for evolving data streams. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–9.

[41] Gregory Plumb, Denali Molitor, and Ameet S Talwalkar. 2018. Model agnostic supervised local explanations. In *Advances in Neural Information Processing Systems*. 2515–2524.

[42] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[43] Gordon J Ross, Niall M Adams, Dimitris K Tasoulis, and David J Hand. 2012. Exponentially weighted moving average charts for detecting concept drift. *Pattern recognition letters* 33, 2 (2012), 191–198.

[44] Tegjyot Singh Sethi and Mehmed Kantardzic. 2017. On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications* 82 (2017), 77–99.

[45] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 3145–3153.

[46] Vinicius MA Souza, Denis M dos Reis, Andre G Maletzke, and Gustavo EAPA Batista. 2020. Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery* 34, 6 (2020), 1805–1858.

[47] Mukund Sundararajan and Amir Najmi. 2019. The many Shapley values for model explanation. *arXiv preprint arXiv:1908.08474* (2019).

[48] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*. PMLR, 3319–3328.

[49] Van-Dai Ta, Chuan-Ming Liu, and Goodwill Wandile Nkabinde. 2016. Big data stream computing in healthcare real-time analytics. In *2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE, 37–42.

[50] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant. 2018. Sketching linear classifiers over data streams. In *Proceedings of the 2018 International*

Conference on Management of Data. 757–772.

[51] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[52] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. 2016. Characterizing concept drift. *Data Mining and Knowledge Discovery* 30, 4 (2016), 964–994.

[53] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: an efficient data clustering method for very large databases. *ACM sigmod record* 25, 2 (1996), 103–114.

[54] Indre Zliobaite. 2010. Learning under Concept Drift: an Overview. *CoRR* abs/1010.4784 (01 2010).

A.6 Standardized Evaluation of Machine Learning Methods for Evolving Data Streams

Publication: Manuscript published on ArXiv (2022). At the time of publication of this thesis, the manuscript was under review at the Springer Journal of Big Data.

Contribution: At the beginning of my PhD, Gjergji Kasneci and I had the idea to develop a Python package for online feature selection. Effi Tramountani (student assistant) and I later extended this package to the float evaluation framework. I implemented large parts of the functionality myself and revised Effi Tramountani's implementations. I also contributed the documentation of float. In addition, I conducted the literature review, specified the evaluation properties, and wrote the paper. Gjergji Kasneci challenged both the Python framework and the manuscript and helped improve the final versions.

RESEARCH

Standardized Evaluation of Machine Learning Methods for Evolving Data Streams

Johannes Haug[*], Effi Tramountani and Gjergji Kasneci

[*]Correspondence: johannes-christian.haug@uni-tuebingen.de
Data Science and Analytics, University of Tuebingen, Tuebingen, Germany
Full list of author information is available at the end of the article

Abstract

Due to the unspecified and dynamic nature of data streams, online machine learning requires powerful and flexible solutions. However, evaluating online machine learning methods under realistic conditions is difficult. Existing work therefore often draws on different heuristics and simulations that do not necessarily produce meaningful and reliable results. Indeed, in the absence of common evaluation standards, it often remains unclear how online learning methods will perform in practice or in comparison to similar work. In this paper, we propose a comprehensive set of properties for high-quality machine learning in evolving data streams. In particular, we discuss sensible performance measures and evaluation strategies for online predictive modelling, online feature selection and concept drift detection. As one of the first works, we also look at the interpretability of online learning methods. The proposed evaluation standards are provided in a new Python framework called *float*. Float is completely modular and allows the simultaneous integration of common libraries, such as scikit-multiflow or river, with custom code. Float is open-sourced and can be accessed at `https://github.com/haugjo/float`. In this sense, we hope that our work will contribute to more standardized, reliable and realistic testing and comparison of online machine learning methods.

Keywords: data streams; online machine learning; evaluation framework; concept drift detection; online feature selection

1 Introduction

Data-driven or web-based applications like social media, e-commerce, and trading systems often generate and operate on large-scale evolving data streams. Unlike traditional (offline) batch machine learning, online learning methods must be able to process a potentially unlimited stream of observations and adjust to changes in the data generating process [1, 2, 3]. Therefore, it can be crucial to gain a solid understanding of a model's strengths and weaknesses before it is deployed – in particular in critical applications like online banking, autonomous driving or fraud detection. However, due to the unspecified behaviour of data streams, it is often unclear how online learning methods can be evaluated under realistic conditions.

Unlike traditional batch-trained machine learning methods, online learning models can only access a fraction of the data at every given time step. Accordingly, common evaluation strategies that require the data to be available in main memory (e.g., cross-validation) are not applicable out of the box. Instead, we usually resort to sequential and simulation-based evaluation schemes. Although some evaluation strategies have emerged in the past [10, 11, 12], we still lack a comprehensive and

Table 1: **Inconsistency of Evaluations in Online Machine Learning.** Due to non-standardized evaluation strategies, test results reported in the data stream literature can diverge by a considerable margin. For example, below *we show the accuracy for the standard Hoeffding Tree [4] on four data sets, as reported in five different papers.* In the extreme case, the reported accuracy scores differ up to 13.51 percent points. Incidentally, the Covertype and Poker data sets are strongly imbalanced. Hence, accuracy is not an ideal evaluation measure to begin with. Such inconsistencies in experiment design and reported measures can be highly misleading – especially for beginners in the field of online machine learning.

Dataset	[5]	[6]	[7]	[8]	[9]	Max. Diff.
Spambase	-	-	-	80.35	85.47	5.12
Poker	-	73.62	76.07	-	-	2.45
Electricity	-	75.35	79.20	-	77.62	3.85
Covertype	66.83	68.30	80.31	80.34	73.71	13.51

well-defined standard for the benchmarking of online learning methods [13]. Indeed, aside from online predictive modelling, there are hardly any evaluation standards for concept drift detection and online feature selection.

As a consequence, there can be considerable differences in the evaluation methods used in existing work. For example, Table 1 depicts test results from five different research papers. Specifically, we compare the accuracy of a Very Fast Decision Tree [14] for four data sets as reported in the respective papers. Due to different evaluation strategies and unspecified behaviour, the results vary considerably. Without common evaluation standards and frameworks, empirical results are often only valid in a very restricted environment and are generally not comparable.

In this work, we discuss good practices and standards for the evaluation of online learning methods. We summarize and extend popular evaluation strategies for data stream learning and introduce a comprehensive catalogue of requirements and performance measures. In particular, we propose important properties for online predictive models, online feature selection and concept drift detection. Although our focus is on online classification, most of the proposed properties also apply directly to regression or unsupervised tasks. Additionally, we briefly discuss the selection of adequate data sets, which remains an open issue.

To make the proposed evaluation standards more accessible, we introduce a new Python framework for *Frictionless Online Analysis and Testing (float)*. Float is a lightweight and high-level framework that automates and standardizes inherent tasks of online evaluations. Float's modular architecture and evaluation pipeline simplify the integration of custom code with common online learning libraries such as scikit-multiflow [15] or river [16]. Moreover, float provides a variety of useful visualizations. The proposed framework is distributed under the MIT license via Github and the Python packaging index PyPI.

In summary, we contribute a comprehensive set of properties and performance measures that allow for a more standardized and realistic evaluation of online learning methods. The new Python framework, float, provides high-level access to the proposed standards and enables a quicker, more comparable and more reliable benchmarking in research and practice.

In Section 2, we introduce basic online learning concepts. In Section 3, we briefly review previous studies on the evaluation of data stream learning. Afterwards, we discuss general evaluation strategies in Section 4 and propose a comprehensive set of properties for online predictive models, concept drift detection and online feature selection in Section 5. In this context, we also discuss the interpretability of models in the presence of incremental updates and concept drift. We provide relevant open-source resources for streaming data in Section 6. Finally, we introduce the float framework and illustrate its use in Section 7.

2 Online Learning Preliminaries

A data stream can be represented by a (potentially infinite) series of time steps. At each time step t, the data stream produces observations $x_t \in \mathbb{R}^{n_t \times m_t}$ and corresponding labels $y_t \in \mathbb{R}^{n_t}$, where n_t is the number of observations and m_t is the number of features. The joint probability distribution $P_t(X, Y)$ denotes the active data concept at time step t, where X and Y are random variables corresponding to the observations and labels.

The active concept may evolve over time. Specifically, *concept drift* describes a change in the joint probability distribution between two time steps, i.e., $P_{t_1}(X, Y) \neq P_{t_2}(X, Y)$. In general, we distinguish between real concept drift, i.e., $P_{t_1}(Y|X) \neq P_{t_2}(Y|X)$, and virtual concept drift, i.e., $P_{t_1}(X) \neq P_{t_2}(X)$. Unlike virtual concept drift, real concept drift affects the decision boundary. Note that there is a broader categorization of concept drift in the literature, e.g., based on its magnitude, length, or recurrences [17, 1].

In general, online machine learning deals with the same tasks as its offline counterpart. This includes supervised tasks like classification and regression or unsupervised tasks like clustering.

3 Related Work

Data streams are subject to external influences and temporal change. Therefore, it is often difficult to set up a testing environment that allows for meaningful evaluation of online learning methods. Various evaluation strategies and best practices have been developed in the past. While we discuss these strategies in more detail in Section 4 and 5, we provide a brief overview of existing work below.

The work of [14] was among the first to talk about important criteria for data stream mining. Much later, the authors in [10] discussed issues that arise in the evaluation of online learning methods. This work was later extended [11]. More recently, the work of [18] summarized the advantages and disadvantages of different evaluation strategies in the context of data stream classification.

Although the criteria proposed in the above works are broadly applicable, additional challenges can arise in more specific contexts. For example, the papers of [19] and [1] discussed evaluation strategies, measures and data sets that can be used to evaluate concept drift detection and adaptation techniques. Likewise, the authors in [20] proposed important criteria regarding the preprocessing of streaming data, including online feature selection. Besides, the works of [13] and [21] investigated the reliable evaluation of ensemble methods, which are a prominent group of high-performing online predictive models. Finally, the survey of [22] provided a summary of popular data streaming tools and benchmarks in the context of big data mining.

Although various evaluation standards have been proposed over the years, most of them are either outdated, have a narrow focus, or remain superficial. In this paper, we provide a concise summary of the most popular evaluation practices in online machine learning. As one of the first works, we discuss and propose evaluation properties and measures for predictive modelling, feature selection and concept drift detection in non-stationary data streams. This work may thus guide beginners and experts alike in the conception of more standardized benchmarks and experiments.

4 Evaluation Strategies

To train and evaluate a machine learning model, we require data. Traditionally, we have access to a training data set during development. This data set is split in a meaningful way (e.g., once in a holdout validation or k times in a k-fold cross-validation). We then train the model on the training set and evaluate it on the test set. In data streams, however, we do not have access to a complete data set at any time. In addition, the data generating distribution can change. As a result, online learning models need to be continuously updated over time. Therefore, it would not be sensible to evaluate an online learning model at a specific point in time, i.e., at a single training stage using a static train/test split. Selecting an adequate time step would also be difficult in practice, since a data stream is potentially infinite. Rather, we need to evaluate online learning models periodically. On this basis, several evaluation strategies have been proposed, which we illustrate in Figure 1 and discuss below.

4.1 Periodical Holdout Evaluation

In a periodical holdout evaluation, we test the online predictive model at frequent intervals. The holdout set comprises test observations that represent the active concept. Since the active concept evolves, the holdout set should be updated over time. For example, we may replace old instances in the holdout set as we progress. Between the periodic evaluations, we continue updating the model, but do not test its performance. The test frequency is usually controlled via a hyperparameter. Depending on the test frequency, a holdout evaluation might miss particularly short-term data concepts. Moreover, holdout evaluations yield only snapshots of the system at particular time steps. That is, there is no guarantee that the holdout evaluation at a time step t is also representative for $t-1$ and $t+1$. Therefore, we believe that a holdout evaluation should generally only be used when both the data distribution and the predictive model are known to be stable.

4.2 Prequential Evaluation

Unlike a periodical holdout evaluation, a prequential strategy (also known as test-then-train) evaluates the predictive model at each time step. Specifically, new observations are first used to test and then update the model. This is the most common evaluation strategy in practice. A prequential evaluation gives a more pessimistic performance estimate than the holdout evaluation [10]. In particular, the prequential evaluation also takes into account all early performance measurements, which are often poor due to the initial training of the model.

To mitigate this effect, one may introduce a forgetting mechanism via sliding windows or fading factors [11]. In the sliding window approach, we aggregate the

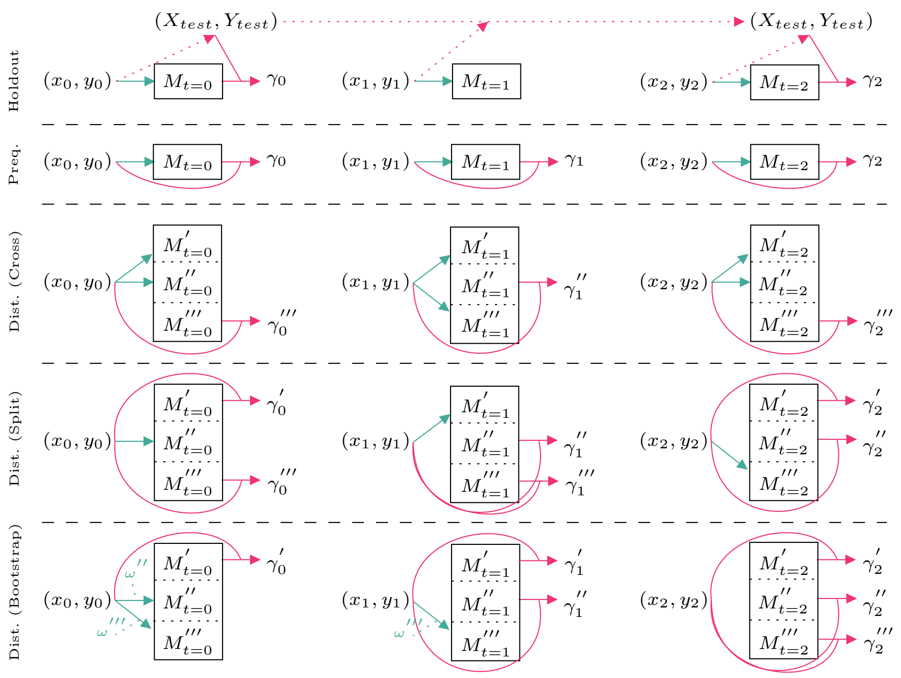


Figure 1: **Evaluation Strategies for Online Machine Learning.** Since online predictive models are trained incrementally, traditional (e.g., sampling-based) evaluation strategies are not applicable. Instead, we can use periodic holdout, prequential, or distributed k-fold evaluation, which we describe in Section 4. For each of the strategies, at each time step we first test the model (pink), which yields a performance measure $\gamma \in \mathbb{R}$, and then update the model (green). In a periodic holdout evaluation, we maintain a test set (X_{test}, Y_{test}) that may be updated over time (indicated by dashed arrows). In a distributed k-fold evaluation, we train k instances of the model in parallel (indicated by quotes). The instances to be trained and tested are randomly selected according to one of three schemes (cross-/split-/bootstrap-validation). In the bootstrap validation scheme, ω specifies a training weight drawn from the Poisson distribution.

measurements obtained in a time window. Any measurement that falls outside the specified window no longer contributes to the performance estimate. In a fading factor approach (i.e., exponentially weighted averaging), the influence of old measurements decreases over time according to the specified fading (or decay) factor. Unlike a sliding window, a fading factor approach is memory-less, meaning that old measurements do not need to be saved.

4.3 Distributed k-Fold Evaluation

Cross-validation is one of the most popular evaluation techniques in offline learning environments because it provides more robust results than a traditional holdout evaluation. However, cross-validation and other sampling techniques are generally not applicable in a data stream, since we cannot access the entire data set and the data distribution is subject to change [10]. Therefore, the periodical holdout and

Table 2: **Evaluation Properties for Data Stream Methods.** Below we summarize all properties and corresponding performance measures introduced in Section 5 for the evaluation of machine learning methods in evolving data streams. In general, we recommend to use a prequential evaluation strategy or, if statistical significance is required, a more costly k-fold distributed cross validation [18] (Section 4). With the novel Python framework *float* (Section 7), we can run standardized experiments and evaluate these properties in a few lines of code.

Property	Meaningful Evaluation Measures
Predictive Performance	Generalization error; F1 measure or κ-statistic [18] (for classification)
Computational Efficiency	Computation time (for training and testing); RAM-Hours [6]
Algorithmic Stability	Noise Variability (Eq. (1))
Concept Drift Adaptability	Drift Performance Deterioration (Eq. (2)); Drift Restoration Time (Eq. (3))
Interpretability	Complexity over time, e.g. no. of parameters/splits [23]
Concept Drift Detection	
Detection Truthfulness	Detected Change Rate (Eq. (4)); False Discovery Rate (Eq. (5)); Mean Time Between False Alarms [12]
Detection Timeliness	Delay (no. of time steps); Mean Time Ratio (Eq. (6)) [12]
Online Feature Selection	
Feature Set Stability	Adjusted Nogueira stability (Eq. (7)) [24, 25]
Feature Selectivity	Reduction rate (in % of the original feature dimensionality)

prequential strategies described above are commonly used even though they do not provide statistical significance. Indeed, hypothesis testing is rather uncommon in the data stream literature.

As an alternative, the authors in [18] proposed a k-fold distributed evaluation strategy that allows us to perform hypothesis testing for online learning models. To this end, we need to run k instances of the same predictive model in parallel. Each streaming observation is distributed to one or multiple model instances according to one of three validation schemes. In the *cross-validation* scheme, at every time step, we randomly pick one classifier instance for testing and use the remaining ones for training. Conversely, the *split-validation* scheme trains one randomly selected model instance per time step and tests all others. Finally, in the *bootstrap-validation* scheme, we sample weights from a Poisson distribution for each classifier instance. The weights indicate whether a model instance is used for testing (zero weight) or training (weight greater than zero). Note that the Poisson weight also controls the influence of an observation during training. For example, with a sample weight of 3, we would use an observation three times to update the corresponding model instance. In this way, the bootstrap validation scheme simulates random sampling with replacement [18].

The experiments of [18] showed that the k-fold distributed cross-validation strategy gives reliable results for different hypothesis tests. However, the distributed k-fold evaluation is usually much more costly than a periodical holdout or prequential evaluation.

5 Properties for Online Machine Learning Methods

As described above, evolving data streams bring several challenges. Next, we translate these challenges into a set of fundamental properties and requirements for online machine learning. In addition, we discuss meaningful performance measures to evaluate each property. We begin with general properties and then address specific aspects of concept drift detection and online feature selection. Unless stated otherwise, we do not impose any restrictions on the functional form of the online predictive model. That is, we assume that the predictive model can be queried but otherwise remains a black-box. Consequently, the proposed properties are directly applicable to any online learning framework. In combination, these properties enable online learning methods that can be effectively used under real-world streaming conditions.

5.1 General Properties

In the following, we specify general properties for high-quality online learning methods. The importance of each property may depend on the application at hand. We list all properties along with recommended performance measures in Table 2.

5.1.1 Predictive Performance and Generalization

High predictive performance, i.e., low generalization error, is a central goal of any predictive machine learning method. In general, this property describes the ability of a model to correctly predict the target of previously unobserved test data. As discussed in Section 4, we need to choose between different strategies to periodically evaluate an online predictive model. In this context, we can apply the same or slightly adjusted performance measures as for offline learning. For example, commonly used measures for online classification are the 0-1 loss and the accuracy (which only gives valid results for balanced target distributions). For more robust results on imbalanced data, one may use a combination of precision and recall (e.g., F1) [1], variations of the κ-statistic [18] or an adaptation of the Area Under the Curve of the Receiver Operating characteristic (AUC) [26].

5.1.2 Computational Efficiency

To model data streams in theory, we usually apply a logical notion of time, i.e., we treat time as a sequence of discrete steps (see Section 2). However, in practice, streaming observations can arrive in very quick succession. For example, large scale web-based applications such as credit card transactions may produce many new observations per second. In order to avoid a backlog and update the model in real-time, incoming observations should be processed at the rate they arrive [10]. Likewise, online applications often have limited access to hardware capacities, e.g., in a distributed or embedded system. Therefore, it can be crucial that online learning models are efficient and use few resources.

Measuring the computation time is relatively simple with standard packages (e.g., the *time*-package in Python). The computation times for model training and testing should be monitored independently [20]. To quantify the memory usage, the authors in [6] proposed to use RAM-hours. One RAM-hour corresponds to one Gigabyte of RAM used for one hour. However, during development, it can be difficult to accurately monitor the RAM usage of a particular model or process. Moreover, existing software packages for monitoring memory usage are often inaccurate or extremely slow. Indeed, both the estimated computation time and memory usage depend heavily on the implementation and the given hardware specifications. Hence, efficiency estimates are generally only comparable if they were obtained under the same conditions. A theoretical analysis of model complexity often provides a more reliable indication of computational efficiency.

5.1.3 Algorithmic Stability

Real-world data is subject to noise, e.g., due to faulty network connections or (human) errors in data collection. If a model is susceptible to noise, its performance usually suffers. Unlike batch learning methods, online learning models must be able to distinguish noise and outliers from concept drift. This can be difficult because both noise and concept drift manifest as previously unobserved behaviour. Surprisingly, the stability of online learning methods has received little attention in the past.

Stability is often estimated by calculating the variability of the model output for perturbed inputs. That is, we can manipulate an input observation with artificial noise (e.g., sampled from some probability distribution) and monitor the change in a performance measure. To the best of our knowledge, there is no common estimate of stability for data stream methods. Therefore, we define the *noise variability* at time step t:

$$NV_t = \frac{1}{N}\sum_{n=1}^{N} L\big(y_t, f_t(x_t + z_n)\big) - L\big(y_t, f_t(x_t)\big), \tag{1}$$

where N is the number of times we sample noise $z_n \sim \mathcal{Z}$ (e.g., $\mathcal{Z} = \mathcal{N}(0,1)$), L is a performance measure function and f_t is the model at time step t. If the noise variability is small for most time steps, i.e., a large number of perturbed observations, we can assume that the online learning model is stable. Nevertheless, it can be difficult to find a meaningful noise distribution $\mathcal{Z}$, so this measure should be used with caution. In general, more attention should be paid to evaluating the stability of machine learning models in data streams.

5.1.4 Concept Drift Adaptability

In general, we can deal with concept drift either by passive model adaptation or by active drift detection. In the former approach, the model is adjusted over time, e.g., through continuous updates or the use of sliding windows. Conversely, in the active approach, we aim to identify the exact time of concept drift using a dedicated drift detection method [27]. Each time we actively detect a concept drift, we re-train the model (or parts of it). Accordingly, active drift detection can, but need not, be part of the drift adaptation process of the online learning model. Next, we discuss the general evaluation of model performance in the presence of concept drift. Note that active drift detection methods should be evaluated in terms of additional properties, which we present in Section 5.2.

Online learning models should be robust enough to suffer only minor performance deterioration after concept drift, and flexible enough to quickly recover previous performance. To the best of our knowledge, we currently lack sensible definitions of corresponding measures. Accordingly, given a concept drift at time step t_d, we define *drift performance deterioration* as:

$$DPD_{t_d} = \frac{1}{W}\left(\sum_{w=0}^{W-1} L\big(y_{t_d+w}, \hat{y}_{t_d+w}\big) - \sum_{w=1}^{W} L\big(y_{t_d-w}, \hat{y}_{t_d-w}\big)\right), \tag{2}$$

where W is the size of a time window, L is a performance measure and $\hat{y}_t = f_t(x_t)$ is the prediction of the model f_t for the observation x_t. The DTD_{t_d} corresponds to the difference between the mean performance in a window of size W before and after a known concept drift at t_d. Similarly, we define the *drift restoration time*:

$$\begin{aligned} &DRT_{t_d} = t_{res} - t_d, \text{with} \\ &\quad t_{res} = \min\left\{t \mid t \geq t_d \text{ and } L\big(y_t, \hat{y}_t\big) \leq \frac{1}{W}\sum_{w=1}^{W} L\big(y_{t_d-w}, \hat{y}_{t_d-w}\big)\right\} \end{aligned} \tag{3}$$

The DRT_{t_d} corresponds to the first time step after a known drift t_d, at which the average predictive performance before the drift has been restored. In the definition of t_{res} in Eq. (3), we have assumed that the measure L is decremental, i.e., small values returned by L correspond to a good performance. If L were instead incremental (i.e., a higher L is better), we would need to replace the "$\leq$" condition with "$\geq$".

To identify and evaluate sensible model adaptations over time, and to compute the above measures, we need ground truth information on known concept drifts. Indeed, without ground truth, it would remain unclear whether model adaptations and performance variations were caused by concept drift, noisy data, unstable model behaviour or random effects. Unfortunately, there are few available real-world data sets with known concept drift. We discuss the selection of adequate data sets in Section 6.

5.1.5 Interpretability

Machine learning models are increasingly used in highly sensitive applications like online banking, medical diagnoses or job application systems. With new data protection regulations (e.g., the General Data Protection Regulation of the European Union), the transparency and interpretability of these models has gained considerable attention [28, 29, 30, 31, 32]. Surprisingly, however, the interpretability of online learning methods is still relatively unexplored.

In general, we say that a model is interpretable, if its internal mechanics can be understood by a human. Unfortunately, there is no objective measure of interpretability. Intrinsic interpretability is thus often linked to the complexity of a model [28, 33], i.e., the lower the complexity, the higher the interpretability. For example, linear models and shallow decision trees are widely considered to be inherently interpretable. If we consider model complexity as an indicator of interpretability, we can use different evaluation measures such as the depth and the number of splits in a decision tree [34, 23], or the number of non-zero parameters in a linear model. Still, such heuristics should be treated with caution, as they usually do not allow for a comparison of different model families.

Moreover, the complexity of an online learning model is generally subject to change over time. Therefore, to achieve good interpretability in the above sense, the changes and updates of model complexity should also be interpretable. For example, the split and prune decisions in an incremental decision tree can be made more interpretable by tying them to shifts in the approximate data concept via meaningful properties [23]. Likewise, reliable concept drift detection methods could improve the interpretability of online learning [27]. However, these temporal dynamics make quantifying interpretability even more difficult. In general, the interpretability of online learning models remains an open issue.

Digression: Post-Hoc Explanations in Data Streams Post-hoc explanation methods allow us to explain the predictions of complex and black-box models [35, 29]. Explanation methods like local feature attributions [36, 37, 38] can generally also be used to explain (non-interpretable) online learning models. Moreover, these explanations may be evaluated with common techniques, e.g., based on feature ablation tests [39, 40]. However, most post-hoc explanation methods, as well as the corresponding evaluation techniques, assume that the complex model has been trained and is therefore stationary. Accordingly, for online learning models, these methods can usually only provide a snapshot for a particular training phase. That is, we can use them to explain the online learning model at a particular time step t. However, there is generally no guarantee that the explanation at a given time step will be valid in the future. Indeed, one should be aware of changes in post-hoc explanations caused by incremental model updates and concept drift. Therefore, instead of trying to explain black-box predictions at individual time steps, we should aim for online learning models that are inherently interpretable.

5.2 Properties For Active Concept Drift Detection

In addition to predictive modelling, machine learning in data streams includes specific tasks required to preprocess streaming observations or deal with temporal change. While the properties presented above generally also apply to these special tasks, additional challenges and properties arise.

One such special task is active concept drift detection. Concept drift detection methods are used to identify changes in the data generating distribution over time. As mentioned earlier, concept drift detection methods often allow for more effective retraining of obsolete parts of a model. This makes them a powerful tool for avoiding performance degradation in the presence of concept drift. Concept drift detection can also improve overall interpretability by revealing hidden dynamics in the data stream. In general, a concept drift detection model should be able to detect concept drift in time and with few false alarms [11, 19, 1]. As mentioned in Section 5.1.4, we require ground truth information about known drifts to evaluate these properties.

5.2.1 Detection Truthfulness

Our goal is to detect every concept drift while minimizing the number of false alarms. To evaluate this property, previous works adopt similar performance measures, albeit using different terminology [19, 12, 13]. For clarification, we define the most important measures below.

Let T_d be the set of time steps corresponding to known concept drifts and let $\hat{T}_d$ be the set of detected drifts of a concept drift detection method. We begin by defining the *detected change rate*:

$$DCR = \frac{1}{|T_d|} \sum_{t_d \in T_d} \mathbb{1} \left(\left\{ t \in \hat{T}_d \mid t_d \leq t \leq t_d + W \right\} \neq \emptyset \right), \tag{4}$$

where $|\cdot|$ is the cardinality of a set and $\mathbb{1}$ is an indicator function that returns 1 if the condition in parentheses is met and 0 otherwise. The window size W can be used to define an interval in which a drift detection is counted as a true positive. This can be useful, as drift detection methods are usually not able to detect a concept drift immediately – in particular, if the change has small magnitude. The *missed detection rate* introduced in [12] is equal to $1 - DCR$.

The *false discovery rate* (also known as false positive rate or false alarm rate) is another popular measure:

$$FDR = 1 - \frac{1}{|\hat{T}_d|} \sum_{t_d \in T_d} \left| \left\{ t \in \hat{T}_d \mid t_d \leq t \leq t_d + W \right\} \right| \tag{5}$$

Additionally, we may compute the *time between false alarms* [12], which is quantified by the number of time steps (or the number of streaming observations) between false drift detections. Good concept drift detection corresponds to a high time between false alarms.

Each above measure can easily be optimised for itself. Specifically, if a method detects concept drift at every time step, it would maximise the detected change rate. Similarly, if a method does not detect a single concept drift, it would minimize the false discovery rate and maximize the time between false alarms. Hence, for a reliable evaluation of active concept drift detection, the measures must be combined.

5.2.2 Detection Timeliness

Aside from accurately detecting concept drifts, we also want to reduce the *delay* between known drifts and corresponding detections. A short delay can be crucial in practice to avoid long-term deterioration of predictive performance. The detection delay measures the number of time steps (or alternatively the number of observations) between a known drift and the first corresponding detection [13].

The mean delay (MD), the mean time between false alarms (MTFA) and the detected change rate (DCR, Eq. (4)) can also be aggregated in a *mean time ratio* measure (MTR) [12]:

$$MTR = \frac{MTFA}{MD} \times DCR \tag{6}$$

The mean time ratio is a simple approach to combine the two fundamental properties of active concept drift detection. However, this measure should be used with caution, as the time between false alarms and the delay are not normalized, which can lead to very different mean time ratios depending on the data set at hand.

Unlike previous measures, the performance measures for active concept drift detection only need to be calculated once at the end of the evaluation. Hence, these results are not influenced by the evaluation strategy that we apply (see Section 4).

5.3 Properties For Online Feature Selection

By reducing the input dimensionality, feature selection methods often allow for more efficient and discriminative online learning. Similar to concept drift detection, online feature selection poses additional requirements, which we discuss in the following.

5.3.1 Feature Set Stability

In offline learning scenarios, feature selection is usually performed once before model training. Conversely, online feature selection models need to be updated over time, since the importance of features may shift with concept drift. Yet, large variations of the selected features between time steps can be perceived as unintuitive or non-robust [25]. Besides, frequent changes to the selected features may entail excessive and costly updates of the predictive model. Consequently, we generally aim for stable feature sets over time – in particular, if the data concept is known to be stable. In this sense, feature set stability is related to the general *robustness to noise* property introduced above.

The stability of feature sets in offline learning scenarios has attracted attention in the past [41, 42]. In [25], the authors proposed an adaptation of a popular offline stability measure due to [24] for the evaluation of online feature selection methods. Accordingly, let $a_t \in \{0,1\}^m$ be the active feature vector at time step t, where m is the total number of features. In the vector a_t, selected features are represented by ones and unselected features are represented by zeros. The *feature selection stability* at time step t for a sliding window of size w is then defined as:

$$FSS_{t,w} = 1 - \frac{\frac{1}{m}\sum_{j=1}^{m} s_j^2}{\frac{k}{m}\left(1 - \frac{k}{m}\right)}, \tag{7}$$

where k is the number of selected features and $s_j^2 = \frac{w}{w-1}\hat{p}_j(1-\hat{p}_j)$ is the unbiased sample variance of the selection of feature j, with $\hat{p}_j = \frac{1}{w}\sum_{i=0}^{w-1} a_{t-i,j}$. To compute the stability between the consecutive feature sets at $t-1$ and t, we can set the window size to $w = 2$. The feature set stability due to (7) decreases, if the total variability of the selected features $\sum_{j=1}^{m} s_j^2$ increases. Conversely, the stability is maximized if $s_j^2 = 0$ for all features j, i.e., if the selected feature set remains stationary over the full length of the sliding window.

Offline feature set stability measures do not take into account concept drift, where we would normally tolerate some degree of variability. Hence, stability measures adopted from the offline literature, as proposed above, should be considered with care. In general, measuring the stability of feature sets in the presence of concept drift is an open problem.

5.3.2 Feature Selectivity

With most online feature selection methods, the size of the returned feature set must be specified in advance. However, if a feature selection method is able to automatically determine the ideal feature set size, the reduction rate could be another useful evaluation measure [20]. Specifically, the reduction rate indicates the percentage of original features that has not been selected. Given similar predictive performance, the smaller feature set, i.e., the larger reduction rate, is generally preferable.

5.4 Additional Considerations

In the following, we briefly list additional properties and considerations that should be taken into account when developing and evaluating online machine learning methods. Although each individual point would deserve its own section, this is beyond the scope of this paper.

- **Label Delay**: In practice, labelling information is often costly and only becomes available some time after the corresponding observation. In fraud prevention scenarios, for example, a fraudulent transaction may not be recognized for several days or weeks. For the sake of simplicity, we often assume that the labels for all observations are available immediately. However, powerful online learning methods should be able to handle delayed label information.
- **Normalization**: Normalization can dramatically improve the performance of a predictive model. Unfortunately, it is not possible to normalize streaming data in one go. In fact, feature scales might shift over time due to concept drift. Accordingly, if the feature scales are not known in advance, the information used to normalize incoming observations should be iteratively updated. In this context, one should also take care of outliers, which can considerably impact the normalization (e.g., in the common min-max scaling). In general, the normalization of streaming observations should receive more attention.
- **Imbalanced Targets**: The target class of streaming data might be heavily imbalanced (e.g., in the context of detecting credit card fraud). Indeed, imbalances might be subject to temporal change. For example, credit card transactions have a higher frequency in the evenings and during weekends. Therefore, we require online learning methods to robustly handle imbalanced target distributions. Likewise, we need to select performance measures that are meaningful in the presence of imbalances, e.g. the F1 measure for classification.
- **Feature/Concept Evolution**: Concept drift may alter the set of features and classes that we observe over time. We call these phenomena *feature evolution* and *concept evolution*, respectively [43]. For example, new trending topics (i.e., classes) on social media might temporarily produce new hashtags (i.e., features). The occurrence of feature evolution and concept evolution poses additional difficulties for online machine learning methods.

6 Finding Real-World Benchmark Data Sets

Many evaluation measures presented above require ground truth information about known concept drift in order to be calculated. However, it is often impossible to obtain ground truth from real-world processes. Although there are approaches to induce artificial concept drift to real-world data sets [44, 27], we argue that there is a fundamental shortage of adequate benchmark data sets for the evaluation of online learning methods.

Indeed, synthetically generated data streams still dominate the literature. Generating synthetic data with various kinds of concept drift is straight-forward through packages like scikit-multiflow [15] or river [16]. A comprehensive collection of synthetic data streams has been proposed by the authors in [45].

Yet, there is as series of real-world data sets that are frequently used for the evaluation of online learning methods. The authors in [46] provide an extensive summary and discussion of some of the most popular real-world data sets for online learning. In addition, they propose a collection of insect classification data sets, which comprise natural concept drift. The *Insects* data sets comprise sensor information from flying insect species, obtained in a controlled environment. By adjusting the temperature and humidity, the authors obtained different types of natural concept drift. Another recently published data set with known concept drift is *TüEyeQ* [47, 48]. TüEyeQ comprises sociodemographic information about participants in an IQ test. The data contains natural concept drifts by switching between different task blocks and increasing difficulty within each block.

Finally, there are various public sources from which streaming data sets can be obtained:

- `https://www.openml.org/search?type=data` (search for "data stream" or "concept drift")
- `https://sites.google.com/view/uspdsrepository` [46]
- `https://github.com/ogozuacik/concept-drift-datasets-scikit-multiflow`
- `https://github.com/vlosing/driftDatasets`

7 The "float" Evaluation Package

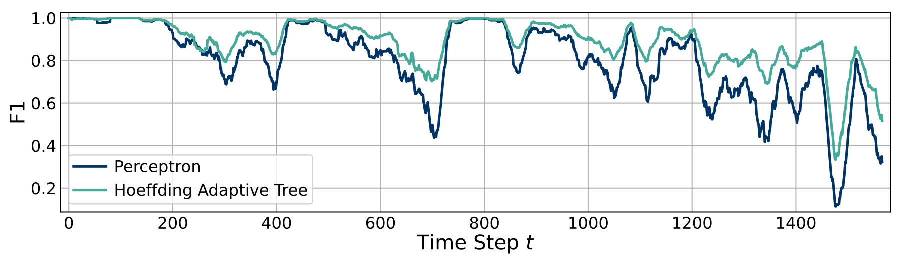
Along with this paper, we introduce *float*. Float is a modular Python framework for simple and more standardized evaluations of online learning methods. Our framework provides easy and high-level access to popular evaluation strategies and measures, as described above. In this way, float handles large parts of the evaluation and reduces the possibility of human error. Float enables joint integration of popular Python libraries and custom functionality. Accordingly, float is a meaningful extension of comprehensive libraries like scikit-multiflow [15] or river [16]. In this sense, *float is not intended to be another library of state-of-the-art models.* Rather, our goal is to provide tools for creating high-quality experiments and visualisations.

7.1 Access and Code Quality

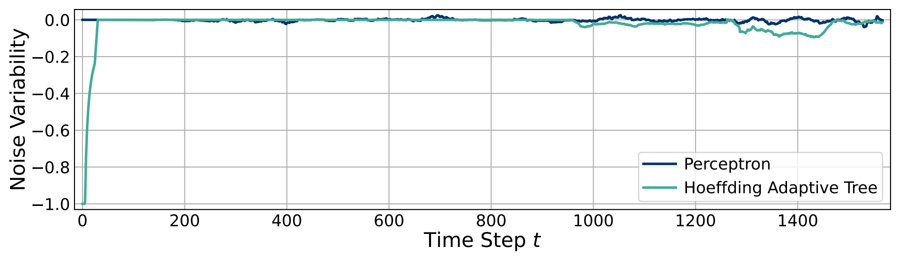
Float is distributed under the MIT license. The framework can currently be accessed via Github (`https://github.com/haugjo/float`) or the Python packaging index Pypi (`https://pypi.org/project/float-evaluation/`). The source code of float is fully documented according to the Google docstring standard. The documentation can also be accessed at `https://haugjo.github.io/float/`. To ensure the quality and readability of our source code, we applied the PEP8 formatting standard. Moreover, we created an extensive set of unit tests to validate all core functionality. The test suite is available on Github.

7.2 Modularity

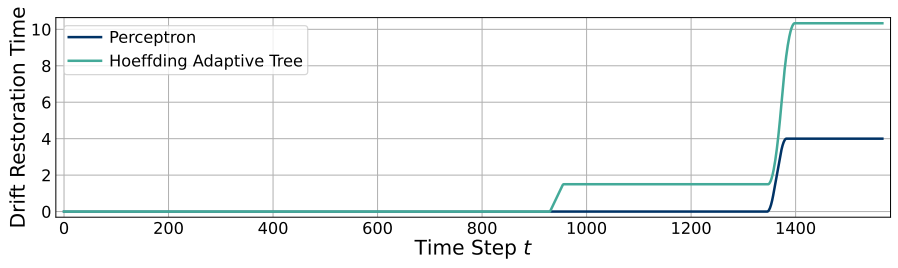
The source code of float is completely modular. We encapsulate related functionality in Python classes. Specifically, there are classes for online prediction, concept drift detection, and online feature selection, as well as corresponding classes for their evaluation. Users can integrate their own models by inheriting from abstract base classes. The evaluation strategies discussed in Section 4 are implemented as pipelines (which are also Python classes). The pipelines allow users to specify custom experiments and run any combination of concept drift detection, online feature selection, and predictive models. Indeed, with float it is possible to configure a pipeline that combines custom models and common Python packages. For example, within the same pipeline, we may load a data set via the scikit-multiflow *FileStream* [15], implement a custom online classifier, and use scikit-learn metrics for the evaluation. Besides, float provides a number of visualisations that can be used to illustrate the results of the pipeline run. Float also includes various recent and state-of-the-art online learning methods that are not part in any of the major libraries yet. We plan to extend the set of available performance measures, preprocessing techniques, and evaluation strategies in the future and welcome contributions by the community.



(a) F1 Performance Measure



(b) Noise Variability (with respect to F1), see Eq. (1).



(c) Drift Restoration Time (in mean no. of time steps), see Eq. (3).

Figure 2: **Illustrating Results with Float.** The float framework contains a visualisation module that provides adaptations of common plot types such as line, scatter or bar plots. In this way, float enables a quick and intuitive visualisation of the results stored in an evaluator object. Above, we see the results of the example described in Section 7.3.1.

7.3 Usage

It is neither meaningful nor feasible to describe all modules and configurations of float in this paper. However, on Github we provide detailed documentation and multiple exemplary experiments in the form of Jupyter notebooks that can help users to familiarize themselves with the float framework and its modules. For illustration, we describe a simple experiment with float below.

7.3.1 Exemplary Experiment

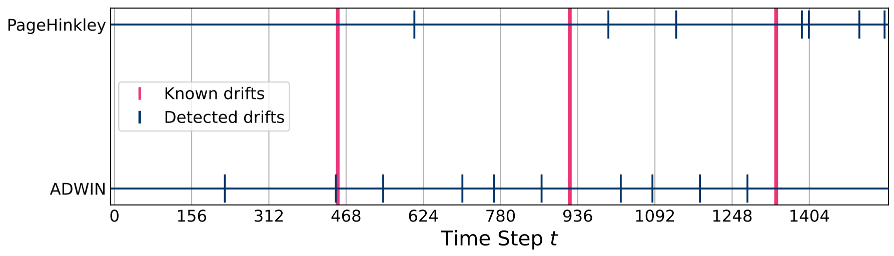
In this example, we want to train an online predictive model on the TüEyeQ data set [47]. The classification task is to decide whether a task was passed or failed given a vector of task-specific features and socio-demographic information about the corresponding subject (77 features in total). Note that we will not optimize any hyperparameters of the models involved, as this is only an illustrative experiment. However, for practical applications, float allows us to run multiple configurations of a predictive model in parallel for effective hyperparameter optimization.

We start by comparing a Perceptron model and a Hoeffding Adaptive Tree [49]. The source code for this first experiment is provided in Figure 3. We load the data set file with the *DataLoader* module of float. Then we set up the two predictive models. In order to use implementations of scikit-multiflow [15], we need to wrap the corresponding objects within a *SkmultiflowClassifier* object. Next, we specify the *PredictionEvaluator* object, which calculates and stores the performance measures of both models. In particular, we instruct the evaluator to compute the F1 measure, the noise variability (Eq. (1)), and the drift restoration time (Eq. (3)). For the latter measure, we need information about known concept drifts, which we provide as a hyperparameter. We also specify the batch size that we will use in the prequential evaluation. Note that we can specify any hyperparameter of a measure function directly in the constructor of the float evaluator object. For example, we may set the *zero_division* parameter of the scikit-learn *f1_score* function, as well as the *reference_measure* and *n_samples* parameters of the noise variability measure. In addition to the raw performance measurements, we also want to obtain the performance aggregated in a sliding window. To this end, we specify a sliding window size of 25. Finally, we set up a prequential pipeline and provide all previously initialised objects. We use 100 observations to pre-train the online learning models before starting the prequential evaluation with a batch size of 10.

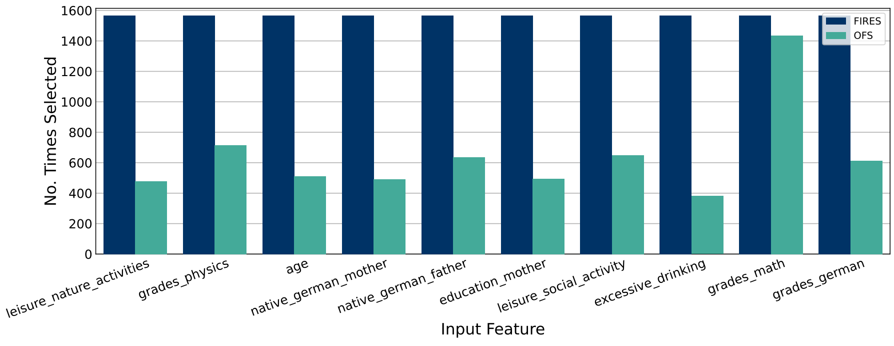
To compare the performance of the two classifiers, we show the aggregated F1 score, the noise variability and the drift restoration time in Figure 2 (using the line plot type of float). All displayed measures are stored in the evaluator object. Based on the predictive performance of the classifiers, we can clearly see the concept drift in TüEyeQ. In particular, we observe that the predictive performance of the classifiers within each task block starts to suffer when the IQ-related tasks become more difficult to solve. However, the drift restoration time only increase for the last two concept drifts. In general, the Perceptron algorithm performs worse than the Hoeffding Adaptive Tree in terms of F1, but slightly better in terms of noise variability and drift restoration time. Since the Perceptron model does not actively adapt to concept drift, we might further improve performance by using a drift detection method to trigger active retraining.

```
from skmultiflow.trees import HoeffdingAdaptiveTreeClassifier
from skmultiflow.neural_networks import PerceptronMask
from sklearn.metrics import f1_score
from float.data import DataLoader
from float.prediction.evaluation import PredictionEvaluator
from float.prediction.evaluation.measures import noise_variability,
  mean_drift_restoration_time
from float.pipeline import PrequentialPipeline
from float.prediction.skmultiflow import SkmultiflowClassifier

# Load a data set from main memory with the DataLoader module.
# Alternatively, we can provide a scikit-multiflow FileStream ...
# ... object via the 'stream' attribute.
data_loader = DataLoader(path='./datasets/iq.csv',
                         target_col=-1)
known_drifts = [4707, 9396, 13570]  # Known drift positions

# Set up online classifiers.
# Note that we need a wrapper to use scikit-multiflow functionality.
models = [SkmultiflowClassifier(model=PerceptronMask(),
                                classes=
                                 data_loader.stream.target_values),
          SkmultiflowClassifier(model=
                                 HoeffdingAdaptiveTreeClassifier(),
                                classes=
                                 data_loader.stream.target_values)]

# Set up an evaluator object for the classifiers:
# Specifically, we want to measure the f1_score, ...
# ... the noise_variability and the drift_restoration_time.
# The arguments of the measure functions can be directly added to ...
# ... the evaluator object constructor, e.g. we may specify ...
# ... the number of samples (n_samples) and the reference_measure ...
# ... used to compute the noise_variability.
evaluator = PredictionEvaluator(measure_funcs=[f1_score,
                                 noise_variability,
                                 mean_drift_restoration_time],
                                window_size=25,
                                zero_division=0,
                                reference_measure=f1_score,
                                n_samples=15,
                                batch_size=10,
                                known_drifts=known_drifts)

# Set up a pipeline for a prequential evaluation of the classifiers.
pipeline = PrequentialPipeline(data_loader=data_loader,
                               predictor=models,
                               prediction_evaluator=evaluator,
                               n_max=data_loader.stream.n_samples,
                               batch_size=10,
                               n_pretrain=100)

# Run the experiment.
pipeline.run()
```

Figure 3: **Conducting Data Stream Experiments in Float.** Here we show the source code for a simple experiment performed with the proposed float evaluation framework. More experiments and a detailed documentation can be found on our Github page.



(a) Concept Drift Detection



(b) Online Feature Selection

Figure 4: **Plotting Concept Drift Detection and Online Feature Selection.** Float provides special plot types for evaluating concept drift detection methods and online feature selection models. Above we compare the detected drifts and the most frequently selected features as described in the simple experiment in Section 7.3.1.

For the following experiments, we no longer provide the corresponding source code in order to maintain brevity. However, the general process of specifying float objects remains the same. In addition, the float documentation contains detailed experiment notebooks for each of the modules used.

We start by comparing two popular concept drift detection methods: an Adaptive Sliding Window (ADWIN) [50] and a Page-Hinkley test [51]. As before, we want to use the corresponding implementation of scikit-multiflow. Float also offers wrapper classes for concept drift detectors from related libraries. We specify a *ChangeDetectionEvaluator* object to compute the detected change rate (Eq. (4)), the false discovery rate (Eq. (5)) and the delay. We set a window size of 500 within which we count a drift alarm as a true positive. Float allows the comparison of multiple drift detection models using a special plot type, as shown in Figure 4. With a detected change rate of 0.33%, a false discovery rate of 0.85% and an average delay of 852 observations, the Page-Hinkley test outperforms ADWIN in the TüEyeQ dataset, although no model performs particularly well. Indeed, if we use Page-Hinkley to reset the Perceptron (which can be done by setting the *reset_after_drift* parameter of the *SkmultiflowClassifier* to True), we get no improvement in the average F1 score. Hence, we continue without concept drift detection.

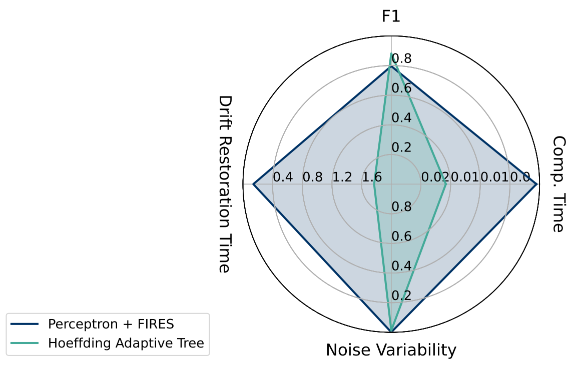


Figure 5: **Comparing Multiple Properties via the Spider Chart.** The float visualisation module contains a spider chart that provides a concise comparison of competing models in terms of various properties. Here we show a summary of the experimental results described in Section 7.3.1.

Finally, we would like to investigate whether we can achieve improvements in any of the performance measures through online feature selection. Accordingly, we compare the online feature selection models FIRES [25] and OFS [52]. Both model implementations are provided by float. As before, float includes a dedicated *FeatureSelectionEvaluator*, which we use to compute the feature set stability (Eq. (7)) of each approach. In this example, we want to select 25 features. The most frequently selected features can again easily be compared with the float visualization module (see Figure 4). The FIRES model outperforms OFS in terms of the feature set stability (0.99 for FIRES and 0.86 for OFS). Moreover, FIRES improves the drift restoration time from 0.5 to 0.4, while OFS worsens the value to 1.08. For the Perceptron, feature selection also leads to a small improvement in the average F1 score. Since the Hoeffding Adaptive Tree performs implicit feature selection, a dedicated feature selection model does not improve performance. Therefore, in our final configuration, we compare the Perceptron in combination with FIRES with the stand-alone Hoeffding Adaptive Tree. We use the spider chart of float to compare both models one last time with regard to various criteria (see Figure 5). In our example, the Perceptron with FIRES has advantages in terms of computation time and drift restoration time, but performs worse than the Hoeffding Adaptive Tree regarding the F1 measure. Both models show little variability with noisy inputs. The proposed float framework allowed us to compare these models in a standardized way and with little effort.

8 Conclusion

Evolving data streams are found in most large-scale and everyday web applications. In this work, we revisited the challenges of evaluating machine learning methods for dynamic data streams. We proposed a comprehensive set of evaluation properties and performance measures that, unlike previous work, extend to the specific tasks of online feature selection and concept drift detection. To enable a more transparent and standardized comparison of online learning methods, we introduced float. Float is a modular and extensible Python framework that can automate major parts of the simulation and evaluation process and provides a flexible basis for extensive benchmarking. The experiments shown in this paper only give a brief impression of the power and versatility of float. We believe that our work can serve as an important reference for the evaluation of online learning models. With this in mind, we hope that this work will help raise awareness of the importance and practical use of online machine learning and data streams.

Abbreviations
NV: Noise Variability; DPD: Drift Performance Deterioration; DRT: Drift Restoration Time; DCR: Detected Change Rate; FDR: False Discovery Rate; MD: Mean Delay; MTFA: Mean Time Between False Alarms; MTR: Mean Time Ratio; FSS: Feature Set Stability; ADWIN: Adaptive Windowing; FIRES: Fast, Interpretable and Robust feature Evaluation and Selection; OFS: Online Feature Selection.

Declarations
Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Availability of data and materials
The TüEyeQ data set analysed during the current study is available in the Harvard Dataverse repository, `https://doi.org/10.7910/DVN/JGOCKI` [47, 48]. The float framework presented in this paper is available at `https://github.com/haugjo/float` and `https://pypi.org/project/float-evaluation/`. The documentation of float is available at `https://haugjo.github.io/float/`.

Competing interests
The authors declare that they have no competing interests.

Funding
Not applicable.

Authors' contributions
JH conceived this study and drafted the manuscript. JH and ET implemented the float framework. JH and GK revised the manuscript. All authors read and approved the final manuscript.

Acknowledgements
Not applicable.

Author details
Data Science and Analytics, University of Tuebingen, Tuebingen, Germany.

References

1. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., Zhang, G.: Learning under concept drift: A review. IEEE Transactions on Knowledge and Data Engineering 31, no. 12 (2018): 2346-2363 (2018). doi:10.1109/TKDE.2018.2876857. 2004.05785
2. Ditzler, G., Roveri, M., Alippi, C., Polikar, R.: Learning in nonstationary environments: A survey. IEEE Computational Intelligence Magazine **10**(4), 12–25 (2015)
3. Gama, J.: A survey on learning from data streams: current and future trends. Progress in Artificial Intelligence **1**(1), 45–55 (2012)
4. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 71–80 (2000)
5. Holmes, G., Kirkby, R., Pfahringer, B.: Stress-testing hoeffding trees. In: European Conference on Principles of Data Mining and Knowledge Discovery, pp. 495–502 (2005). Springer
6. Bifet, A., Holmes, G., Pfahringer, B., Frank, E.: Fast perceptron decision tree learning from evolving data streams. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 299–310 (2010). Springer
7. Read, J., Bifet, A., Pfahringer, B., Holmes, G.: Batch-incremental versus instance-incremental learning in dynamic and evolving data. In: International Symposium on Intelligent Data Analysis, pp. 313–323 (2012). Springer
8. Barddal, J.P., Enembreck, F.: Learning regularized hoeffding trees from data streams. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, pp. 574–581 (2019)
9. Barddal, J.P., Enembreck, F.: Regularized and incremental decision trees for data streams. Annals of Telecommunications **75**(9), 493–503 (2020)
10. Gama, J.a., Sebastião, R., Rodrigues, P.P.: Issues in evaluation of stream learning algorithms. KDD '09, pp. 329–338. Association for Computing Machinery, New York, NY, USA (2009). doi:10.1145/1557019.1557060. https://doi.org/10.1145/1557019.1557060
11. Gama, J., Sebastião, R., Rodrigues, P.P.: On evaluating stream learning algorithms. Machine Learning **90**(3), 317–346 (2012). doi:10.1007/s10994-012-5320-9
12. Bifet, A., Read, J., Pfahringer, B., Holmes, G., Žliobaitė, I.: CD-MOA: Change detection framework for massive online analysis, pp. 92–103. Springer (2013). doi:10.1007/978-3-642-41398-8_9

13. Krawczyk, B., Minku, L.L., Gama, J., Stefanowski, J., Woźniak, M.: Ensemble learning for data stream analysis: A survey. Information Fusion **37**, 132–156 (2017). doi:10.1016/j.inffus.2017.02.004
14. Domingos, P.M., Hulten, G.: Catching up with the data: Research issues in mining data streams. In: DMKD (2001)
15. Montiel, J., Read, J., Bifet, A., Abdessalem, T.: Scikit-multiflow: A multi-output streaming framework. The Journal of Machine Learning Research **19**(1), 2915–2914 (2018)
16. Montiel, J., Halford, M., Mastelini, S.M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H.M., Read, J., Abdessalem, T., et al.: River: machine learning for streaming data in python (2021)
17. Webb, G.I., Hyde, R., Cao, H., Nguyen, H.L., Petitjean, F.: Characterizing concept drift. Data Mining and Knowledge Discovery **30**(4), 964–994 (2016)
18. Bifet, A., de Francisci Morales, G., Read, J., Holmes, G., Pfahringer, B.: Efficient online evaluation of big data stream classifiers. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York, NY, USA (2015). doi:10.1145/2783258.2783372
19. Gonçalves, P.M., de Carvalho Santos, S.G.T., Barros, R.S.M., Vieira, D.C.L.: A comparative study on concept drift detectors. Expert Systems with Applications **41**(18), 8144–8156 (2014). doi:10.1016/j.eswa.2014.07.019
20. Ramírez-Gallego, S., Krawczyk, B., García, S., Woźniak, M., Herrera, F.: A survey on data preprocessing for data stream mining: Current status and future directions. Neurocomputing **239**, 39–57 (2017). doi:10.1016/j.neucom.2017.01.078
21. Gomes, H.M., Barddal, J.P., Enembreck, F., Bifet, A.: A survey on ensemble learning for data stream classification. ACM Computing Surveys **50**(2), 1–36 (2017). doi:10.1145/3054925
22. Kolajo, T., Daramola, O., Adebiyi, A.: Big data stream analysis: a systematic literature review. Journal of Big Data **6**(1) (2019). doi:10.1186/s40537-019-0210-7
23. Haug, J., Broelemann, K., Kasneci, G.: Dynamic model tree for interpretable data stream learning. arXiv preprint arXiv:2203.16181 (2022)
24. Nogueira, S., Sechidis, K., Brown, G.: On the stability of feature selection algorithms. J. Mach. Learn. Res. **18**(1), 6345–6398 (2017)
25. Haug, J., Pawelczyk, M., Broelemann, K., Kasneci, G.: Leveraging model inherent variable importance for stable online feature selection. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, New York, NY, USA (2020). doi:10.1145/3394486.3403200
26. Brzezinski, D., Stefanowski, J.: Prequential AUC: properties of the area under the ROC curve for data streams with concept drift. Knowledge and Information Systems **52**(2), 531–562 (2017). doi:10.1007/s10115-017-1022-8
27. Haug, J., Kasneci, G.: Learning parameter distributions to detect concept drift in data streams. In: 2020 25th International Conference on Pattern Recognition (ICPR). IEEE, USA (2021). doi:10.1109/icpr48806.2021.9412499
28. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence **267**, 1–38 (2019). doi:10.1016/j.artint.2018.07.007
29. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Computing Surveys **51**(5), 1–42 (2019). doi:10.1145/3236009
30. Ntoutsi, E., Fafalios, P., Gadiraju, U., Iosifidis, V., Nejdl, W., Vidal, M.-E., Ruggieri, S., Turini, F., Papadopoulos, S., Krasanakis, E., *et al.*: Bias in data-driven artificial intelligence systems—an introductory survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **10**(3), 1356 (2020)
31. Pawelczyk, M., Bielawski, S., Van den Heuvel, J., Richter, T., Kasneci, G.: Carla: A python library to benchmark algorithmic recourse and counterfactual explanation algorithms (2021)
32. Carvalho, D.V., Pereira, E.M., Cardoso, J.S.: Machine learning interpretability: A survey on methods and metrics. Electronics **8**(8), 832 (2019). doi:10.3390/electronics8080832
33. Bibal, A., Frénay, B.: Interpretability of machine learning models and representations: an introduction. In: ESANN (2016)
34. Moshkovitz, M., Yang, Y.-Y., Chaudhuri, K.: Connecting interpretability and robustness in decision trees through separation. In: International Conference on Machine Learning, pp. 7839–7849 (2021). PMLR
35. Du, M., Liu, N., Hu, X.: Techniques for interpretable machine learning. Communications of the ACM **63**(1), 68–77 (2019). doi:10.1145/3359786
36. Ribeiro, M.T., Singh, S., Guestrin, C.: "why should i trust you?" explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144 (2016)
37. Lundberg, S.M., Lee, S.-I.: A unified approach to interpreting model predictions. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 4768–4777 (2017)
38. Kasneci, G., Gottron, T.: Licon: A linear weighting scheme for the contribution ofinput variables in deep artificial neural networks. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, pp. 45–54 (2016)
39. Hooker, S., Erhan, D., Kindermans, P.-J., Kim, B.: A benchmark for interpretability methods in deep neural networks. In: Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc., Red Hook, NY, USA (2019)
40. Haug, J., Zürn, S., El-Jiz, P., Kasneci, G.: On baselines for local feature attributions. AAAI 2021 Workshop on Explainable Agency in Artificial Intelligence (2021). 2101.00905
41. Bommert, A., Lang, M.: stabm: Stability measures for feature selection. Journal of Open Source Software **6**(59), 3010 (2021)
42. Kalousis, A., Prados, J., Hilario, M.: Stability of feature selection algorithms. In: Fifth IEEE International Conference on Data Mining (ICDM'05). IEEE, USA (2005). doi:10.1109/icdm.2005.135
43. Masud, M.M., Chen, Q., Gao, J., Khan, L., Han, J., Thuraisingham, B.: Classification and novel class detection of data streams in a dynamic feature space. In: Machine Learning and Knowledge Discovery in Databases, pp.

337–352. Springer, ??? (2010). doi:10.1007/978-3-642-15883-4_22

44. Sethi, T.S., Kantardzic, M.: On the reliable detection of concept drift from streaming unlabeled data. Expert Systems with Applications **82**, 77–99 (2017)
45. Manapragada, C., Gomes, H.M., Salehi, M., Bifet, A., Webb, G.I.: An eager splitting strategy for online decision trees. arXiv preprint arXiv:2010.10935 (2020)
46. Souza, V.M., dos Reis, D.M., Maletzke, A.G., Batista, G.E.: Challenges in benchmarking stream learning algorithms with real-world data. Data Mining and Knowledge Discovery **34**(6), 1805–1858 (2020)
47. Kasneci, E., Kasneci, G., Appel, T., Haug, J., Wortha, F., Tibus, M., Trautwein, U., Gerjets, P.: Tüeyeq, a rich iq test performance data set with eye movement, educational and socio-demographic information. Scientific Data **8**(1), 1–14 (2021)
48. Kasneci, E., Kasneci, G., Appel, T., Haug, J., Wortha, F., Tibus, M., Trautwein, U., Gerjets, P.: TüEyeQ, a rich IQ test performance data set with eye movement, educational and socio-demographic information. Harvard Dataverse, USA (2020). doi:10.7910/DVN/JGOCKI
49. Bifet, A., Gavalda, R.: Adaptive learning from evolving data streams. In: International Symposium on Intelligent Data Analysis, pp. 249–260 (2009). Springer
50. Bifet, A., Gavalda, R.: Learning from time-changing data with adaptive windowing. In: Proceedings of the 2007 SIAM International Conference on Data Mining, pp. 443–448 (2007). SIAM
51. Page, E.S.: Continuous inspection schemes. Biometrika **41**(1/2), 100–115 (1954)
52. Wang, J., Zhao, P., Hoi, S.C., Jin, R.: Online feature selection and its applications. IEEE Transactions on knowledge and data engineering **26**(3), 698–710 (2013)