Continual learning from stationary and non-stationary data

Lukasz Korycki
Virginia Commonwealth University

CONTINUAL LEARNING FROM STATIONARY AND NON-STATIONARY DATA

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

ŁUKASZ KORYCKI

Ph.D. Candidate

Director: Bartosz Krawczyk,

Assistant Professor, Department of Computer Science

Virginia Commonwealth University

Virginia Commonwealth University

Richmond, Virginia

May, 2022

Acknowledgements

This dissertation could not have been written without the guidance and active support of my advisor – Dr. Bartosz Krawczyk. It started with the enthusiasm for artificial intelligence that he instilled in me during my freshman year in Poland and it continued for all of these years through our countless discussions that shaped me as a researcher. His pure scientific optimism allowed me to believe that I can be not only the reader but also the writer.

There are no words to fully express my gratitude towards my parents. Their love and enormous effort allowed me to follow the direction I chose and realize my dreams. My mother always impeccably believed that I can achieve my goals, helping me develop the same conviction and encouraging to move forward regardless of my failures and doubts. And it was the hard work of my father that gave me the opportunity to focus on my growth. I owe them everything.

I would also like to thank my brother, who added so many bright colors to our childhood, and my friends in Poland and in the USA, who accepted me as I am and with whom I share so many great memories. I want to thank Damian Wiecek, Mateusz and Magda Koguc, Grzegorz Drzazga and Bartosz Polek for almost 20 years of friendship, Michal and Dominika Polanscy and Dominik and Agata Mrzyglod for all of the great student years we could spend together, and Adam Ziemba, Andriy Mulyar and Jorge Gonzalez Lopez for making the USA feel like home.

The last paragraph is reserved for my precious wife, Agnieszka, who has always been next to me, and whose love makes me a better person every day. I am happy to have you by my side.

Table of Contents

List of Tables

List of Figures

Abstract

Continual learning aims at developing models that are capable of working on constantly evolving problems over a long-time horizon. In such environments, we can distinguish three essential aspects of training and maintaining machine learning models – incorporating new knowledge, retaining it and reacting to changes. Each of them poses its own challenges, constituting a compound problem with multiple goals.

Remembering previously incorporated concepts is the main property of a model that is required when dealing with stationary distributions. In non-stationary environments, models should be capable of selectively forgetting outdated decision boundaries and adapting to new concepts. Finally, a significant difficulty can be found in combining these two abilities within a single learning algorithm, since, in such scenarios, we have to balance remembering and forgetting instead of focusing only on one aspect.

The presented dissertation addressed these problems in an exploratory way. Its main goal was to grasp the continual learning paradigm as a whole, analyze its different branches and tackle identified issues covering various aspects of learning from sequentially incoming data. By doing so, this work not only filled several gaps in the current continual learning research but also emphasized the complexity and diversity of challenges existing in this domain. Comprehensive experiments conducted for all of the presented contributions have demonstrated their effectiveness and substantiated the validity of the stated claims.

CHAPTER 1

INTRODUCTION

Almost all contemporary hardware and software products constantly exchange information between each other not only to fulfill their basic tasks but also to provide improved performance and extend the scope of their functionalities. As the rate and diversity of data generators increases [1], the horizon of potential improvements and new ideas seems to be limitless. Companies, developers and researchers compete with each other to discover these new directions and either provide further improvements to already known problems (e.g., market analysis based on social media content [2] or autonomous driving adjustments based on information collected from individual cars [3]), or propose completely new perspectives (e.g., blockchain [4], metaverse [5]). The products of their work constitute a tightly interconnected and dynamic reality where we can process and connect *the dots* practically indefinitely.

Due to the aforementioned observations, modern machine learning calls for algorithms that are able not only to generalize patterns from a finite dataset but also to continually maintain or improve their performance, as well as extend their capabilities while accumulating knowledge from constantly arriving data. Continual (or lifelong) learning aims at developing models that will be capable of working on constantly expanding problems over a long-time horizon [6]. Such models should keep utilizing new instances (online or data-incremental learning), new classes (class-incremental learning), or even new tasks (multi-task learning) [7]. Whenever new information becomes available it must be incorporated into the continual learning model to expand its knowledge base and make it suitable for predictive analytics over a new, more complex view of the analyzed problem. This requires a flexible model structure capable of continual storage of incrementally arriving data.

In such a setting, just like for humans, there are three fundamental aspects of training and maintaining machine learning models – incorporating new knowledge, retaining it and adapting to changes [8]. They pose their own specific challenges, creating a complex multi-objective problem. While the first of them is essential to any machine learning algorithm, either offline or online, the two remaining ones can be considered specific to continual learning.

Remembering the previously incorporated concepts (classes or tasks) is the main characteristic of a model that is needed when dealing with stationary data sources [9, 10]. Unfortunately, most of the state-of-the-art machine learning models seem to be unfit for incremental learning scenarios. The reason for this is that they were designed as offline, batch-based algorithms entirely focusing on a single data source given at a time. As a consequence they tend to overwrite the previously learned knowledge with the new one, leading to a phenomenon known as catastrophic forgetting [11, 12]. This problem has been explored mostly for (deep) neural networks, which, over the last couple of years, has been resulting in the development of several new approaches to mitigating the issue [9]. Nevertheless, we can still find this research domain to be in its early stage, having the main focus on very constrained scenarios with many limitations [7, 13] and with very little attention given to different than neural networks models.

Learning in non-stationary environments comes with additional requirements and challenges [14, 15]. Here, we have to take into consideration the fact that our data generators are not static, therefore, they can generate different data distributions depending on when we sample from them. In such scenarios, we have to be ready to deal with dynamic, non-stationary data sources which are usually characterized by concept drifts [1]. Learning from such data sources requires models to selectively forget outdated decision boundaries and adapt to new concepts. This subject has been addressed mostly in the context of data stream mining [14, 16], which focuses on the most generalized form of continual learning involving online, task-free scenarios with revisiting concepts [17, 18]. Due to the broad generalization, in such environments, one has to deal with various compound problems related to different forms of dynamics and resource management [19]. The natural consequence of relaxing all of the constraints is that there is a plethora of potential research directions and problems that have not been properly addressed yet [14, 19, 20].

Finally, if we carefully observe the current research in the deep continual learning domain we will notice that almost all of the published works are entirely concentrated on preventing catastrophic forgetting [6, 9, 21]. There is a strong emphasis on aggregating only stationary concepts, which is a limited view of the required flexibility. The problem of learning from drifting data is being addressed by data stream mining, but, on the other hand, it is almost exclusively focused on the problem of change adaptation and shallow learning, ignoring the issue of unwanted forgetting (insufficient retention) [14, 22]. A simple conclusion is that there are practically no works that attempt to tackle both tasks at once. Those published trivialize the problem since continual learning should not be solely focused either on accumulating observed patterns nor on forgetting outdated information. Both mechanisms are needed to create a complete generic learning paradigm, robust to difficulties present in real-world problems [8, 18].

1.1 Research goals and problems

The general purpose of this dissertation is to: **identify and address deficiencies and shortcomings of the current state-of-the-art machine learning**

models for continual learning. The main hypothesis is that for both stationary and non-stationary data the learning algorithms and methods still have not been sufficiently explored. They lack proper generalization to more realistic scenarios and there is a lot of room for additional performance improvements or alternative solutions. The presented work forms an attempt to explore and implement potential enhancements, involving modifications, extensions and novel approaches, in order to make the continual learning models more reliable, flexible and overall applicable to real-world problems. More specific **research goals** for each of the aforementioned subdomains are given as follows.

RG1: Improving continual learning from stationary data. Most of the works on continual learning from stationary data are exclusively focused on handling catastrophic forgetting in neural networks. In addition, they usually avoid more complex solutions and rely on strong assumptions, which impairs the retention of models and stops us from scaling continual learning up (e.g., with the number of classes) to utilize it in more feasible settings. This dissertation attempts to open new directions in the research focused on tackling catastrophic forgetting by exploring alternative machine learning models that could be applied to the given scenarios.

RG2: Improving continual learning from non-stationary data. Analogously to the previous goal, the main focus can be put on addressing complex scenarios of adaptive learning from drifting distributions, involving different problems at the same time (real-world scenarios), and more in-depth analysis of existing approaches to provide further improvements of already proposed ideas. This work considers the following directions: learning and concept drift detection on a budget, handling dynamic imbalance in data streams and improvements of the existing state-of-the-art streaming methods.

RG3: Addressing the lack of holistic approaches. The improvements of methods providing stability, when data is stationary, and plasticity, when learning new concepts or adapting to changes, are necessary yet not sufficient to significantly advance continual machine learning and provide generic frameworks. The main difficulty can be found in combining these abilities within a single learning algorithm, since, in such scenarios, we have to ensure optimal proportions between remembering and forgetting instead of focusing on only one aspect. This work addresses the given problem by introducing a new benchmarking approach for deep learning problems along with a method that explicitly considers catastrophic forgetting and concept drift.

1.2 Motivation

Intellectual merit. Advancing continual learning to a more complete form is of significant importance since it is potentially a more generic learning paradigm than the current state of the art – offline batch-based learning. Ideally, continual learning should be providing at least the same capabilities (convergence) and, in addition to that, even more of them (ability to handle dynamic data). Acknowledging non-stationary classes and attempting to create adequate continual learning systems opens new research perspectives, since we have to focus not only on the classes themselves but also on existing dynamics in data. Exploring those areas may give us a much better understanding of phenomena around us. Most of them are not static in nature, thus without looking into their dynamics we will not be able to truly comprehend them.

Broader impacts. Advanced continual machine learning may play a crucial role in providing impactful benefits to society, including: better accessibility of specialized services (it is enough to update a complex model with specific smaller tasks without the need of having access to all expensive data), local and global empowerment (continual transfer learning may allow for benefiting in both directions – *from everyone to one and from one to everyone*), more easily achievable fairness (no need to retrain whole models to reduce imbalance/bias). Deploying online models will allow for the development of new very important applications that should constantly maintain high accuracy and reactivity, e.g., those related to medicine, which may elevate the quality of life for the entire population. One of the most recent areas for such an application could be COVID modeling [23, 24].

1.3 Structure

The structure of the dissertation is given as follows. After the general introduction of the continual learning ideas, presented in Chapter 1, Chapter 2 formalizes basic concepts and taxonomy related to the subject in the context of learning from both stationary and non-stationary data sources. In addition, it introduces research directions and works already published in the given area, emphasizing problems that have not been sufficiently (or at all) covered yet. The next two chapters focus on continual learning from stationary data (**RG1**) by introducing two alternative approaches – a hybrid of the GMM model (Chapter 3) and streaming decision trees (Chapter 4) with a deep learning extractor. After that, the subsequent chapters address the problem of learning from non-stationary data (**RG2**) – intensive exploitation of scarce examples of temporary concepts (Chapter 5), online oversampling for drifting concepts (Chapter 6), concept drift detection for imbalanced data (Chapter 7), change detection under strictly limited supervision (Chapter 8) and dynamically diversified ensembles (Chapter 9). Finally, Chapter 10 introduces a holistic approach to learning from stationary and non-stationary data (**RG3**). At the end, a final summary consisting of conclusions and future works is presented in Chapter 11.

CHAPTER 2

BACKGROUND AND RELATED WORKS

2.1 Continual learning

In continual learning we consider a sequence (continuum or stream) of data that potentially infinitely comes to our incremental machine learning model from a source. We can call it online learning, as opposite of the offline procedures that work only with a finite, static dataset [1].

Data sequences. The sequence S, used for learning, may consist of different data blocks D_i, giving us $S = \langle D_1, D_2, ..., D_n, ...\rangle$, depending on a processing framework or considered scenario, as given in Fig. 1. Here, we can distinguish three main settings [7]: (i) **task-incremental**, where D_i is a task T_i containing data for a finite set of classes $T_i = \langle C_1, C_2, ..., C_m\rangle$, (ii) **class-incremental**, where D_i is a class C_i containing data only for a given class $C_i = \langle(\boldsymbol{x}_1, y), (\boldsymbol{x}_2, y), ..., (\boldsymbol{x}_k, y)\rangle$, where y is a class label, and (iii) **data-incremental** or **domain-incremental**, where we do not distinguish any tasks or classes and simply accept batches of new instances $B_i = \langle(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_k, y_k)\rangle$ as our data blocks D_i.

Other taxonomies. In addition to the most fundamental taxonomy, it is worth elaborating on some details or different perspectives proposed in other works, coming from slightly different research domains, that call for unification. First of all, for data-incremental scenarios, we can distinguish either **batch-incremental** or **online/streaming learning** [1], where for the latter we basically define a single data block as a single instance $D_i = (\boldsymbol{x}_i, y_i)$. In practice, mini-batches are usually treated

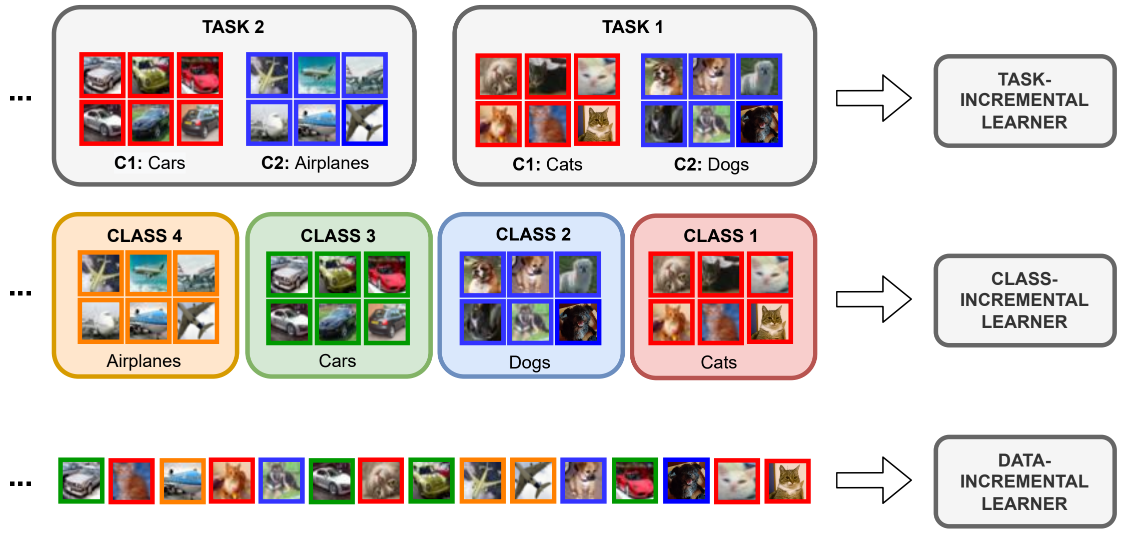


Fig. 1.: Different continual learning settings: task-incremental, class-incremental and data-incremental.

as a form of online/streaming processing. Secondly, streaming scenarios can also be seen as a special case of the online settings. Here, we have to acknowledge very restrictive time and memory requirements, usually boiling down to high-throughput and discarding instances after processing them [1]. Finally, while the task-incremental scenarios assume that task boundaries and ids are given as a part of the learning procedure [9], **task-agnostic** or **task-free** works apply relaxations to this very strong assumption. They either state that tasks are not defined a priori, in which case an algorithm has to discover and learn them, or assume that tasks do not exist at all [17, 25]. One should easily notice that, from the perspective of data processing, these scenarios (especially the latter one) are practically synonymous to the data-incremental approaches.

Revisiting. One of the crucial aspects of incremental learning is whether we allow for revisiting previously seen observations, in the form of tasks ($T_i = T_j$), classes ($C_i = C_j$) or simply instances ($\boldsymbol{x}_i = \boldsymbol{x}_j$), where $i < j$. It is essential for learning from non-stationary data sources and enables cumulative improvements in stationary environments [26]. Furthermore, it is worth noting that, task-incremental and class-incremental methods that allow for revisiting can sometimes be seen as data-incremental approaches, since we effectively switch to updates at the level of each task or class instead of just incrementally adding them to the model.

Class distributions. We assume the supervised learning scenario with classification goal and thus we will define each instance as $\boldsymbol{x}_i \sim p_i(x^{(1)}, \cdots, x^{(d)}, y) = p_i(\boldsymbol{x}, y)$, where $p_i(\boldsymbol{x}, y)$ is a joint distribution of i-th instance, defined by a d-dimensional feature space and assigned to class y. Each instance is independent and drawn randomly from a probability distribution $\Psi_i(\boldsymbol{x}, y)$, which, for the purpose of the further explanations, we call a **class generator** of a data source. Obviously, the same definition of the generator stands for task-incremental, class-incremental and data-incremental scenarios, since in all of them sources generate data per class. Depending on whether the generator is **stationary or non-stationary**, we encounter stationary or non-stationary data[1].

2.2 Learning from stationary data

The most basic continual learning scenarios assumes that all incoming data are generated by stationary data sources, therefore, for any point in time t and any class y we can state that $p_t(\boldsymbol{x}, y) = p_{t+1}(\boldsymbol{x}, y)$. In such cases, where class distributions remain static, our focus should be on effective incorporation of the arriving data and retention of the acquired knowledge to provide sufficient stability to the learning process [12].

[1]Some works consider stationary and non-stationary data based on the observed distributions, instead of source generators. We do not use this definition, since in such a case practically all continual learning scenarios could be treated as non-stationary.

2.2.1 Catastrophic forgetting

The main problem that learning algorithms will encounter here is **catastrophic forgetting** [6]. In fact, most of the machine learning models cannot handle long incremental learning sequences and they tend to overfit their learning parameters to the most recent observations without considering the significance of the previously obtained knowledge. This issue has been identified mainly in the context of neural networks and their gradient-based learning, but can also be observed for other types of models [27]. Nevertheless, almost all of the attention in the related research has been given to the incremental deep neural networks, which resulted in multiple different approaches dedicated to alleviating catastrophic forgetting in these models. The description and examples for each of them can be found in the next paragraph.

2.2.2 Preserving knowledge

Experience replay. The most straightforward approaches involve replaying instances of previously seen tasks or classes while learning new ones. The mixed composition of old and recent examples helps preserving subspaces modeled for the former while simultaneously allowing for learning how to discriminate it from the latter [28]. To make such methods feasible in potentially infinite continual learning scenarios, such methods usually rely only on a small memory buffer consisting of the most representative examples. Several different strategies for selecting the exemplars have been proposed, including simple uniform sampling, k-means or herding for the class mean approximation [29], as well as more sophisticated methods including diversification of the memory buffer based on classification uncertainty and synthetic augmentation [30], cardinality-constrained bi-level optimization for core-sets construction [31] or gradient-driven criteria proposed for GEM [32], A-GEM [33] and GSS [34]. There are also different approaches when it comes to the form of the examples being stored – iCaRL stores raw instances which are dynamically transformed by an adaptive extractor [29], REMIND maintains a buffer of compressed representations generated with an auto-encoder to provide better memory management [35], while saliency maps (support regions) are added to the buffer in [36] to ensure that a model also remembers reasoning (explanations) behind each example. Furthermore, the examples can also be synthetically or semi-synthetically generated – they can be parametrized as so-called mnemonics and trainable in a meta-learning optimization procedure [37], and pillars consolidation (preserving the known) combined with contrastive loss (learning new) can provide high-quality prototypes that can be used as the replay examples [26]. Finally, it is also possible to avoid storing any instances in the memory and generate them on-the-fly. This can be achieved by either using the inversion of a classifier [38], or by utilizing popular generative models [39, 40, 41, 42, 43]. The experience replay methods have been proven as very solid baselines capable of keeping the incremental model in low-loss training regions [28] and providing competitive results even using the most simplistic configurations [13], even if they are still missing some crucial elements [44]. On the other hand, even though the size of the buffer can be effectively reduced, their memory complexity still grows linearly or sublinearly with the number of new tasks and classes, and, in addition to that, they tend to overfit towards the very few instances stored in the buffer [28]. Both obstacles may be prohibitive for more complex problems.

Regularization. Instead of putting instance-level constraints on the learning directions, we can apply direct adjustments to the loss using dedicated regularization terms. The most commonly used approach involves utilizing the knowledge-distillation loss [45] combined with standard cross-entropy. Many of more advanced methods are based on the task-incremental LwF algorithm [46], which uses the former loss to preserve the knowledge of older concepts by enforcing more stable extractor outputs while incorporating new tasks [47, 48]. Alternative approaches involve maintaining importance weights to distinguish parameters that are crucial for the retention. Here, we can focus on neuron weights, like in EWC [49] incorporating the Fisher matrix into the training loss to preserve the most important weights per task, contribution to the global loss, like in SI [50] using the synapse state space accumulating relevant information, or magnitude of gradient changes, like in MAS [51] tracking the output sensitivity similarly to the Hebb's learning. Some works emphasize the the importance of controlling the outputs [52], especially in the context of providing sparse representations, instead of weights sparsity to ensure additional learning space [53], while others motivate anticipatory (proactive) learning instead of post-hoc knowledge safeguarding, based on self-supervised proxy tasks involving input transformations [54] or unsupervised reconstruction loss [55], to provide more generic and reusable knowledge base. Similar goal can be achieved through adversarial continual learning where task-invariant features (trained against the task discriminator) are maintained [56]. Finally, one can also apply regularization directly to all layers, like in PODNet [57], or focus on alleviating critical changes in the feature space identified in [26] and addressed with topology-preserving Hebbian graph-based learning. While regularization-based methods are less prone to memory complexity (usually they have to store only importance weights), one should notice that most of them have been applied only to task-incremental scenarios with shared feature extractors, task-specific output layers and task oracles [46, 58]. These are very strong assumptions and while there have been attempts to relax them [59, 60, 61], these methods generally cannot be used in more realistic class-incremental or data-incremental scenarios (if they do not use memory buffers), since they cannot learn how to discriminate new instances from the older ones [7].

Bias correction. The analysis of the classification layers in continual neural networks revealed that one of the main reasons of catastrophic forgetting is a weight bias (higher magnitudes) towards new classes [62, 63]. The skewness correction seems to be crucial in the context of large-scale class-incremental scenarios and has been addressed, for example, by using simple weight scaling [63], linear bias correction [62] or employing the cosine classifier in the output layer combined with intra-class separation and additional geometric constraints [64, 65]. In addition, it has been shown that KD-loss contributes to the new-old bias problem, since it takes only old classes logits into loss, leading to the situation where mistakes between new and old classes are less important that mistakes between previously observed classes [63]. While this group of methods improves incremental learning performance, it does not solve the problem of missing discrimiantive capabilities without a memory buffer.

Masking. Another group of methods employs masking to isolate parameters per task to keep them static when learning new ones. They usually maintain a single fixed or expandable backbone, upon which task-specific subnets are built, and binary masks selecting weights trainable for a given task, like in Piggyback [66], PackNet [67] or PathNet [68]. Since the backbone networks are usually implemented as fixed-size extractors, it is essential to limit the size of each mask (subnet) in order to be able to scale with a large number of tasks. This issue can be addressed with pruning, which releases less important weights, that do not affect the performance in a significant way, to make more space for new tasks [67, 69]. Some methods allow for extending the backbone network with new weights [70]. Finally, although these methods provide almost perfect isolation of learning parameters and are very efficient in preventing catastrophic forgetting, they were applied only in task-incremental scenarios, they struggle with scaling up to higher numbers of tasks and are very limited when it comes to forward and backward transfer, which has been somehow addressed in [71] considering similar and dissimilar tasks to decide whether trigger or not transfer learning.

Dynamic structures. Isolating parameters per task can also be achieved through dynamic expansion of the network. Most basic methods, like progressive neural networks [72], simply add new column weights and lateral connections (adapters) while, at the same time, fixing the previously created ones. Such approaches tend to relax the memory constraints too much (compared, for example, with masking that attempts to pack everything into a fixed architecture), since they grow linearly with the number of tasks. The expansion can be dynamically adjusted based on the complexity of a problem, like in DEN [73] that finds relevant neurons using BFS and adds new units if loss significantly increases, DER [74] that expands its feature extractor and prunes it using channel-level mask-based shrinking, or modular networks [75] that maintain modules representing atomic skills and dynamically decide whether to reuse older units or add new ones, as well as how to combine them. Some methods utilize continuous neural search approaches to look for the most efficient potential expansions [76]. Most of the algorithms combine the dynamic structure growth with other techniques to make the expansion even more efficient, including applying sparsity loss or detecting critically dissimilar tasks [73]. Analogously to masking, these methods have been applied almost exclusively in task-incremental scenarios. On the other hand, they offer much better scalability and natural opportunities for obtaining forward transfer learning when utilizing previously created units [73, 75].

Ensembles. Probably the most extreme case of dynamic expansion is based on the ensemble techniques where a separate model is created for each task (or class). Two main improvement directions focus either on providing a reliable task selector, e.g. using nearest-mean [59] or an autoencoder in ExpertGate [77], or compressing the task learners, e.g., by using rank-one matrices and Hadamard product in BatchEnsemble [78] or weight rectification and scaling [79]. The ensemble-based solutions can also be implemented in class-incremental scenarios by using one-class classifiers, e.g. based on VAE [80] or holistic regularization, parameter transfer and contrastive information extraction [81]. Last but not least, diverse committees have been used to obtain more generic, complementary and robust decision boundaries helping regularize forgetting [82, 65]. The ensemble-based methods struggle with memory efficiency and decision space unification (class-incremental scenarios), but, on the other hand, they may offer a lot of improvements known from the vast research focused on offline settings, which seemingly have not been fully utilized in continual learning.

Other approaches. There are several interesting works that either cannot be directly put in any of the main categories, or focus on more specific continual learning scenarios. One group of such methods concentrates on working under strictly limited supervision, in few-shot learning settings [83, 84], for example, by forcing gradients to adapt to auxiliary simulated processes [85], employing neural gas and class centroids [86], or addressing the weaknesses of the KD-loss when very few labels are available [87]. A very promising category of methods consists of approaches utilizing meta-learning and hypernetworks, which usually allow for learning towards more efficient representations [88, 89] or dynamic selection of parameters per task [90, 91, 92, 93]. Other works worth mentioning include: using efficient streaming LDA as a data-incremental classifier [94], employing learning based on rate reduction and white-box ReduNet [95], dynamically combined stable and plastic residual blocks [96], methods utilizing the nearest-mean classifier in class-incremental scenarios [97, 98], or algorithms built around Bayesian and Gaussian components [99, 100].

2.3 Learning from non-stationary data

In many real-world applications data do not fall under stationary assumptions [101]. It is more likely to evolve over time and form temporary concepts, being subject to various types of changes. In such scenarios, at least one of the data source generators is non-stationary and may generate different distributions depending on time t. Therefore, we can distinguish some instances and classes y for which $p_t(\boldsymbol{x}, y) \neq p_{t+1}(\boldsymbol{x}, y)$. The natural consequence of this is that previously modeled decision boundaries may become outdated after a change, creating a situation where a classifier should be able to forget the invalid information and adapt to a new state [15]. When encountering dynamic distributions, the main goal of a learning algorithm is to ensure efficient selective plasticity (flexibility) of a model.

2.3.1 Concept drift

The described phenomenon is known as **concept drift** [1, 14]. It may affect different aspects of data and thus can be analyzed and addressed from multiple perspectives.

Influence on decision boundaries. Firstly, we need to take into account how concept drift impacts the learned decision boundaries, distinguishing between real and virtual concept drifts [102]. The former influences previously learned classification boundaries, decreasing their relevance for newly incoming instances. Real drift affects posterior probabilities $p_t(y|\boldsymbol{x}) \neq p_{t+1}(y|\boldsymbol{x})$ and additionally may impact unconditional probability density functions. It must be tackled as soon as it appears, since it invalidates (to some extent) the underlying classifier. Virtual concept drift affects only the distribution of features $p_t(\boldsymbol{x})$ over time. While it seems less dangerous than

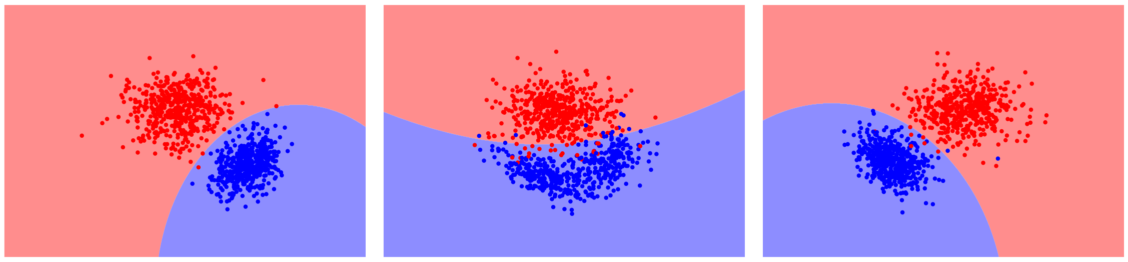

(a) Before drift (b) During drift (c) After drift

Fig. 2.: Example of real concept drift.

real concept drift, it cannot be ignored. Despite the fact that only the values of features change, it may trigger false alarms and thus force unnecessary and costly adaptations.

Locality of changes. It is important to distinguish between global and local concept drifts [103]. The former one affects the entire stream, while the latter one affects only certain parts of it (e.g., regions of the feature space, individual clusters of instances, or subsets of classes). Determining the locality of changes is of high importance, as rebuilding the entire classification model may not be necessary. Instead, one may update only certain parts of the model or sub-models, leading to a more efficient adaptation.

Speed of changes. We can distinguish between three main types of concept drifts [14]: (i) sudden, when instance distribution abruptly changes with t-th example arriving from the stream, (ii) incremental, when we have a continuous progression from one concept to another (thus consisting of multiple intermediate concepts in between), such that the distance from the old concept is increasing, while the distance to the new concept is decreasing, and (iii) gradual, where instances arriving from the stream oscillate between two distributions during the duration of the drift, with the old concept appearing with decreasing frequency.

Recurrence. In many scenarios it is possible that a previously seen concept may reappear over time [14]. One may store models or instances for previously seen concepts in order to speed up recovery rates after a known concept reemerges [104].

Presence of noise. Apart from concept drift, one may encounter other types of changes in data. They are connected with the potential appearance of incorrect information in the stream, and known as blips or noise. The former stand for singular random changes in a stream that should be ignored and not mistaken for a concept drift. The latter stands for significant corruption in the feature values or class labels and must be filtered out in order to avoid feeding false [105] or even adversarial information to the classifier [106].

Feature drift. This is a type of change that happens when a subset of features becomes, or stops to be, relevant to the learning task [20]. Additionally, new features may emerge (thus extending the feature space), while the old ones may cease to arrive.

2.3.2 Change adaptation

The problem of adaptation to non-stationary distributions has been addressed mainly by the data stream mining research community [1, 14, 16], therefore, most of the published works are focused on streaming/online algorithms working in data-incremental regimes. In general, very few methods directly addresses the continual learning perspective, with the main focus given to streaming decision trees.

Adaptive classifiers. In order to be able to adapt to evolving data, classifiers must either use continuous learning to follow the progression of a stream (blind adaptation), or have an explicit information on when to update their model (informed adaptation) [1]. Algorithms based on sliding windows storing only the most recent instances are very popular, allowing for natural forgetting of older instances [107]. The size of the window is an important parameter and adapting it over time seems to yield the best results [108]. Online learners are capable of learning instance by instance, discarding data after it passed the training procedure. Here, streaming decision trees can be considered to be state-of-the-art learning algorithms, including Adaptive Hoeffding Trees [109] and its several improved versions [110]. Alternative approaches involve change detectors that can be paired with any classifier to monitor a state of the stream [111] and decide when to actively update a model. This reduces the cost of adaptation by lowering the number of times when we train the new classifier, but may be subject to costly false alarms or missing changes appearing locally or on a smaller magnitude. Although very few, there are some works that focus directly [18] or indirectly [112] on adapting neural networks to non-stationary environments, including online deep neural networks [113], adaptive exemplar sets combined with online nearest-mean classifier in the output layer [114], evolving prototypes [25] or task-free models based on current task (change) detection [17] greedily adjusting to incoming data.

Dynamic ensembles. Combining multiple classifiers is a very popular and powerful approach for standard learning problems [115, 116]. The technique transferred seamlessly to data stream mining scenarios, where ensemble approaches have displayed a great efficacy [22]. They not only offer improved predictive power, robustness, and reduction of variance, but also can easily handle concept drift and use it as a natural way of maintaining diversity. By encapsulating new knowledge in the ensemble pool and removing outdated models, one can assure that the base classifiers are continuously mutually complementary, while adapting to changes in the stream. There are two main approaches for ensemble design in streaming environments: (i) updating base classifiers or (ii) updating the ensemble setup. The first approach assumes that our ensemble uses classifiers capable of incremental or online learning. New instances arriving from the stream are used to continuously update those classifiers, without adding or removing any models. Popular solutions are based on the usage of online versions of Bagging [117], Boosting [118] or Random Forest [119]. Maintaining diversified base learners in an ensemble is a critical aspect of training effective online committees as it plays a crucial role in improving adaptation and stabilization of the multi-classifier models. It has been shown that diversification should be increased when there is a change and our ensemble suffers from lower performance, and decreased when our model is stable and performs well [120]. The second approach to the ensemble design assumes that we can add new classifiers to the ensemble pool and that outdated or irrelevant classifiers can be removed via a pruning procedure. Classifiers are combined with weights reflecting the time they spent in the ensemble and their current performance [121]. AWE [122], AUE [123] and KUE [116] enable natural encapsulation of changes in data in a form of new classifiers and allow for using any base learner. However, methods using the dynamic ensemble setup react slower to concept drifts than their online counterparts, as they need to gather enough of new instances before updating the classifier.

Concept drift detection. During the last years, several algorithms have been proposed to tackle the problem of drift detection [111]. In general, we can group them, just like classifiers, into supervised, semi-supervised and unsupervised methods. The critical difference between them is that the first one is able to detect changes in class boundaries, while the last one can only indicate drifts in the data distribution [1]. Although, in fact, we look for the former changes, which explains a very high popularity of the supervised detectors, it has been shown that in practice detecting the latter type of drifts may be sufficient enough [124, 125]. The most popular group of supervised drift detectors is constituted by methods that are based on the classification error or accuracy calculated over labeled instances, like DDM [126], EDDM [127] or RDDM [128]. Some of them, in order to decide if a change occurs, apply statistical tests to check whether differences between monitored drift measures are significant, for example, FTDD [129] that triggers detection based on the Fisher's exact test, or WSTD [130] that relies on the Wilcoxon rank sum test. Another popular category of detectors are methods based on metrics within subwindows of a stream, like, for example, STEPD [131] monitoring correct predictions in the older and recent window, or ADWIN [108] – an adaptive sliding window based on the Hoeffding's inequality that inspired multiple novel detectors [132, 133, 134]. An interesting idea is to use ensemble techniques for the purpose of drift detection. Several different heterogeneous and homogeneous committees along with various combining strategies have been evaluated in [135]. Authors of [136] proposed a reservoir of diverse adaptive classifiers and drift detectors to address the problem of temporal optimality of different combinations. Some of very few pure semi-supervised approaches include combining detectors with active learning [137, 138]. Strictly unsupervised drift detectors focus directly on finding differences only in unlabeled data without any additional supervision. It usually boils down to the statistical comparison of two samples of data – from the older and recent chunk [139, 135]. Probably the most sophisticated and precise methods try to go even further and attempt to find exact regions within the feature space that are affected by drifts. We say that, as opposed to the time-based detectors focusing on finding a moment of a drift, these detectors focus on the spatial search [124] utilizing various partitioning schemes [140] or dissimilarity measures [141]. In addition to standard learning scenarios, concept drift detection can also be considered in the context of more complex tasks, like adversarial attacks [142].

Temporal imbalance. In non-stationary continual learning, not only class boundaries can be subject to changes. In fact, class ratios can also change over time, which creates additional learning difficulty [143, 144]. Here, the proportions between classes change dynamically, as well as class roles – a minority may become a majority over time. Long delays between receiving samples of the same class may lead to local imbalance, even if for a much longer period of time the same stream is balanced [145]. Obviously, the relation between class imbalance ratios may change over time. The most complex and challenging scenarios consist of both concept drift and dynamic skewness [143]. One way of overcoming the imbalance problems is to use the algorithm-level adaptation [146]. It is often applied as cost-sensitive internal updates of a single algorithm (perceptron-based RLSCAP [147], CSOAL [148]), adaptation of imbalance-sensitive weights in ensembles (DWMIL [149], ESOS-ELM [150], WELM [151]), or even through the optimization of embeddings [152]. Another very important group of methods for imbalanced data streams consists of those that attempt to solve the problem using an instance-level approach – resampling [153, 154]. Here, oversampling or undersampling methods are dynamically adjusted based on the class ratios to balance the learning process. In [155] two classifiers based on the well-known Learn++.NSE algorithm [156] were proposed – Learn++.CDS utilizing SMOTE for balancing and Learn++.NIE that applies bagging subensembles along with imbalance-sensitive metrics for base learners. Dynamic resampling combined with ensemble-level adaptation [157] was fully utilized in improved online bagging presented in [158, 159]. It is also possible to combine balanced buffers of instances with ensemble techniques as it was done in ROSE [160]. Finally, worth noting that non-stationary class imbalance may affect not only the classifiers but also drift detectors or sampling strategies [161], posing difficult challenges when using more complex methods.

Learning on a budget. Companies generate incomparable amounts of data every second [1] and the assumption that most of the instances can be labeled is rather naive, due to the labeling costs [162]. This problem is especially severe in non-stationary continual learning, since we are forced to constantly strictly control the incoming data distributions and the whole learning process. An intuitive solution to this problem is to limit labeling only to those instances that are likely to provide the best trade-off between cost and information. Active learning focuses on finding such valuable data points that should be labeled [162]. There are several ways of defining a useful instance. It can be, for example, uncertainty of a classifier, representativeness in a data distribution, potential influence on the error or variance reduction [163, 164, 151]. Ensemble techniques can also be used as active learning strategies. They are known as *Query by Committee* [165] methods and they define valuable instances as those for which there is a strong disagreement between voting base learners [166, 167]. Other active learning approaches include unsupervised querying based on deviation from previously modeled distribution [168], sampling according to policy-adaptive submodular functions [169], or aiming at reducing the estimated level of error [170]. The semi-supervised learning methods, which should be seen as a natural choice for the considered scenarios (a lot of unlabeled data and limited supervision), are represented mainly by algorithms based on the incremental cluster-then-label approach [171, 172], online self-labeling [173], graph-based label propagation techniques [174] or online co-training [175]. The unsupervised methods present a different, proactive approach, in which models try to anticipate changes in decision boundaries without labeled instances [176, 177]. Some of the continual learning methods for neural networks address the problem of limited supervision by considering few-shot learning scenarios [25, 114, 112]. Finally, by utilizing unsupervised drift detection methods we may avoid the inconvenience of *spending budget to save it*, which occurs when we use supervised drift detectors to trigger more budget-friendly classifier updates.

2.3.3 Holistic approaches

While the main distinctive goal of methods designed for non-stationary data is to effectively handle concept drifts, one has to remember that these algorithms should still fulfill requirements defined for stationary data since non-stationarity does not have to occur for all of the classes or for all of the time [8, 83]. It means that at a given time t for some classes y_s the data source generators may be stationary $p_t(\boldsymbol{x}, y_s) = p_{t+1}(\boldsymbol{x}, y_s)$, while for other classes y_n we may observe non-stationary properties leading to $p_t(\boldsymbol{x}, y_n) \neq p_{t+1}(\boldsymbol{x}, y_n)$. Obviously, classes that have been static may become dynamic and those that were changing may transition into static states. We can consider a recommendation system as a real-world example where such scenarios may very likely occur.

Users are constantly processing new information given to them from social media, the internet, or news outlets, learning about new things they have not seen before. Those new things may become interesting to a user or not – but they still need to be processed in a continuous manner, calling for class-incremental mechanisms. A new topic does not become the major or only interest for the user; thus it cannot overshadow the previously seen ones. Therefore, catastrophic forgetting must be avoided to retain not only the most current, but all topics relevant to a given user. At the same time, our preferences and tastes are not static. We change our interests within the span of years, months, or even days. A concept that was interesting to the user at a given point cannot be assumed to be interesting indefinitely. A continual learning system must thus be able of revisiting previously learned knowledge and updating it according to any shifts in preferences. This calls for concept drift adaptation approaches, as previously seen topics may evolve over time and the interest of users

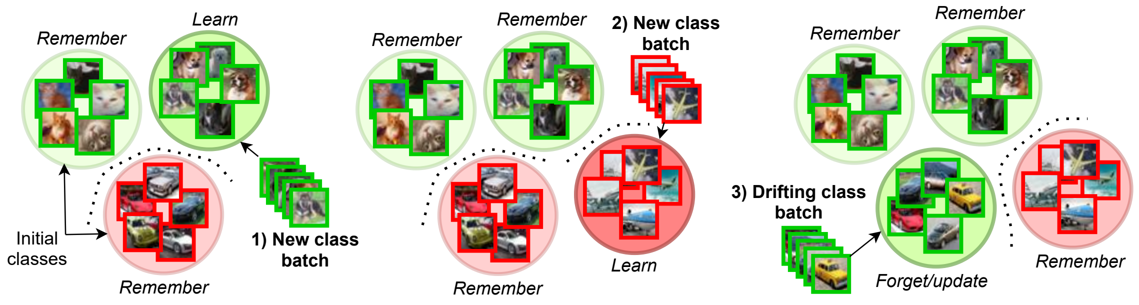


Fig. 3.: Three vital aspects of a holistic approach to continual learning: learning new classes, retaining previous knowledge and adapting to concept drifts, illustrated by the example of a binary recommendation system (like or dislike).

in them may either increase or decrease over time. Creating a true continual learning system over user preferences is a real-world and practical illustration explaining the need for holistic approaches capable of remembering new concepts and selective forgetting with adaptation to changes in the old ones (Fig. 3).

Unfortunately, if we carefully observe the current research in continual learning domain we will notice that almost all of the published works are either entirely focused on concept drift adaptation, ignoring the problem of dealing with insufficient retention, or almost exclusively targeting catastrophic forgetting, with a sole emphasis on aggregating only stationary concepts. Methods that entirely focus on the former will inevitably lead to only locally optimal solutions completely ignoring what was observed over a longer period of time [27]. While, on the other hand, blindly following the only objective of the latter may force a model to retain already invalid knowledge, leading to completely counterproductive results and severely impeding required adaptation [178]. In fact, both mechanisms are needed to create a more generic learning paradigm, robust to difficulties present in real-world problems [8].

CHAPTER 3

CLASS-INCREMENTAL GRADIENT-BASED MIXTURE OF GAUSSIANS

Continual learning models for stationary data focus on effectively learning and retaining concepts coming to them in a sequential manner [9, 12]. While the initial research done in this domain was, in large part, oriented towards task-incremental solutions, more recent works attempt to address generalized cases consisting of purely class-incremental and data-incremental (also known as domain-incremental) settings [10, 179]. These scenarios are usually more universal but also more challenging and restrictive mainly due to the lack of task or even class labels.

In the most generic class-incremental environment, we have to be ready to deal with classes coming one by one, without any higher-level grouping. This requirement invalidates many of the previously proposed methods, e.g., memory-free regularization-based ones [9], which are not capable of discriminating between older and new classes, even if they address the catastrophic forgetting problem [179, 61].

While the most standard experience replay methods can be effectively applied in the class-incremental scenarios [180, 28], there has been also a search for alternative approaches that could provide natural capabilities required for such cases. A significant group of methods can be identified based on their reliance on centroids (or prototypes) combined with the nearest-centroid classification methods [98]. Since those centroids can be independently added to the classifier, they are examples of methods that can be very smoothly incorporated into class-incremental scenarios, offering almost no interference in the latent space, as opposed to the regularization models.

In this chapter, we explore an advanced version of these alternatives by proposing the first incorporation of the gradient-based Gaussian mixture model into a class-incremental deep continual learning framework, called **MIX**. In fact, it requires us to tackle three major problems at the same time: (i) gradient-based mixture training, (ii) combining it with a trainable deep feature extractor and, finally, (iii) making it suitable for class-incremental scenarios. To achieve these goals, we introduce a set of dedicated losses, configurations and methods, providing a probabilistic classifier on top of a feature extractor and within a model capable of learning end-to-end. This opens many potential research directions that could exploit the well-modeled statistical properties of Gaussians. In addition to that, we show that our class-incremental mixture model, analogously to the centroid-driven algorithms, is characterized by some inherent properties useful in continual learning scenarios. They allow it for much better separation of concepts at the level of the classification module, leading to significant improvements in memory-free scenarios when pre-trained extractors are used. Through an extensive empirical study, we analyze different configurations of our method, provide the reader with some intuition about its parameters and show its competitiveness in the context of other state-of-the-art continual learning algorithms.

3.1 Mixture of Gaussians for class-incremental learning

The general goal of our work is to incrementally learn a classification model defined as $\phi^{(t)} : \mathcal{X} \rightarrow \mathcal{C}$ that can effectively incorporate subsequent class batches $\langle(\boldsymbol{X}^{(1)}, c = 1), (\boldsymbol{X}^{(2)}, c = 2), ..., (\boldsymbol{X}^{(t)}, c = t)\rangle$, where $\boldsymbol{X}^{(t)}$ contains instances $\boldsymbol{x}$ only for a given class c. After t classes the model $\phi^{(t)}$ should aim at minimizing the loss for the current class $c = t$ and all previously observed ones:

$$\boldsymbol{\mathcal{L}}^{(t)} = \sum_{c=1}^{t} \sum_{n=1}^{N_c} \mathcal{L}^{(c)}(\phi^{(t)}(\boldsymbol{x}_n^{(c)})), \tag{3.1}$$

where $\boldsymbol{x}_n^{(c)} \in \boldsymbol{X}^{(c)}$ and $\mathcal{L}^{(c)}$ can be any supervised loss.

Additionally, since we are interested in deep learning, we define the whole model as a tuple $\phi^{(t)} = \langle \mathcal{F}^{(t)}, \mathcal{G}^{(t)} \rangle$ consisting of a feature extractor $\mathcal{F}^{(t)}$ and a classifier $\mathcal{G}^{(t)}$ jointly aggregating knowledge from t classes. The model makes prediction by classifying the features provided from the extractor $\phi^{(t)}(\boldsymbol{x}) = \mathcal{G}^{(t)}(\mathcal{F}^{(t)}(\boldsymbol{x})) = G^{(t)}(\hat{\boldsymbol{x}})$. In this work, we aim at employing the mixture of Gaussians as a jointly trained incremental classifier. Although the model learns from dedicated features $\hat{\boldsymbol{x}}$, in the next section, we use $\boldsymbol{x}$ for the sake of simplicity of notation.

3.1.1 Generic supervised mixture model

Formally, in a standard unsupervised setting the density for a given point $\mathbf{x}$ can be expressed using a multivariate normal distribution defined as:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}_k|}} exp\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\}, \tag{3.2}$$

where μ and $\boldsymbol{\Sigma}$ are its mean and covariance, and D is the size of the input (number of dimensions). The Gaussian mixture models (GMM) have been designed to approximate more complex multivariate densities by decomposing them into K components:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \omega_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{3.3}$$

where each of them is defined using a single Gaussian defined above and ω_k are their weights. The combined model, equipped with more degrees of freedom, should be capable of providing more accurate expressions of the overall observed distributions than a simpler approach utilizing only a single component. In such a framework, the fitting of the mixture to given data $\boldsymbol{X}$ is based on minimizing the loss defined using the log-likelihood function:

$$\bar{\mathcal{L}} = -\log p(\boldsymbol{X}|\omega, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{1}{N}\sum_{n=1}^{N}\log\sum_{k=1}^{K}\omega_k\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{3.4}$$

where we adjust the free parameters of the model – means $\boldsymbol{\mu}$, covariance matrices $\boldsymbol{\Sigma}$ and weights ω. To adapt the given framework to supervised scenarios we can simply specify a separate mixture model for each class c:

$$p(\mathbf{x}|c) = \sum_{k=1}^{K}\omega_k^{(c)}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k^{(c)}, \boldsymbol{\Sigma}_k^{(c)}), \tag{3.5}$$

and focus on minimizing the aforementioned loss also per class $\bar{\mathcal{L}}^{(c)}$:

$$\hat{\mathcal{L}} = \sum_{c=1}^{C}\bar{\mathcal{L}}^{(c)} = -\log\sum_{c=1}^{C}p(\boldsymbol{X}^{(c)}|\omega^{(c)}, \boldsymbol{\mu}^{(c)}, \boldsymbol{\Sigma}^{(c)}), \tag{3.6}$$

where $\boldsymbol{X}^{(c)}$ are N_c class-specific observations.

In continual learning we should aim at minimizing the interference of current updates with previously created models to alleviate the detrimental effect of catastrophic forgetting. Therefore, it is worth mentioning here that GMMs create such an opportunity by allowing for maximizing the log-likelihood only for a currently learned class through $\bar{\mathcal{L}}^{(c)}$. It provides a perfect separation at the level of the classification model.

3.1.2 Optimization techniques

Various techniques can be applied for the task of fitting the mixture model to given data. The most standard approach utilizes the EM algorithm, which can be realized in both offline and online settings [181, 182]. While EM provides a stable framework for learning the mixtures – in terms of mathematical constraints and convergence – it is critically limited when it comes to working with high-dimensional data and feasible memory consumption [183]. On top of that, this algorithm is intrinsically incapable of being fully integrated with neural networks, preventing it from achieving joint end-to-end deep learning and benefiting from dedicated features.

An alternative approach involves gradient-based optimization [183]. This method has been proved to be able to provide more scalable and flexible algorithms capable of working in challenging scenarios with high-dimensional data and in online settings. Most importantly, the gradient-based approach naturally enables combining the model as a classifier with a trainable deep feature extractor [184], allowing for extending the optimization process with the input space adjustments. Methods utilizing such a compound learning process showed much evidence of its usability in offline and unsupervised scenarios, while at the same time encouraging researchers to develop further extensions and improvements [183, 185]. Given all of the characteristics, we decided to use this approach in our scenario of continual learning.

3.1.3 Mixture optimization for class-incremental deep learning

In order to apply gradient-based learning to GMM in class-incremental deep learning scenarios, we have to address several different issues. Some of them are common for all GMM models using gradient-based learning, while others are specific for the class-incremental deep learning settings.

In general, we say that our goal is to optimize the class-incremental joint model $\phi^{(t)} = \langle \mathcal{F}^{(t)}, \mathcal{G}^{(t)} \rangle$, defined in Sec. 3.1, using some supervised loss $\mathcal{L}$. Since we set $\mathcal{G}^{(t)} = \boldsymbol{\mathcal{N}}^{(t)}$, where $\boldsymbol{\mathcal{N}}^{(t)}$ is a whole GMM model, we have $\phi^{(t)}(\boldsymbol{x}) = \boldsymbol{\mathcal{N}}^{(t)}(\mathcal{F}^{(t)}(\boldsymbol{x}))$ and the trainable parameters are weights $\partial \mathcal{L}/\partial \boldsymbol{W}$ and biases $\partial \mathcal{L}/\partial \boldsymbol{b}$ for the extractor, and means $\partial \mathcal{L}/\partial \boldsymbol{\mu}$, covariance matrices $\partial \mathcal{L}/\partial \boldsymbol{\Sigma}$ and component weights $\partial \mathcal{L}/\partial \omega$ for the classifier. All of the subsequent paragraphs focus on designing optimization in the classifier (mixture) space, just like it was introduced in Sec. 3.1.1.

3.1.3.1 Loss design

Max-component. It has been shown that optimizing the full loss $\bar{\mathcal{L}}^{(c)}$ given in Eq. 3.4 may lead to some numerical instabilities, especially for high-dimensional data [183]. To address this issue a *max-component approximation* can be used. This approach is very straightforward. Since all $p(\boldsymbol{x}|c,k)$ in Eq. 3.5 are positive, any component provides a lower bound for the whole sum used in $\bar{\mathcal{L}}^{(c)}$. If now, for every point $\boldsymbol{x}_n$ we find a component providing the highest log-likelihood and sum all of them, we will get the largest (max-component) lower bound [183]:

$$\mathcal{L}^{(c)}_{max} = -\frac{1}{N_c}\sum_{n=1}^{N_c}\max_k \log(\omega_k^{(c)}\mathcal{N}(\mathbf{x}_n^{(c)}|\boldsymbol{\mu}_k^{(c)},\boldsymbol{\Sigma}_k^{(c)})). \tag{3.7}$$

Since we can state that:

$$\mathcal{L}^{(c)}_{max} \geq \bar{\mathcal{L}}^{(c)}, \tag{3.8}$$

we are able to minimize $\bar{\mathcal{L}}^{(c)}$ by focusing only on $\mathcal{L}^{(c)}_{max}$. It is also worth mentioning that just like the general formula given Eq. 3.6 may eliminate the interference with previously learned classes, the max-component approximation can limit the same issue at the level of class components, for example, in data-incremental scenarios [179], making this approach a natural candidate for continual learning settings.

Inter-contrastive loss. All of the introduced losses are limited to scenarios either without a feature extractor or with a fixed pre-trained one. Unfortunately, if we operate in a setting where we can modify the input space of the mixture model and we utilize any of the aforementioned metrics relying entirely on maximizing log-likelihood, we will inevitably end up with a local minimum that for a joint model $\phi^{(t)}$

exists, for example, where $\forall \boldsymbol{x}(\mathcal{G}^{(t)}(\boldsymbol{x}) = \mathbf{0})$. This issue can be solved by incorporating an inter-contrastive loss that will distance representations for different classes. We define the loss as:

$$\mathcal{L}^{(c)}_{inter} = \frac{1}{N_c} \max_{j \neq c} \sum_{n=1}^{N_c} \max_k \log(\omega_k^{(j)} \mathcal{N}(\mathbf{x}_n^{(c)} | \boldsymbol{\mu}_k^{(j)}, \boldsymbol{\Sigma}_k^{(j)})), \tag{3.9}$$

which boils down to finding the closest component in other classes, and then optimizing against the class that on average is the closest to the one currently being considered. We choose the outer maximum function instead of the mean since we observed that for the latter the contrastive part becomes too fuzzy, preventing itself from delivering what was intended. We keep the log-likelihood to ensure a similar numerical space of loss values as the one for the positive part given in Eq. 3.7. However, now one should notice that minimizing such a loss may very easily destabilize learning since optimization will gravitate towards $\bar{\mathcal{L}}^{(c)}_{inter} \rightarrow -\infty$ preventing the model from actually fitting to the class examples. To avoid it we introduce a *tightness bound* τ that clips the contrastive loss value at some pre-defined point:

$$\mathcal{L}^{(c)}_{inter}(\tau) = \max(\tau, \mathcal{L}^{(c)}_{inter}), \tag{3.10}$$

which basically means that we stop the decrease of the contrastive loss below the given bound, allowing for a more significant contribution of the actual fitting part $\mathcal{L}^{(c)}_{max}$. We parametrize the τ value with a simple linear transformation:

$$\tau = \bar{p}^{(c)}_{max} - \frac{1}{\tau_p}, \tag{3.11}$$

where $\bar{p}^{(c)}_{max}$ is the average maximum density value observed across all class components (can be obtained on-the-fly) and τ_p is a tunable hyperparameter that takes values between $\langle 0, 1\rangle$. Such a loss can provide effective discrimination between components of different classes, as shown for an example in Fig. 4.

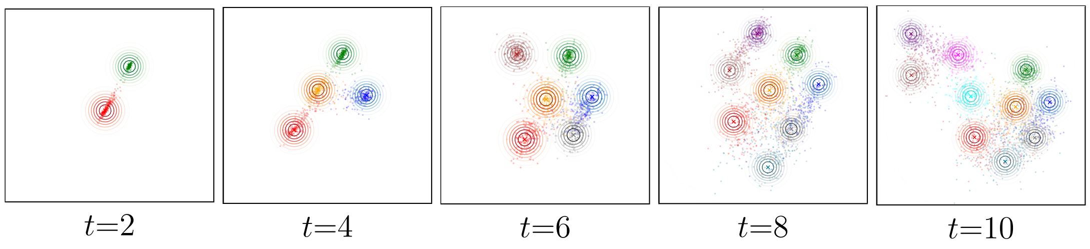


Fig. 4.: Learning subsequent classes of FASHION incrementally (K=1) with the inter-contrastive loss utilizing the tightness bound ($\tau_{p,inter}$=0.2).

Diverse components. While all of the introduced techniques and modifications ensure reliable discrimination between components of different classes, they do not consider differentiation between components of the same class or their quality. In fact, even in offline gradient-driven settings without dynamic feature extraction it is common to obtain mixtures reduced to a single component per class with all the others practically meaningless, e.g., due to zeroed weights [183]. In scenarios with a trainable extractor, this problem becomes even more significant as it is very easy for the optimizer to focus on maximizing log-likelihood from a single component when not only the mixture part allows for it but also the flexible extractor. While in standard scenarios this problem can be successfully addressed with a good initialization method, e.g., using k-means [186], we observed that it was not enough in our case. As a consequence, we introduced two elements to the learning process.

- **Regionalization** – before learning each class, we first divide it into K clusters using the k-means clustering. Then we force each component to fit only to the data from its cluster called a *region* $\mathcal{R}_k^{(c)}$. This replaces the max-component loss $\mathcal{L}_{max}^{(c)}$ defined in Eq. 3.7 with:

$$\mathcal{L}_{reg}^{(c)} = -\sum_{k=1}^{K} \frac{1}{N_k} \sum_{\boldsymbol{x} \in \mathcal{R}_k^{(c)}} \log(\omega_k^{(c)} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k^{(c)}, \boldsymbol{\Sigma}_k^{(c)})). \tag{3.12}$$

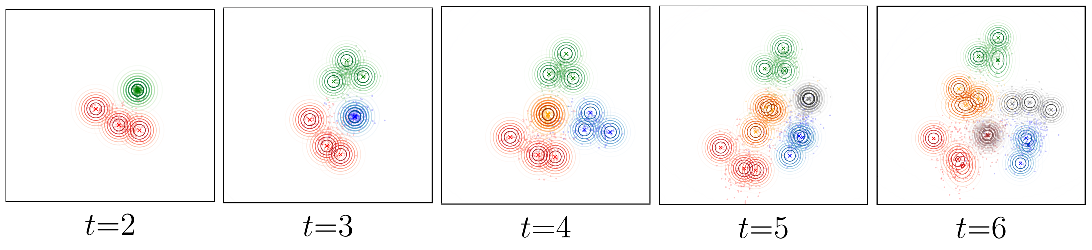


Fig. 5.: Learning subsequent classes of FASHION incrementally (K=3) with regionalization and the intra-contrastive loss utilizing the tightness bound ($\tau_{p,intra}$=0.25).

- **Intra-contrastive loss** – the regionalization approach is necessary yet not sufficient to provide sufficient diversification between same-class components. The reason for it is the same as for discrimination between different classes, as described in the previous paragraph. Analogously to the inter-contrastive loss, we add the intra-contrastive loss with the tightness bound τ:

$$\mathcal{L}_{intra}^{(c)}(\tau) = \sum_{k=1}^{K} \max(\tau, \max_{m \neq k} \frac{1}{N_k} \sum_{\boldsymbol{x} \in \mathcal{R}_k} \log(\omega_m^{(c)} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_m^{(c)}, \boldsymbol{\Sigma}_m^{(c)})). \tag{3.13}$$

  which for each class region pushes away other same-class components that on average are closest to the currently being considered one, based on the regionalization conducted in the previous step. We choose the outer maximum function for the same reason as for inter-contrastive loss. Obviously, one can define separate τ for the inter- and intra-contrastive loss.

Such an approach can effectively increase the diversity of the same-class components, as given for an example in Fig. 5. However, it also has to be mentioned that the effectiveness of this algorithm is heavily dependent on the quality of the initial clustering, which may be imperfect, especially when we just start learning a new class. In addition to that, this approach imposes a hard constraint on how the representation and mixture may look, which limits the flexibility of the whole model. Regardless of these concerns, this method can still effectively improve the overall performance of a multi-component model over a method without the proposed improvement, as we will show in our extensive experiments.

Final component-based losses. To summarize, we distinguish two component-based losses. One uses the max-component approach:

$$\mathcal{L}_{mc} = \sum_{c=1}^{t} \mathcal{L}_{max}^{(c)} + \mathcal{L}_{inter}^{(c)}(\tau_{inter}), \tag{3.14}$$

while the second loss adds the regionalization technique with the intra-contrastive part:

$$\mathcal{L}_{mcr} = \sum_{c=1}^{t} \mathcal{L}_{reg}^{(c)} + \beta(\mathcal{L}_{inter}^{(c)}(\tau_{inter}) + \mathcal{L}_{intra}^{(c)}(\tau_{intra})), \tag{3.15}$$

where β=0.5 by default.

Cross-entropy loss. Last but not least, we can also attempt to directly optimize the whole standard loss $\hat{\mathcal{L}}$ given in Eq. 3.4, using a high-level supervised wrapper loss, e.g., cross-entropy. In such a case, our loss is defined as:

$$\mathcal{L}_{ce} = -\sum_{c=1}^{t} \sum_{n=1}^{N_c} \boldsymbol{y}_n^{(c)} \log \hat{y}_n^{(c)}, \tag{3.16}$$

where $\boldsymbol{y}$ is a one-hot target vector and $\hat{y}_n^{(c)}$ comes from the softmax function:

$$\hat{y}_n^{(c)} = \frac{e^{p_n^{(c)}}}{\sum_{c=1}^{t} e^{p_n^{(c)}}}, \tag{3.17}$$

and $p_n^{(c)} = p(\boldsymbol{x}_n|c)$ is a density value for a given class produced by the mixture model accordingly to Eq. 3.5.

3.1.3.2 Constraints

Other issues that have to be addressed when using gradient-based mixture training are the mathematical constraints that have to be enforced to preserve a valid mixture model. This is required since gradient-based learning does not constrain the possible values for means, covariance matrices and weights, and the last two have to remain in a specific range of values.

Component weights. For the GMM model its component weights ω_k have to sum up to one: $\sum_{k=1}^{K} \omega_k = 1$. To ensure that the effective weights satisfy this requirement we simply train auxiliary free parameters $\hat{\omega}_k$ and use the softmax-based normalization to obtain required values [187, 183]:

$$\omega_k = \frac{e^{\hat{\omega}_k}}{\sum_{j=1}^{K} e^{\hat{\omega_j}}}. \tag{3.18}$$

Covariance matrices. For a general case, the covariance matrices of the GMM model should be symmetric positive definite $\boldsymbol{v}^T \Sigma \boldsymbol{v} > 0$ for all nonzero vectors $\boldsymbol{v}$. This can be enforced using the Cholesky decomposition [188]:

$$\boldsymbol{\Sigma} = \boldsymbol{A}\boldsymbol{A}^T, \tag{3.19}$$

where $\boldsymbol{A}$ is a triangular matrix with positive diagonal values: $a_{ii} > 0$ and, at the same time, our trainable proxy parameter. To enforce positive diagonal values, after each gradient-based update we clamp them with $a_{ii} = min(a_{ii}, d_{min})$ using some predefined d_{min} value. Finally, we also consider a case of a mixture using only the diagonal of the covariance – variance $\boldsymbol{\sigma}$, which we control using the same clamp-based approach: $\sigma_i = min(\sigma_i, d_{min})$.

3.1.4 Memory buffer

In our work, we consider the class-incremental scenario with strictly limited access to previously seen observations (classes). Therefore, in all of the introduced losses we use all available data for the currently learned class t, while for the others we sample from the memory buffers $\mathcal{M}_c$ that store an equal number of examples per each previously seen class. For the method with regionalization, these instances are equally distributed between different components.

As shortly mentioned in Sec. 3.1.1, the standard GMM model (without the inter-contrastive loss), due to the nature of its loss that is defined per each class, has the capability of learning a new class without interfering with others. In fact, if the feature extractor was pre-trained and static we could remove the inter-contrastive loss and even get rid of the memory buffer, allowing for memory-free training, as we will show in the experimental study. The memory buffer is needed in a general case when we assume the joint training of the whole model.

3.1.5 Classification

Finally, in the presented model the classification of an instance $\boldsymbol{x}_n$ can be performed using two approaches, either the softmax function:

$$\hat{y}_n^{(c)} = \frac{e^{p_n^{(c)}}}{\sum_{c=1}^{t} e^{p_n^{(c)}}}, \tag{3.20}$$

where $p_n^{(c)} = p(\boldsymbol{x}_n|c)$, or by taking the weighted support of the closest component:

$$\hat{y}_n^{(c)} = \max_k \omega_k^{(c)} \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k^{(c)}, \boldsymbol{\Sigma}_k^{(c)}). \tag{3.21}$$

We will empirically show that these methods work best with specific loss functions designed in the previous sections.

3.2 Experimental study

In our experiments, we want to empirically explore all of the introduced methods and parameters to provide the reader with guidance and intuition about the effective configurations of our algorithm. In addition to that, we put our method in the performance context of different state-of-the-art baselines that can be used in class-incremental settings. We show how our model performs in full end-to-end scenarios, as well as with a pre-trained extractor, compared with other solutions.

3.2.1 Data

For the purpose of the evaluation we used commonly used visual datasets that were turned into class-incremental sequences by presenting their classes subsequently to the models [10, 61]. We used: MNIST, FASHION, SVHN, CIFAR10 and IMAGENET10 – a subset of the tiny IMAGENET200, to gain deeper insights into our method while conducting experiments with hundreds of different configurations. Then, we extended this set with CIFAR20 – the coarse-grained version of CIFAR100, IMAGENET20A and IMAGENET20B – larger subsets of IMAGENET200 – to benchmark our method against other algorithms. Finally, for the experiments involving fixed extractors, we used pre-trained features to construct four additional sequences – CIFAR100-PRE10, CIFAR100-PRE100, IMAGENET200-PRE20 and IMAGENET200-PRE200, which consisted of features extracted for CIAFR100 and IMAGENET200, using extractors trained on 10, 20, 100 and 200 classes of the original datasets. The summary of the used benchmarks is given in Tab. 1. Details of the feature extractors can be found in the next section.

3.2.2 Model configurations

In the first section of our experiments, we explored different configurations of our algorithm, which can be mostly seen as an ablation study. Firstly, we evaluated different **losses** defined in Sec. 3.1.3.1 (CE, MC and MCR) combined with different **classification methods** given in Sec. 3.1.5 (softmax, max-component). Secondly, we checked different settings for the **tightness bound** parameter τ_p by evaluating a grid of values for inter-tightness and intra-tightness – we considered $\tau_p \in \langle$1e-06, 1e-05,

Table 1.: Summary of used datasets.

Dataset	Train	Test	Shape	Cls	Feats
MNIST	50 000	10 000	1x28x28	10	No
FASHION	60 000	10 000	1x28x28	10	No
SVHN	73 257	26 032	3x32x32	10	No
IMAGENET10	5000	500	3x64x64	10	No
CIFAR10	50 000	10 000	3x32x32	10	No
IMAGENET20A	10 000	1000	3x64x64	20	No
IMAGENET20B	10 000	1000	3x64x64	20	No
CIFAR20	50 000	10 000	3x32x32	20	No
CIFAR100-PRE10	50 000	10 000	3x32x32	100	128
CIFAR100-PRE100	50 000	10 000	3x32x32	100	512
IMAGENET200-PRE20	100 000	10 000	3x64x64	200	256
IMAGENET200-PRE200	100 000	10 000	3x64x64	200	256

0.0001, 0.001, 0.01$\rangle$ for both. Thirdly, we analyzed how assuming different **numbers of components** affects the classification performance on different datasets. We used $K \in \langle 1, 3, 5, 10, 20\rangle$. Then we checked if it is better to maintain a whole **covariance matrix** or only its variance (FULL, VAR). Finally, we evaluated different **learning rates** for the extractor and GMM part, using $\alpha_{\mathcal{F}} \in \langle$1e-07, 1e-06, 1e-05, 0.0001, 0.001$\rangle$ and $\alpha_{\mathcal{G}} \in \langle$1e-05, 0.0001, 0.001, 0.01, 0.1$\rangle$, to check whether it may be beneficial to configure them separately, and different **memory sizes** $\mathcal{M}_c \in \langle 8, 64, 128, 256, 512\rangle$ to analyze how our method exploits limited access to class examples.

While evaluating specific parameters we kept others fixed. For our base configuration we chose a setup that was capable of providing performance comparable with a standard experience replay. We used the MCR with max-component as our loss and classification method, $K = 3$, $\tau_{p,inter} = 0.002$, $\tau_{p,intra} = 0.01$, $\alpha_{\mathcal{F}} =0.0001$, $\alpha_{\mathcal{G}} =0.001$ and $d_{min} = 0.001$ with only variance stored per each component. We assumed a modest memory buffer per class $\mathcal{M}_c = 256$ and matched the size of a memory sample per class with the training batch size. The model was trained for 10 (MNIST, FASHION) or 20 epochs per class, with 32 (IMAGENET) or 64 instances in a mini-batch.

3.2.3 Algorithms

In the final section of this work, we compared our class-incremental Gaussian mixture model (**MIX**) with other classifiers dedicated for continual learning scenarios. We considered: standard experience replay (**ER**) [189], experience replay with subspaces (**ERSB**) [178], centroid-based **iCARL** [29], two gradient-based sample selection methods (**GSS** and **AGEM**) [34, 33], experience replay combined with knowledge distillation and regularization (**DER**) [180], and two purely regularization-based approaches – **LWF** [46] and **SI** [50]. For the last two we used their modifications adjusted for single-task learning [61]. As our lower bound we used a naively learning net (**NAIVE**), and for the upper bound we present results for the offline model (**OFFLINE**) either trained by us (IMAGENET20A, IMAGENET20B and fine-tuned models for IMAGENET200), or by referring to other publications [190, 178].

Based on the observations made in the first section of the experiments, in the final evaluation we used two variants of our algorithm: **MIX-CE** and **MIX-MCR** with $\tau_{p,inter}$ =0.0001, $\tau_{p,intra}$ =0.001, $\alpha_{\mathcal{F}}$ =0.0001, $\alpha_{\mathcal{G}}$ =1e-05 and, once again, $d_{min} = 0.001$ with only variance maintained per each component. The only parameter that we tuned per each dataset was the number of components K. We used Adam as the optimizer. For the memory-free scenarios with pre-trained extractors, we turned off the inter-contrastive loss to minimize interference with previously learned classes as mentioned in Sec. 3.1.4.

The main parameters of the baselines methods were set based on the original papers and other literature, including empirical surveys or works containing vast empirical studies [61, 179, 10, 180, 178, 33]. For all memory sampling methods we matched the memory sampling size with the training batch size. For ERSB we used 10 centroids per class each containing up to either 25 or 15 instances to match the total memory size. DER used α_d=0.5, for LWF we set the softmax temperature $T = 2$ and progressively increased its distillation coefficient as suggested in [61], and SI used λ =0.0001. All of the methods utilized the Adam optimizer with a learning rate α=0.0001 as we did not observe any significant differences when changing this parameter.

Analogously to the configuration section, all of the algorithms, including ours, were trained for 10 (MNIST, FASHION) or 20 epochs per class, using 32 (IMAGENET) or 64 instances per mini-batch. The offline models were trained for either 50 or 100 epochs, until they achieved a saturation level. The memory buffer was set to $\mathcal{M}_c = 128$ (IMAGENET) or $\mathcal{M}_c = 256$ for methods supporting memory per class (ER, ERSB, iCARL), and $\mathcal{M} = C \cdot 128$ or $\mathcal{M} = C \cdot 256$ for the remaining ones (GSS, AGEM, DER), where C was the total number of classes. The latter group was equipped with reservoir buffers [180]. For the experiments with pre-trained extractors we wanted to check the **memory-free scenario**, therefore we set $\mathcal{M}_c = 0$ for our methods and $\mathcal{M}_c = 1$ or $\mathcal{M} = C$ for others, since most of them could not be run without storing any examples.

All of the algorithms, including different configurations of our method described in the previous section, were combined with feature extractors. For MNIST and FASHION we used a simple CNN with two convolutional layers consisting of 32 (5x5) and 64 (3x3) filters, interleaved with ReLU, batch normalization and max pooling (2x2). For SVHN and IMAGENET we utilized ResNet18, its modified version for CIFAR10 and CIFAR20, and ResNeXt29 for CIFAR100 [191]. The classification layers consisted of the default configurations. Finally, for our method, ER, ERSB, AGEM and DER we disabled batch normalization, since, consistently with [192, 193], we observed a significant difference in performance when those layers were turned off for the given methods. As mentioned in Sec. 3.2.1, for the memory-free scenarios, the extractors were pre-trained on either 10, 20, 100 or 200 classes of CIFAR100 and IMAGENET200.

3.2.4 Evaluation

We evaluated the presented methods in a class-incremental setting, where all of the classes were presented to the models subsequently and were not shown again after their initial appearance. Since our current algorithm do not support revisiting, we do not consider this scenario. We measured the accuracy of a given algorithm after each class batch, utilizing holdout testing sets, and then, based on [12], used it to calculate the average incremental accuracy over the whole sequence:

$$\Omega_{all} = \frac{1}{T}\sum_{t=1}^{T}\alpha_t, \tag{3.22}$$

where α_t is the model performance after t classes and $T = C$ is the total number of classes. In addition to the whole aggregation, for the final comparison, we provided these values after each batch to present a more complete perspective of the obtained results.

3.2.5 Results

In this section, we present and describe all of the results that were obtained for the experiments introduced in the previous paragraphs. The first part consists of the analysis of different configurations of MIX, while the second one focuses on a comparison with other class-incremental algorithms.

3.2.5.1 Configurations

Loss and classification. First, we analyze different combinations of the proposed losses and classification methods. Based on Fig. 6, we can make three major observations. Firstly, the softmax classification works significantly better with the CE loss and max-component can be more efficiently paired with MC and MCR than softmax. It was evident for almost all cases (except for MC on CIFAR10) and resulted in almost 0.15 difference on average between softmax and max-component for CE, and about 0.05 for MC and MCR.

Secondly, the MCR loss performed better than MC, showing consistent improvements, especially for more complex datasets like SVHN, CIFAR10 or IMAGENET10, which resulted in more than 0.1 for a difference on average. This demonstrate that the regionalization and intra-contrastive loss are capable of providing meaningful improvements over simpler MC loss utilizing only max-component and inter-contrastive

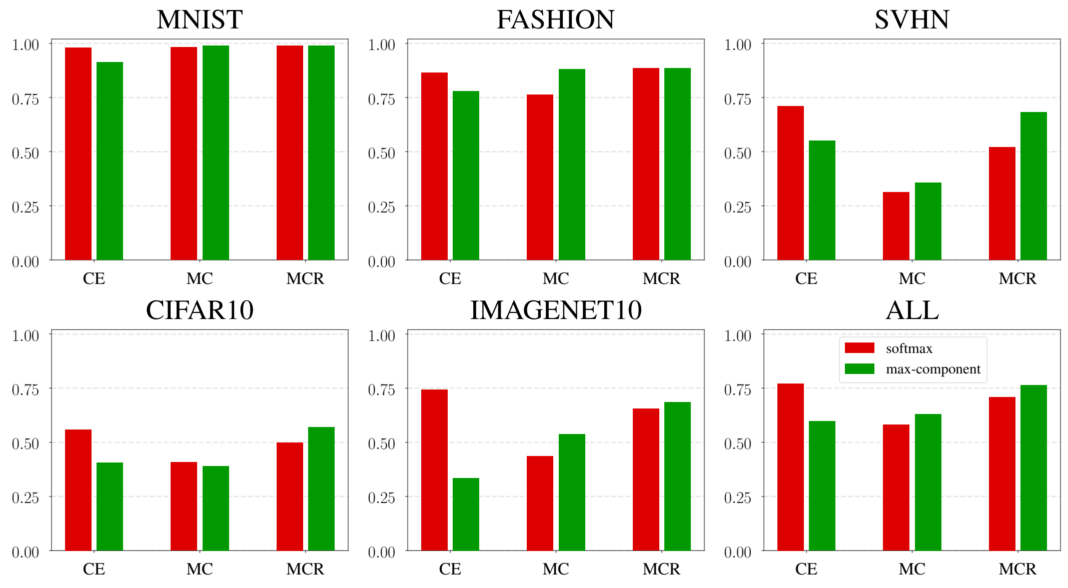


Fig. 6.: Average incremental accuracy for different losses combined with different classification methods.

elements and that ensuring higher diversity among class components can be beneficial to the model. Finally, we can see that overall CE with softmax could provide very similar results as MCR with max-component, which means that the general GMM learning formula, wrapped with a high-level supervised loss, can be sometimes as useful as more complex MCR without the need for tuning additional parameters.

One drawback of using CE, however, is the fact that it does not model the Gaussian mixtures well. As we can see in Fig. 7, the CE loss does not really have to fit the mixtures to the data since it is enough for it to ensure high classification quality. Compared with MC for K=1 or MCR for both K (Fig. 4 and 5), while it still provides similar discriminative performance, it does not produce a high-quality Gaussian model. It may be prohibitive if one wants to obtain a reliable description of the latent space. Last but not least, the model produced for MC with K=3 clearly shows that it is incapable of effectively utilizing multiple components for the same class. Please notice that only the Gaussians in the middle actually cover some data points, while the remaining components are completely unrelated to the observed data. These are examples of the degenerate solutions that we mentioned in Sec. 3.1.3.1. While for FASHION this loss could still, analogously to CE, provide similar performance as MCR (the components in the middle are fitted to the data and they are sufficient to model it), the observed desynchronization of components results in

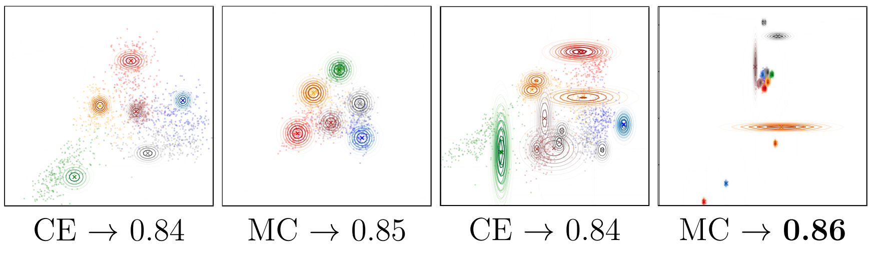


Fig. 7.: Mixtures learned with cross-entropy and simple max-component strategy (K=1 and K=3) after 6 classes of FASHION.

its weaknesses for more complex data. The MCR loss achieves both objectives at the same time: high classification accuracy and high quality of the mixture models for features. This may be important if someone requires interpretable models or would like to extend the proposed algorithm with some Gaussian-oriented techniques that MCR may enable.

Tightness. In Fig. 8, we presented a grid of values for the average incremental accuracy per each pair of inter- and intra-tightness for every dataset. One can clearly see that imposing the constraint (tightness) on the inter- and intra-contrastive loss values is beneficial to the learning process. Most of the benchmarks required $\tau_{p,inter}$ at the level of 0.0001 or 0.001 and slightly higher intra-tightness $\tau_{p,intra}$ around 0.001 or 0.01 to achieve the best results. At the same time, one should notice that imposing too

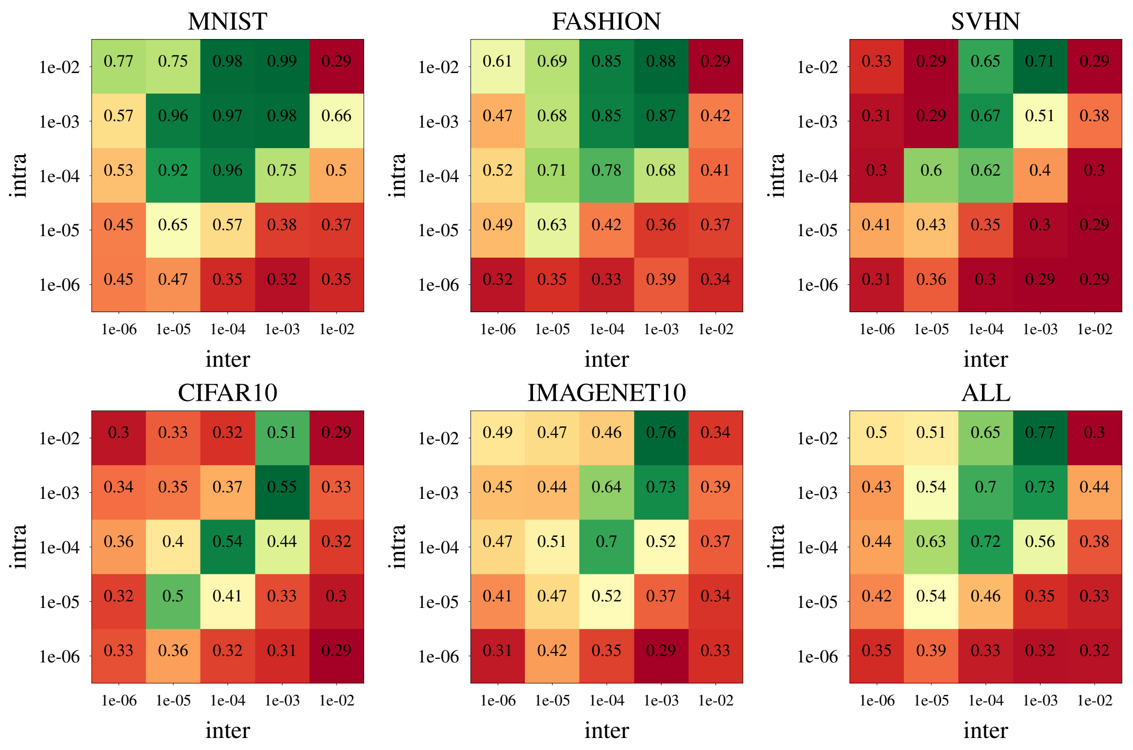


Fig. 8.: Average incremental accuracy for different combinations of the tightness parameter values (inter and intra).

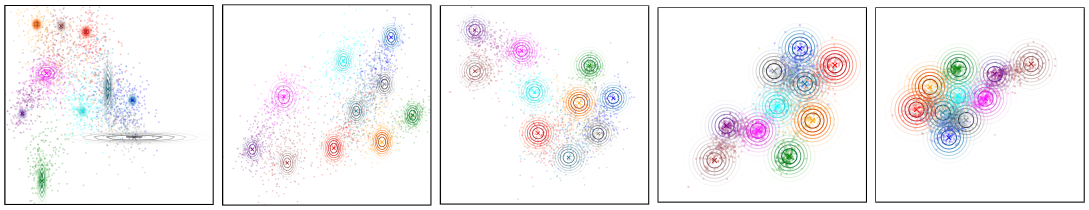


Fig. 9.: Visualization of the inter-tightness effect on learned representations (K=1) after 10 classes of FASHION.

high inter-tightness (0.01) leads to abrupt deterioration of quality, which is a result of blocking the contrastive part of the loss from pushing components of different classes from each other. The influence of setting too high intra-tightness is less important since we may simply end up with a single component that can still be effectively used for classification (Fig. 7).

The examples for FASHION, given in Fig. 9 and 10, show how increasing the inter-tightness (the first one) and intra-tightness (the second one) affects learned representations and mixture models. We can clearly see the positive impact of the constraint and the potential for sweet spots providing a good balance between differentiating components between each other and fitting them to the actual data. It is evident that too low values will introduce critical instabilities to the learning process (very high contrastive loss values overwhelming the fitting part), while too

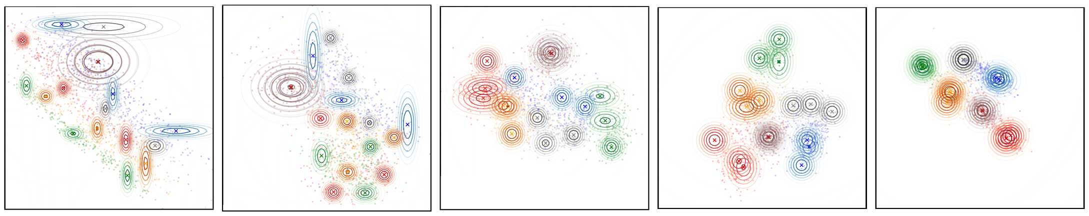


Fig. 10.: Visualization of the intra-tightness effect on learned representations (K=3) after 6 classes of FASHION.

high thresholds lead either to the decline of discriminative properties of the model or degenerate solutions.

Number of components. Tab. 2 presents how many components were required to obtain the best solutions per each dataset for the given settings. We can observe that for simpler datasets (MNIST, FASHION) using a single component per class for sufficient and that introducing additional ones led to slightly worse performance, most likely due to the fact of fitting to simple concepts and overcomplicating the optimization problem. On the other hand, more complex benchmarks (SVHN, CIFAR10, IMAGENET10) preferred access to more components per class, which could provide significant improvements, e.g., for SVHN the difference between K=1 and K=10 was almost 0.3. While for these experiments we set the learning rate slightly higher for the GMM model (0.001) than for the extractor (0.0001), we observed that when the former used rate lower than the latter (as suggested by the results for learning rates that will be presented below), the optimal K tended to be lower on average. It is possible that if GMM is dominant it prefers having more flexibility (components), while when the extractor has a higher learning rate it may be more effective in adjusting representations to lower numbers of components.

Table 2.: Average incremental accuracy for MIX using different numbers of components K.

Config	MNIST	FASHION	SVHN	CIFAR10	IMGNET10	ALL
K=1	**0.9885**	**0.8859**	0.4862	0.4282	0.6466	0.6871
K=3	0.9875	0.8782	0.5978	0.5407	0.6584	0.7325
K=5	0.9463	0.8562	0.6994	0.5522	**0.6604**	0.7429
K=10	0.9393	0.8577	**0.7438**	**0.5620**	0.6252	**0.7456**
K=20	0.9521	0.8517	0.6868	0.5532	0.4270	0.6942

Table 3.: Average incremental accuracy for MIX with diagonal and full covariance.

Config	MNIST	FASHION	SVHN	CIFAR10	IMGNET10	ALL
FULL	0.7304	0.6577	0.2931	0.3298	0.3255	0.4673
VAR	**0.9888**	**0.8849**	**0.6393**	**0.5777**	**0.6865**	**0.7555**

Covariance. Results presented in Tab. 3, unequivocally show that our gradient-based MIX can much better adapt to data if it maintains only the variance of the covariance matrix (better by almost 0.3 when compared with full covariance). It is not surprising since previous publications related to the gradient-based GMMs for offline settings suggested a similar thing [183]. Most likely, working with a full covariance matrix leads to less stable loss values, and many more free parameters (especially if the feature space is high-dimensional) likely cause problems with convergence.

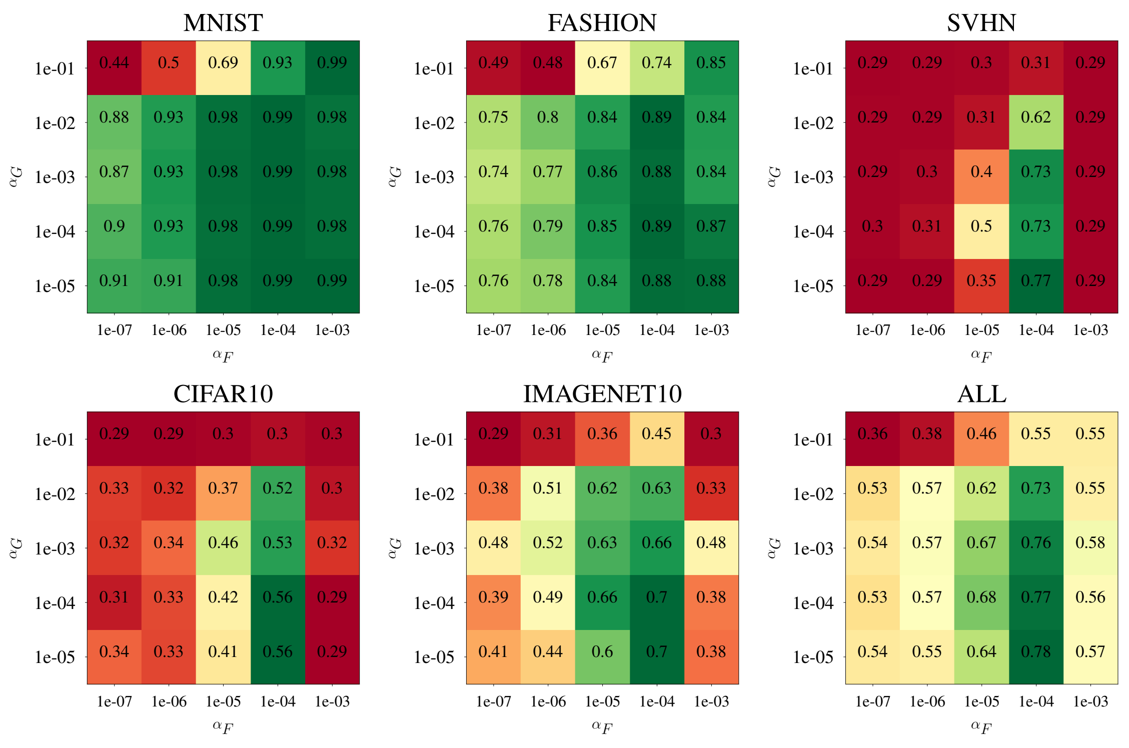


Fig. 11.: Average incremental accuracy for different learning rates.

Learning rates. Analogously to the experiments for tightness, in Fig. 11 we presented the grid evaluation results for different extractor (horizontal) and mixture (vertical) learning rates. The obtained results suggest that the former part is more important – once the optimal rate is set (0.0001 for the given settings) tuning the latter seems less significant, although overall it should be set to a similar or slightly lower value.

Memory size. Finally, if we look at the results of class-incremental learning using different memory sizes, given in Fig. 12, we will see that MIX can effectively utilize larger buffers and that it seems to be quite memory-dependent, especially for SVHN where the difference between subsequent sizes ranged from 0.1 to 0.2. Still, the gap was much smaller for all of the remaining datasets. While this characteristic of the algorithm may be problematic (the fewer examples we need, the better), it is still valid that if we can use a pre-trained extractor, the whole model does not need to

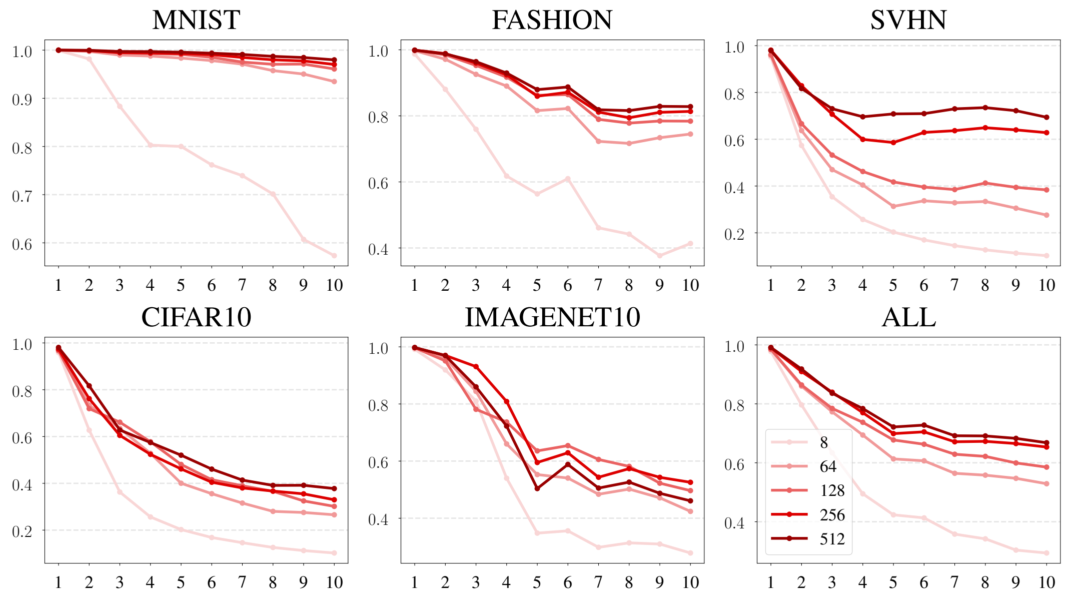


Fig. 12.: Incremental accuracy after each class batch for different sizes of the replay buffer.

use the memory buffer at all, as discussed in Sec. 3.1.4, and as it will be shown in the next section.

3.2.5.2 Baseline comparison

In the second section of our experimental study, we place our algorithm in the class-incremental performance context by comparing it with the introduced baselines. Tab. 4 and 5 present the average incremental accuracy for all of the considered algorithms. First of all, we can see that the MIX-MCR variant performed better than the MIX-CE for most of the datasets, while being very close to it for the longer sequences (difference between less than 0.01 and 0.03). This proves that MIX-MCR is capable of providing not only a better representation (mixture) model but also is more reliable from the accuracy perspective. This also means that it is worth trying to maximize the quality of the produced Gaussian models as an alternative to high-level

Table 4.: Average incremental accuracy for MIX and different baselines using end-to-end learning.

Algorithm	MNIST	FASHION	SVHN	CIFAR10	IMG10	CIFAR20	IMG20A	IMG20B	ALL
OFFLINE	1.0	0.9865	1.0	1.0	1.0	0.7805	0.7180	0.7929	0.9097
NAIVE	0.2928	0.2929	0.2929	0.2929	0.2929	0.1799	0.1799	0.1799	0.2505
ER	**0.9898**	**0.9022**	0.7271	0.5210	0.7685	0.3886	**0.4533**	**0.4622**	0.6516
ERSB	0.9887	0.8791	0.7677	0.4903	0.6787	0.3463	0.3444	0.3158	0.6014
iCARL	0.9616	0.8711	0.8298	0.5724	**0.8143**	0.4524	0.4324	0.4243	**0.6698**
GSS	0.9748	0.8385	**0.8406**	**0.6208**	0.7292	0.3891	0.3198	0.3354	0.6310
DER	0.9899	0.8782	0.8375	0.6042	0.5133	**0.4547**	0.3255	0.3758	0.6224
AGEM	0.6696	0.5782	0.2929	0.2929	0.2929	0.1799	0.1799	0.1799	0.3333
LWF	0.3508	0.3150	0.2929	0.2929	0.2929	0.1799	0.1799	0.1799	0.2605
SI	0.3338	0.3081	0.2929	0.2929	0.2929	0.1799	0.1799	0.1799	0.2575
MIX-CE	0.9804	0.8617	0.6927	0.4841	0.6878	0.3432	0.3942	0.3795	0.6030
MIX-MCR	0.9856	0.8833	0.7249	0.5304	0.7876	0.3114	0.3802	0.3784	0.6227

cross-entropy for classification.

Secondly, although our model cannot be distinguished as the best classifier (significantly worse than iCARL on average, with a difference equal to about 0.04), it is, at the same time, reliably competitive when compared with the remaining baselines (ER, GSS, DER) with a difference about 0.01 and less than 0.03. Also, it does not fall into the same pitfalls as either the weakest replay method (AGEM) or the regularization-based ones (LWF, SI), outperforming them by almost 0.4 for accuracy on average. In Fig. 13 and 14 we can see that MIX could be found among the best models for MNIST, FASHION, IMAGENET10, IMAGENET20A and IMAGENET20B, especially at the end of the datasets, providing relatively reliable performance throughout the whole sequences. On the other hand, it struggled with catching up with the best replay methods for SVHN and CIFAR-based datasets, showing that there is still a

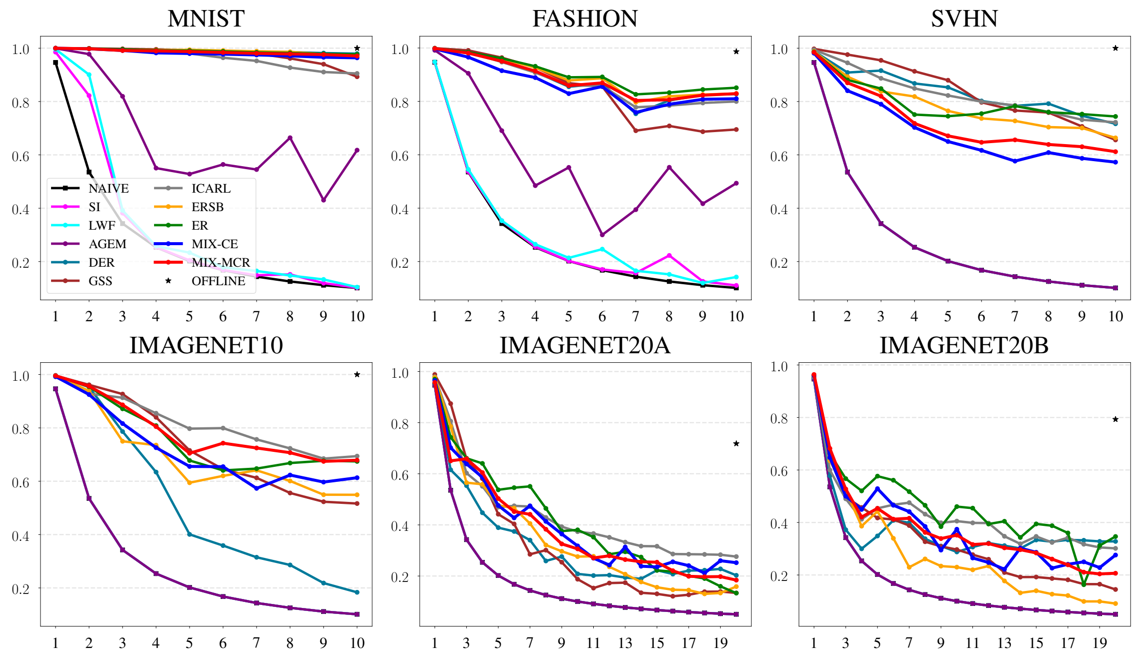


Fig. 13.: Incremental accuracy after each class batch for our methods and different baselines (1/2).

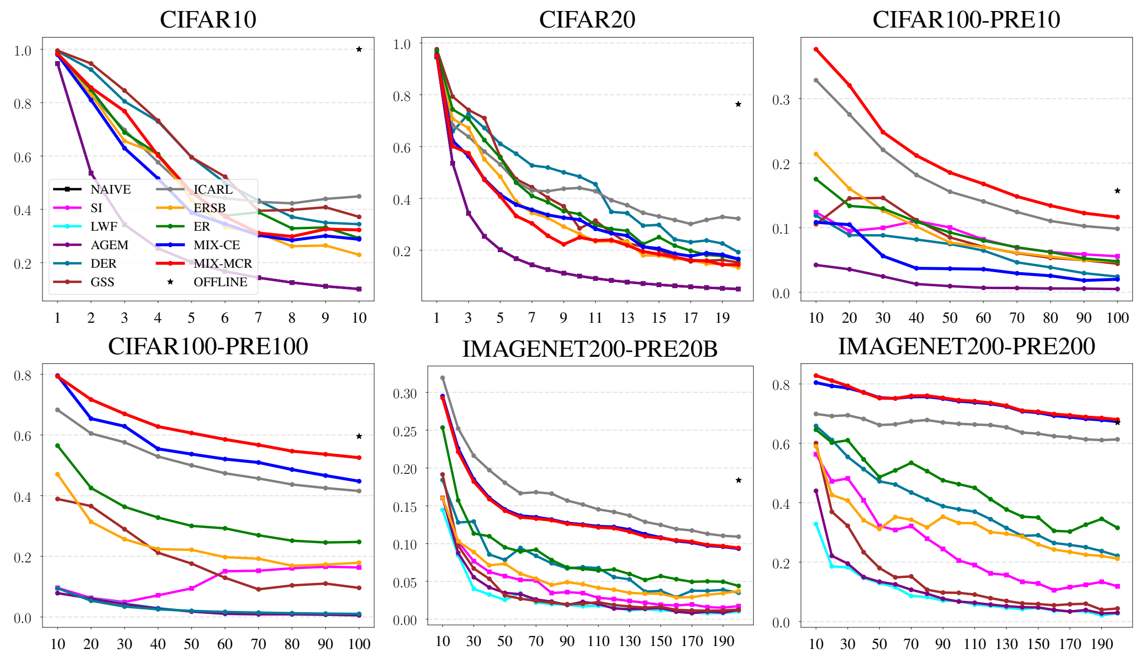


Fig. 14.: Incremental accuracy after each class batch for our methods and different baselines (2/2).

potential for important improvements when it comes to predictive accuracy.

The overall very poor performance of LWF and SI (but also AGEM), which were not much better than the NAIVE approach, confirms the observations made in other publications that the regularization-based methods cannot handle the most challenging 1-class-incremental scenarios without memory buffers [179] even after improvements proposed in [61].

We can also see that the for the scenarios with end-to-end training the models were much closer (0.01-0.3) to the OFFLINE upper bound for the shorter sequences (MNIST, FASHION, SVHN and IMAGENET10, except for CIFAR10) than for the longer ones (IMAGENET20A, IMAGENET20B, CIFAR20) with differences between 0.4-0.5, which shows that all of the state-of-the-art methods still struggle with bridging the gap between incremental learning and offline optimum.

Table 5.: Average incremental accuracy for MIX and different baselines using pre-trained extractors.

Algorithm	CIFAR100 (PRE-10)	CIFAR100 (PRE-100)	IMG200 (PRE-20)	IMG200 (PRE-200)	ALL
NAIVE	0.1042	0.1969	0.0892	0.2836	0.1685
ER	0.1130	0.3898	0.1271	0.4713	0.2753
ERSB	0.1630	0.3087	0.0998	0.3505	0.2305
iCARL	0.1738	0.5100	0.1611	0.6556	0.3751
GSS	0.0872	0.1962	0.0352	0.1860	0.1261
DER	0.0859	0.1175	0.1143	0.4106	0.1821
AGEM	-	-	-	-	-
LWF	0.0154	0.0269	0.0315	0.1024	0.0441
SI	0.0154	0.0273	0.0278	0.0899	0.0401
MIX-CE	0.1338	0.5999	**0.1782**	0.7470	0.4147
MIX-MCR	**0.2756**	**0.6522**	0.1771	**0.7520**	**0.4642**

Finally, the results for the memory-free scenarios with pre-trained models, given in Tab. 5 and Fig. 14, exhibit the main strength of the MIX algorithm. Since in these scenarios, it does not use the inter-contrastive loss, it can perfectly separate the incremental learning process for each class, preventing catastrophic forgetting at the level of the classifier. As a result, it does not have to rehearse the previous concepts at all ($\mathcal{M}_c$=0) while still being able to conduct very effective learning producing results very close to the OFFLINE upper bounds (difference between about 0 and 0.1), regardless of the quality of the extractor (pre-trained on 10 and 20 or 100 and 200 classes). The MIX-MCR method outperforms all of the baselines for all cases except for IMAGENET200-PRE20, for which only iCARL was able to provide slightly higher accuracy, even though they had a slight advantage of having approximately one example per class in the buffer. It is not a coincidence that practically only iCARL is close to our method on average (worse by about 0.1), since it uses a somehow similar paradigm in the classification layer by storing prototypes/centroids that are used for classification. And even a single instance may be enough. All of the remaining algorithms cannot handle the memory-free scenario effectively, producing solutions worse by at least 0.2 on average. For AGEM we were not able to obtain any results, given the assumed scenario. This can be a crucial property when one has to consider, for example, data privacy issues or mobile and edge computing.

3.2.6 Lessons learned

Based on the theoretical and empirical analysis presented for this work we can conclude the following.

Class-incremental learner. Regardless of many combined challenges, it is possible to successfully hybridize the gradient-based mixture models on top of convolutional feature extractors, and use them in class-incremental end-to-end continual learning scenarios. The presented results show that MIX is capable of providing competitive results when compared with well-known incremental baselines.

Dedicated losses. It has been shown that the training of the mixture models combined with dynamic feature extractors requires the inter-contrastive loss to effectively distinguish components of different classes from each other. In addition to that, to ensure diversity among same-class components and avoid degenerate solutions, such techniques as regionalization combined with the intra-contrastive loss are required. We showed that not only do the proposed approaches deliver what was intended, but also that they can translate into significant performance gains for more complex datasets. Finally, although the more generic high-level cross-entropy loss may provide good solutions in many cases, only the most advanced variant (MIX-MCR) delivers both high predictive performance and high quality of generated mixture models, which may be important from the perspective of interpretability or potential Gaussian-based extensions.

Effective tightness. The tightness bounding plays a crucial role in stabilizing the mixture learning procedure. Setting the optimal values of inter- and intra-tightness leads to striking a balance between pushing different components from each other and actually fitting them to the data. Intuitively, the inter-tightness prefers slightly lower values than intra-tightness.

Recommended configurations. By analyzing other different hyperparameter settings and combinations of our methods we could observe that: (i) the CE loss works much better with the softmax classification method, while MC and MCR should be combined with the max-component approach, (ii) different numbers of components may be required for different data and different learning rates may also affect the optimal number, (iii) maintaining only the diagonal of the covariance matrices leads to more stable optimization and better results, (iv) the learning rate for the feature extractor dominates over the one for the mixture model, and that (v) MIX is quite memory-dependent in general end-to-end scenarios.

Memory-free scenarios. At the same time, MIX is capable of learning without a memory buffer if we use a fixed pre-trained extractor and disable the contrastive loss that is not needed in this case. Our method stands out as the best model for such class-incremental scenarios which can be very important if there are any data privacy concerns or strict memory limits.

3.3 Summary

In this chapter, we introduced a class-incremental mixture of Gaussians model (MIX) for deep continual learning. We proposed different variants of the algorithm to make it suitable for gradient-based optimization and, through an extensive experimental study, we exhibited its practical configurations and capabilities in the context of other state-of-the-art continual learning models.

As it has been shown in our experiments, there is still potential for improvements of our class-incremental mixture. Firstly, the very strict regionalization based on initial clustering may be eliminating many degrees of freedom of the algorithm when adapting the feature extractor to new classes. It would be a good idea to find a more flexible solution that do not assume any pre-training structure and allows the gradient-based procedure to fully explore potential solutions. Some publications dedicated to offline scenarios addressed this issue, for example, by utilizing an annealing procedure [183]. Secondly, similar constraints may be imposed by the static tightness hyperparameter. Probably, it could be more beneficial to either find a better (parameter-free) distance function or propose an adaptive threshold. Thirdly, it is still an open question whether it is possible to effectively train a gradient-based mixture using a full covariance matrix. Finally, it is an exciting idea to consider some kind of hybridization of the mixture models with the feature extractor to benefit from the capabilities of the former to limit interference with previously learned concepts by utilizing max-component losses. All of these potential improvements combined could provide significant performance gains in the class-incremental continual learning scenarios.

CHAPTER 4

STREAMING DECISION TREES FOR CONTINUAL LEARNING

Streaming decision trees are highly popular and effective algorithms for learning from continuously arriving data. They offer a combination of a lightweight model, adaptiveness and interpretability while being able to handle ever-growing streams of instances [109, 194]. At the same time, surprisingly, they have not been investigated from the perspective of continual learning problems imposing the need for not only integrating new knowledge into the model, but also retaining the previously learned one.

In fact, neither these trees, nor any streaming ensemble technique using them, can retain useful knowledge over time. Their success in data stream mining can be attributed to their ability to adapt to the newest information, but no research so far has addressed the fact that they cannot memorize learned concepts well over a long-term time horizon. This fundamental problem that can be found at leaves of the streaming decision trees, as they are not able to maintain information about distributions of previously seen classes.

In this chapter, we offer the first detailed analysis of Hoeffding Trees [194] in the context of catastrophic forgetting. We show that the splitting procedure for creating new leaves in this model directly contributes to the occurrence of the considered problem. To alleviate this issue, we enhance the streaming tree induction with the propagation of class-conditional attribute estimators and utilization of the class priors during entropy calculation and Bayesian classification. We demonstrate that the proposed modification of Hoeffding Tree can be used to create highly effective ensembles robust to catastrophic forgetting, allowing us to introduce Incremental Random Forest for continual learning. Finally, we empirically study the robustness of the proposed streaming decision trees through a detailed experimental study in a forgetting-aware continual learning setting. We evaluate not only the global and per-class accuracy over time, but additionally the propagation of errors and model retention after being exposed to multiple new classes.

4.1 Decision trees and continual learning

Typical scenarios of continual learning and catastrophic forgetting involve cases in which classes arrive subsequently one after another. This means that once a given class was presented it may never appear again. Extensive work on using neural networks in such scenarios showed that such settings lead to severe learning problems for them. While very little attention has been given to decision trees in similar scenarios, our preliminary studies of hybridizing convolutional networks with tree-based classifiers for continual learning indicated that streaming decision trees may struggle with exactly the same problems as neural networks. In this section, we want to emphasize this issue and propose a possible solution.

4.1.1 Forgetting in streaming decision trees

Online decision trees have been proven to be excellent algorithms for learning from stationary and non-stationary data streams [110]. However, a more in-depth analysis of the conducted experimental research may reveal that algorithms like Hoeffding Tree [194] and Adaptive Random Forest [119] have been evaluated mainly in scenarios where incoming data per class is generally uniformly distributed over time, which means that instances of different classes are reasonably mixed with each other, without long delays between them [22]. Although researchers usually take into consideration the dynamic imbalance of analyzed streams [159], they still assume that instances of all classes appear rather frequently, even if ratios between them are skewed. The class-incremental scenarios are edge cases of extreme temporal imbalance, where the older classes do not appear ever again and the newer ones completely dominate the learning process. Let us introduce the main components of the state-of-the-art streaming decision trees and analyze what consequences the given scenario has for them.

Entropy and splits. The Hoeffding Tree model is built upon two fundamental components used at leaves: (i) Hoeffding bound that determines when we should split a node, and (ii) node statistics that are used for finding the best splits. The former is defined as:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}, \tag{4.1}$$

where R is a value range, equal to $R = \log C$ for information gain calculations (C is the total number of classes), n is a number of examples seen at a node and δ is a confidence parameter. If a difference between the best potential split and the current state of the node is greater than ϵ, then there is a $1 - \delta$ confidence that the attribute introduces superior information gain and it should be used to create a split. We can express it using the following condition:

$$\Delta G(x_i, s_j) = E(x_i, s_j) - E_0 > \epsilon, \tag{4.2}$$

where the best potential information gain $\Delta G(x_i, s_j)$ is equal to the difference between the entropy after the best possible split $E(x_i, s_j)$ on an attribute x_i using a split value s_j, and before the split E_0. Although the condition alone is not directly related to the forgetting problem, the entropy values are, as we will show in the next steps.

The entropy for a given binary split s_j on an attribute x_i can be calculated as:

$$E(x_i, s_j) = \sum_{k=1}^{C} -p(c_k|x_i \leq s_j)\log(p(c_k|x_i \leq s_j)) - p(c_k|x_i > s_j)\log(c_k|x_i > s_j) \quad (4.3)$$

which simply boils down to the entropy on the left ($x_i \leq s_j$) from the split s_j and on the right ($x_i > s_j$). For the current entropy E_0 at the node we simply have:

$$E_0 = \sum_{k=1}^{C} -p(c_k)\log(p(c_k)). \quad (4.4)$$

Based on the given formulas, in order to find the best potential splits over all attributes and classes, we need to maintain two groups of estimators at leaves: (i) class priors $p(c_k)$, and (ii) conditional class probabilities $p(c_k|x_i)$. The former estimations can be easily obtained by counting occurrences of each class:

$$p(c_k) = \frac{n_k}{n}, \quad (4.5)$$

where n_k is the number of instances of class k counted for a node and n is the total number of examples received. For the latter values we use the fact that we have discrete classes and apply the conditional probability formula:

$$p(c_k|x_i) = \frac{p(x_i|c_k)p(c_k)}{p(x)}, \quad (4.6)$$

where $p(x)$ is the normalizing constant for all classes. The prior probability $p(c_k)$ can be omitted here, as a part of the prior scaling, to alleviate the class imbalance problems. The required class-conditional attribute probabilities $p(x_i|c_k)$ are modeled using Gaussian estimators, which provide a quick and memory efficient way of obtaining the required values [195]. We use triplets consisting of a count $n_{k,i}$, mean $\mu_{k,i}$ and variance $\sigma_{k,i}$ for all pairs of classes c_k and attributes x_i. By having those models we can easily apply Eq. 4.6 to obtain $p(c_k|x_i \leq s_j)$ and $p(c_k|x_i > s_j) = 1.0 - p(c_k|x_i \leq s_j)$. We end up with $p(x_i \leq s_j|c_k)$, which can be calculated using the cumulative distribution function for the standard normal distribution $\Phi_k(s_j)$. It can be expressed using the error function:

$$p(x_i \leq s_j|c_k) = \Phi_k(s_j) = 0.5(1 + erf_k(s_j/\sqrt{2}), \tag{4.7}$$

where the value of the error function can be calculated using the stored triplets.

Finally, after finding the best possible split s_j for an attribute x_i that minimizes the entropy after a split (Eq. 4.3) and passing the Hoeffding bound test (Eq. 4.2) we can split the node and estimate the total number of instances that will go to the left and right child:

$$p_l(c_k) = p(c_k)p(c_k|x_i \leq s_j) = 1 - p_r(c_k), \tag{4.8}$$

where $p_l(c_k)$ and $p_r(c_k)$ are priors for the left and right child for the given class c_k, and x_i is the selected split attribute.

By default, we omit the estimation of all $p(c_k|x_i)$ after the split as it is a non-trivial task, which most likely cannot be quickly solved in the current form of the algorithm. This fact has a crucial impact on the streaming decision trees in the class-incremental scenario as we will show in the subsequent paragraphs.

Classification at leaves. After forwarding an incoming instance to a leaf in the decision tree, it is classified using majority voting based on the class priors. To improve the classification process the simple procedure is often combined with a naive Bayes classifier [109], which can be easily applied using the already stored estimators:

$$p(c_k|\mathbf{x}) = \frac{p(\mathbf{x}|c_k)p(c_k)}{p(\mathbf{x})}, \tag{4.9}$$

where $\mathbf{x}$ is the vector of input attributes and $p(\mathbf{x}|c_k)$ is equal to:

$$p(\mathbf{x}|c_k) = \prod_{i=1}^{m} p(x_i|c_k), \tag{4.10}$$

where m is the number of features. Each $p(x_i|c_k)$ can be obtained using the Gaussian density function.

Forgetting scenario. After the introduction of the leaf components and required calculations, let us now consider what will happen in the class-incremental scenario after subsequent splits. In Fig. 15 we can see an example of a sequence of 3 class batches. In the beginning, there are only instances of the first class (C0) for which the algorithm accumulates values for the prior count (Eq. 4.5) and conditional estimators (Eq. 4.6) only at the root, since there is no need for a split.

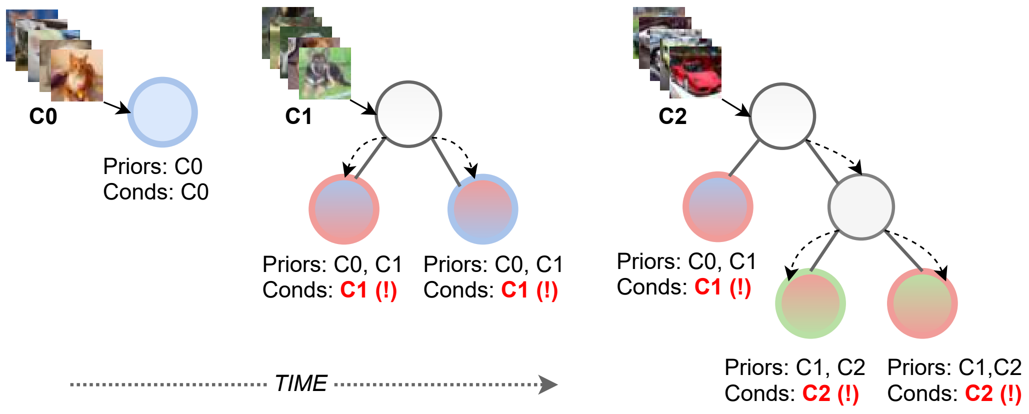


Fig. 15.: Catastrophic forgetting in streaming decision trees learning from a class-incremental sequence.

Next, the second class (C1) starts arriving and at some point the Hoeffding Tree algorithm finds a good split, which creates two additional nodes and distributes priors accordingly to Eq. 4.8. After this step, the child nodes have some smaller priors for C0 and C1, however, the conditional estimators have been reset by default. Although we can assume that after the split some instances of class C1 can still appear and rebuild the conditional estimators, there is no chance that the same will happen for C0, which means that while its priors will be good for now, its conditional estimators (Eq. 4.6) will remain equal to zero, resulting in an inability of the naive Bayes classifier (Eq.

4.9) to recognize this class.

When the next class starts coming (C2) we can already observe a problem – since there are no instances of C0, its $p(c_0|x_i)$ is still equal to zero, which leads to the situation in which the older class is completely ignored during the entropy calculations when looking for a split (Eq. 4.3). Finally, once a new split is created, there will be no prior for the class at the newest leaves, since based on Eq. 4.8 it has to be zeroed. This concludes the learning process for class C0 which has been completely erased at the third level of the tree, and which may very likely disappear from the model completely. Even worse is the fact that the same will most likely happen to C1 and C2 as soon as new classes arrive.

Based on the analysis, we can conclude that in the class-incremental scenario, catastrophic forgetting in streaming decision trees manifests itself in three ways: **(i)** by excluding older classes from a meaningful contribution to the best split criterion, **(ii)** by disabling the conditional classification, and finally **(iii)** by erasing priors which leads to complete class forgetting at a given node.

4.1.2 Overcoming catastrophic forgetting

The observations from the previous section clearly indicate that the source of the problem with forgetting can be found at leaves and their conditional estimators. It is worth emphasizing that this issue practically does not exist in most of the commonly used data stream benchmarks, which provide instances of different classes most of the time during the learning process. In such a case, the estimators can always rebuild themselves after new instances arrive, preventing them from forgetting most of the classes. The longer are gaps between subsequent instances of one class, the higher the chance that the class will be temporarily or forever forgotten.

To make a step towards solving the introduced problem in Hoeffding Trees, we propose using a rough class-conditional attribute estimation after the split to prevent the model from forgetting older classes. The approach consists of two modifications: **(i)** propagating class-conditional attribute estimators (needed for Eq. 4.7) to children of a node being split, and **(ii)** keeping the class priors in the entropy and naive Bayes calculations to calibrate the rough estimation.

Estimator propagation. We can simply achieve the first step by copying the Gaussian parameters of each class-conditional distribution $p_{t-1}(x_i|c_k)$ before split at time step t to the left node with $p_{t,l}(x_i|c_k)$ and to the right one with $p_{t,r}(x_i|c_k)$, which results in:

$$p_{t,l}(x_i|c_k) = p_{t,r}(x_i|c_k) = p_{t-1}(x_i|c_k), \tag{4.11}$$

for each class c_k and attribute x_i. This is obviously a very rough estimate, however, since we assume simple Gaussian distributions, the error does not have to be critical and may provide more benefits than obstructions. Most likely, providing any platform for an older class is more important than the risk of making the estimation error. In addition, the estimate may still be fine-tuned by instances that come to this node before the class batch ends.

Prior scaling. By sticking to the prior probabilities $p(c_k)$ in the entropy calculations (Eq. 4.3) and Bayesian classification (Eq. 4.9), we attempt to somehow adjust the rough estimate from the previous step. Since the split class priors are relatively well-estimated, we can utilize them to softly scale the class-conditional distributions to become more adequate to the state after the split. Although this step does not change the shape of the distribution horizontally, it may increase or decrease the influence of the distribution by scaling it vertically based on the formula:

$$p_t(x_i|c_k) = p_{t-1}(x_i|c_k)p_t(c_k). \tag{4.12}$$

Ensembles. Finally, the modified Hoeffding Tree can be simply used as a base learner of the Incremental Random Forest, which is an Adaptive Random Forest without change detectors and node replacement mechanisms. The only difference between the standard forest and the ensemble using our modified tree is that we have to keep statistics for all attributes at leaves, not only for those within a random subspace, since we do not know which attributes will be needed at a lower level. By combining the robustness of ensemble techniques with improvements of the base learner we may potentially alleviate the catastrophic forgetting problem even more.

4.2 Experimental study

In the following experiments, we aim at showing that our proposed modifications of the Hoeffding Tree algorithm are capable of alleviating the catastrophic forgetting in decision trees learning from class-incremental streams, allowing for the application of these models in such scenarios. Our goal was to answer the following research questions.

- **RQ1**: Does the proposed algorithm effectively address the problem of catastrophic forgetting in streaming decision trees?
- **RQ2**: Can the presented decision tree be utilized as a base learner of a random forest? Does it further improve the classification performance?
- **RQ3**: Is it possible to solve the presented problem by using a different already available ensemble technique?

In order to improve the reproducibility of this work, all of the presented algorithms and details of the evaluation have been made available in a public repository: github.com/lkorycki/lldt.

4.2.1 Data

To evaluate the baseline and proposed models in the scenario of continual learning and catastrophic forgetting, we used popular visual datasets commonly used for the given task. The first three were used as simpler sequences consisting of 10 classes: MNIST, FASHION, SVHN. Next, we utilized 20 superclasses of the CIFAR100 dataset (CIFAR20), as well as we extracted two 20-class subsets of the IMAGENET: IMAGENET20A and IMAGENET20B. All of the sets were transformed into class-incremental sequences in which each batch contained only one class and each class was presented to a classifier only once. Although, our models can work with recurring classes, we considered only the most standard class-incremental scenario to limit the scope of experiments. The evaluated models were processing the incoming batches in a streaming manner, one instance after another.

The MNIST and FASHION datasets were transformed into a series of flattened arrays (from raw images), which provided us with feature vectors of size 784. The rest of the used benchmarks were pre-processed using pre-trained feature extractors. For SVHN and CIFAR20 we used ResNeXt-29 with its cardinality equal to 8 and using widen factor equal to 4. We extracted the output of the last 2D average pooling and processed it with an additional 1D average pooling, which resulted in a feature vector consisting of 512 values. For the IMAGENET-based sets we directly utilized the output of the last average pooling layer of the ResNet18 model, which once again gave us 512-element vectors.

4.2.2 Algorithms

In our experiments, we compared the proposed single tree (**HT+AE**) with the original streaming algorithm (**HT**) [194], as well as the incremental random forest using our base learner (**IRF+AE**) with its baseline (**IRF**) to answer the first two research questions. Next, we evaluated other ensemble techniques to check whether it is possible that a solution to the introduced problem lies solely in a different committee design (the last research question). We investigated drift-sensitive Adaptive Random Forest (**ARF**) [119], online bagging without random subspaces per node (**BAG**) , online random subspaces per tree (**RSP**) [196] and the ensemble of 1-vs-all classifiers (**OVA**) [197].

All of the algorithms used Hoeffding Trees as base learners with confidence set to $\delta = 0.01$, bagging lambda equal to $\lambda = 5$, split step $s = 0.1$ (10% of a difference between the maximum and minimum attribute value) and split wait equal to $w = 100$ for all sets except for the slightly smaller IMAGENET-based ones for which we set $w = 10$. All of the ensembles used $n = 40$ base learners.

4.2.3 Evaluation

Firstly, for all of the considered sequences, we measured **hold-out accuracy** [198] per each class after each class batch and used it to calculate the average accuracy per batch and the overall average for a whole sequence [12]. Secondly, we collected data for **confusion matrices** after each batch to generate the average matrices which could help us illustrate the bias related to catastrophic forgetting. Finally, we measured the **retention** of the baseline and improved algorithms to show how well the given models remember previously seen concepts.

4.2.4 Results

Analysis of the average predictive accuracy. Tab. 6 presents the average accuracy over all classes for all six used class–incremental benchmarks. This is the bird's eye view of the problem and the performance of the analyzed methods, allowing us to assess the general differences among the algorithms. We can see that the standard HT and IRF were significantly outperformed by the proposed HT+AE and IRF+AE approaches. For HT the proposed propagation of class-conditional attribute estimators and storing the class priors led to very significant improvements on all datasets, which is especially visible on CIFAR20 (almost 0.3) and IMAGENET20A (0.2). Similar improvements can be observed for IRF, especially for CIFAR20 where the modifications led to 0.28 improvement. The SVHN benchmark shows the smallest improvements out of all six datasets, which can be explained by the extractor potentially being very strongly fine-tuned for this problem. Thus extracting well-separated class embeddings may slightly alleviate the catastrophic forgetting on its own (although the proposed modifications still help).

The impact of different ensemble architectures. To truly understand the impact of catastrophic forgetting on HT and IRF, we decided to see if other ensemble architectures may behave better in class-incremental continual learning scenarios. Tab. 6 presents results for four other popular streaming ensemble architectures. We can see that all of them performed poorly on every dataset, offering inferior predictive

Table 6.: The average accuracy on all class-incremental sequences.

Model	MNIST	FASHION	SVHN	CIFAR20	IMGN20A	IMGN20B
HT	0.6283	0.5720	0.8845	0.3511	0.4589	0.5301
HT+AE	**0.8398**	**0.7037**	**0.9510**	**0.6497**	**0.6530**	**0.6730**
IRF	0.8662	0.7355	0.9334	0.4467	0.6890	0.7500
IRF+AE	**0.9645**	**0.8698**	**0.9733**	**0.7298**	**0.7777**	**0.8121**
ARF	0.2929	0.2929	0.2929	0.1799	0.2411	0.2849
OVA	0.3416	0.2929	0.5033	0.1805	0.3842	0.3847
BAG	0.7096	0.6446	0.9029	0.3737	0.5709	0.6635
RSP	0.6202	0.5898	0.9087	0.3734	0.6337	0.6995

accuracy to the baseline IRF. This shows that the choice of an ensemble architecture on its own does not offer improved robustness to catastrophic forgetting. As a result, we have a good indication that our modifications of the HT splitting procedure are the sole source of the achieved impressive gains in accuracy. However, a more in-depth analysis of these models will allow us to gain better insights into the nature of catastrophic forgetting in streaming decision trees.

Analysis of the class-batch performance. Fig. 16 depicts the average accuracy after each class appearing incrementally. This allows us to visually analyze the stability of the examined methods and their response to the increasing model size (when more and more classes need to be stored and remembered). We can see that both proposed HT+AE and IRF+AE offered significantly improved stability over the baseline approaches, maintaining their superior predictive accuracy regardless of the number

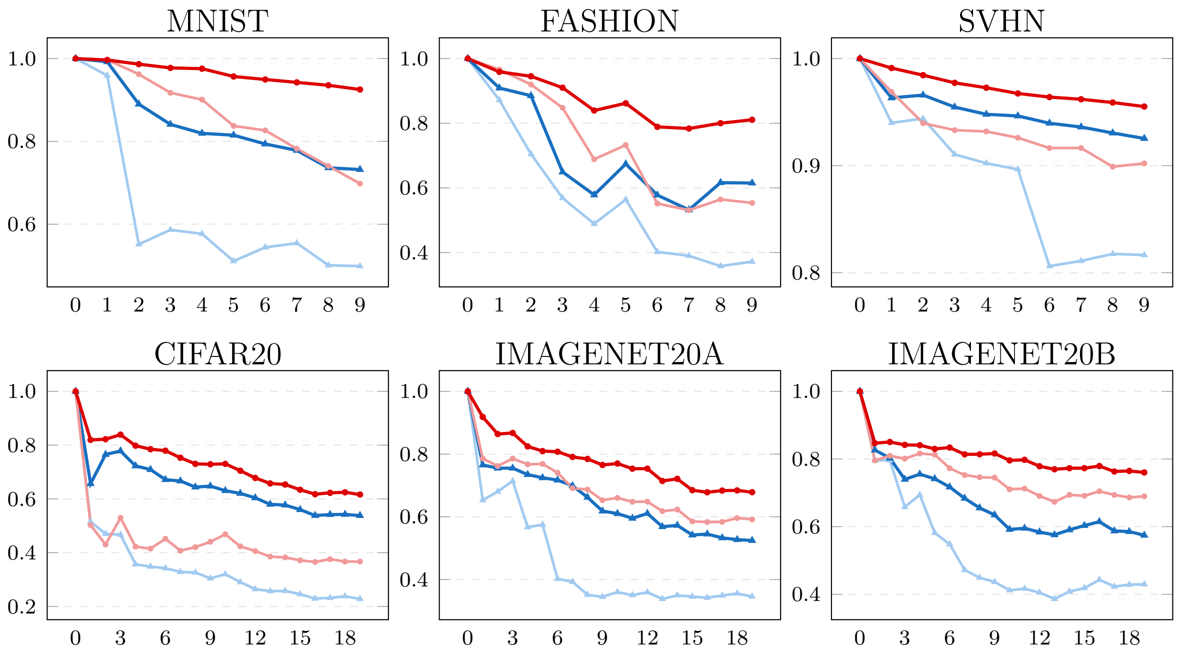


Fig. 16.: Average class accuracy for the baseline tree-based models (▲ HT, ● IRF) and the proposed ones (▲ HT+AE, ● IRF+AE) after each class batch.

of classes. Additionally, we can see that the baseline models tended to deteriorate faster when the number of classes became higher (e.g., HT and IRF on MNIST and FASHION). At the same time, the proposed modifications could accommodate all the classes from the used benchmarks without destabilization of their performance. It is worth noting that HT+AE was often capable of outperforming IRF. This is a very surprising observation, as the modification of class-conditional estimators allows a single decision tree to outperform a powerful ensemble classifier. This shows that the proposed introduction of robustness to catastrophic forgetting into streaming decision trees is a crucial improvement of their induction mechanisms.

Analysis of the class-based performance. Fig. 17 presents the accuracy per batch on selected classes. This allows us to understand how the appearance of new classes affects the performance for previously seen ones. We can clearly see that both HT and IRF were subject to catastrophic forgetting, very quickly forgetting the old classes. While they were very good at learning the newest concept, their performance degraded with every newly arriving class, showing their capabilities of aggressively adapting to new knowledge, but not retaining it over time. This was especially vivid for the first class for each dataset (C0), where it was completely forgotten (i.e., accuracy on it drops to zero) as soon as 1-2 new classes appeared. The proposed propagation of class-conditional attribute estimators and storing the class priors in HT+AE and IRF+AE led to a much better retaining of knowledge extracted from old classes. In some cases (e.g., CIFAR20 or IMAGENET20) we can see that the accuracy for old classes remained almost identical through the entire duration of the continual learning process. This is a highly sought-after property and attests to the effectiveness of our proposed modifications.

Analysis of the confusion matrices. Fig. 18 depicts the confusion matrices av-

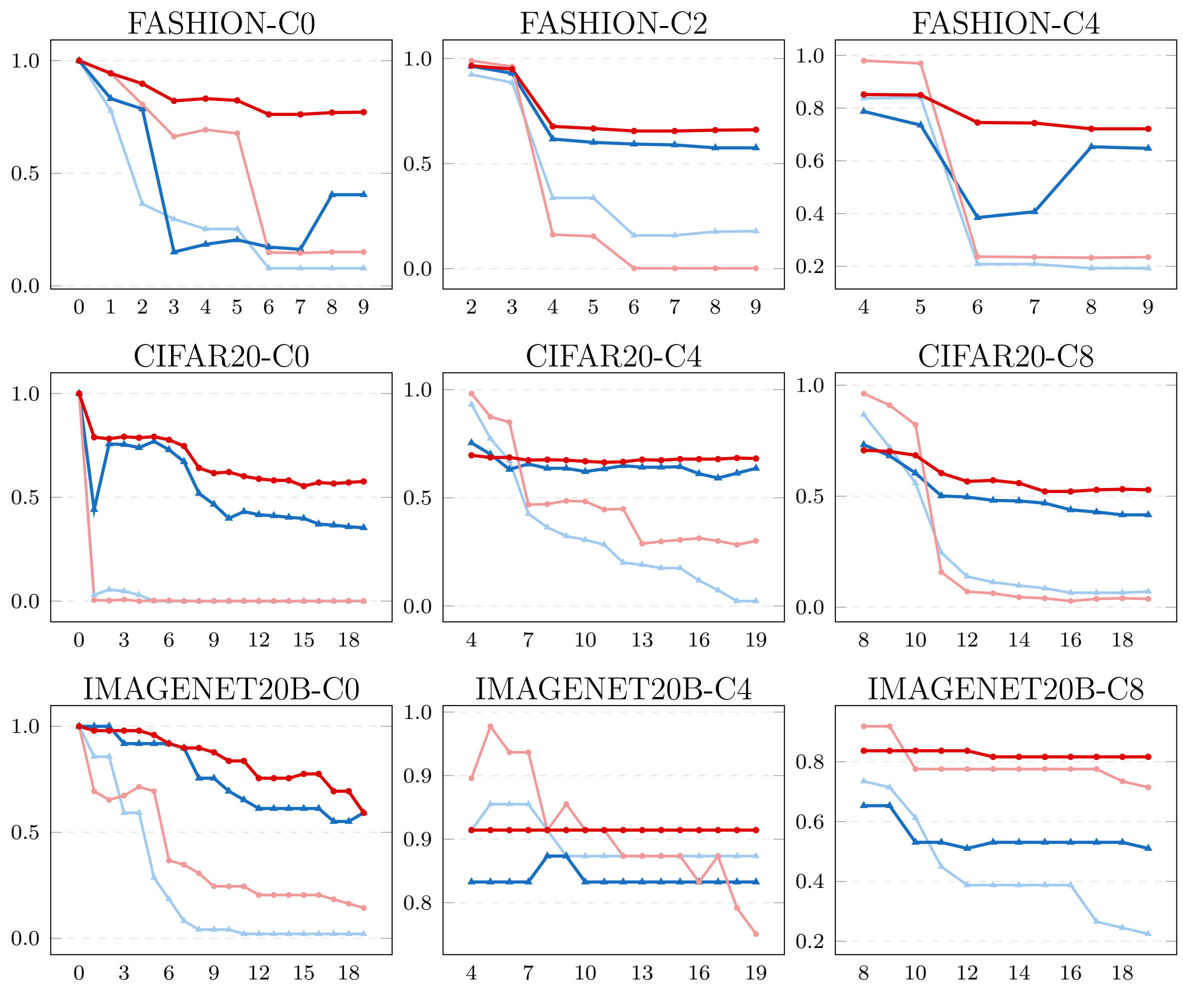


Fig. 17.: Average accuracy for selected classes of FASHION, CIFAR20 and IMAGENET20B for the baseline tree-based models (▲ HT, ● IRF) and the proposed ones (▲ HT+AE, ● IRF+AE) after subsequent class batches.

eraged over all examined datasets (10 classes from each for the visualization sake). Based on that we can directly compare how errors are distributed among classes for HT vs. HT+AE and IRF vs. IRF+AE. We can see that the proposed modifications in HT+AE and its ensemble version led to a much more balanced continual learning procedure that both avoided the bias towards the newest class (i.e., is robust to catastrophic forgetting) and the bias towards older classes (i.e., offers capabilities for

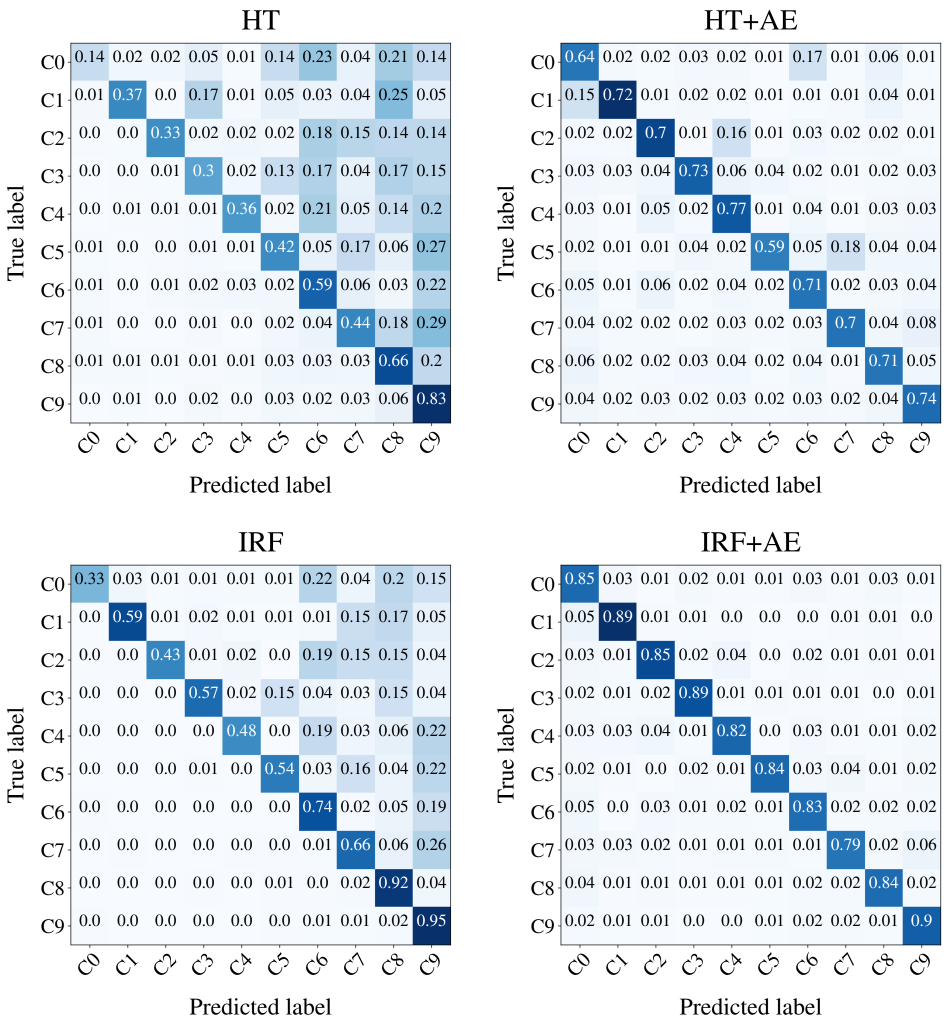


Fig. 18.: Average confusion matrices.

incorporating new information into the model in an effective manner). These confusion matrices further confirm our observations that our proposed modifications lead to robust streaming decision tree induction for continual learning.

Analysis of the retention of information. Fig. 19 shows the average retention of information about a class after +k new classes appeared. This helps us analyze how each of examined models manages its knowledge base and how flexible it is to add new information to it. An ideal model would perfectly retain the performance

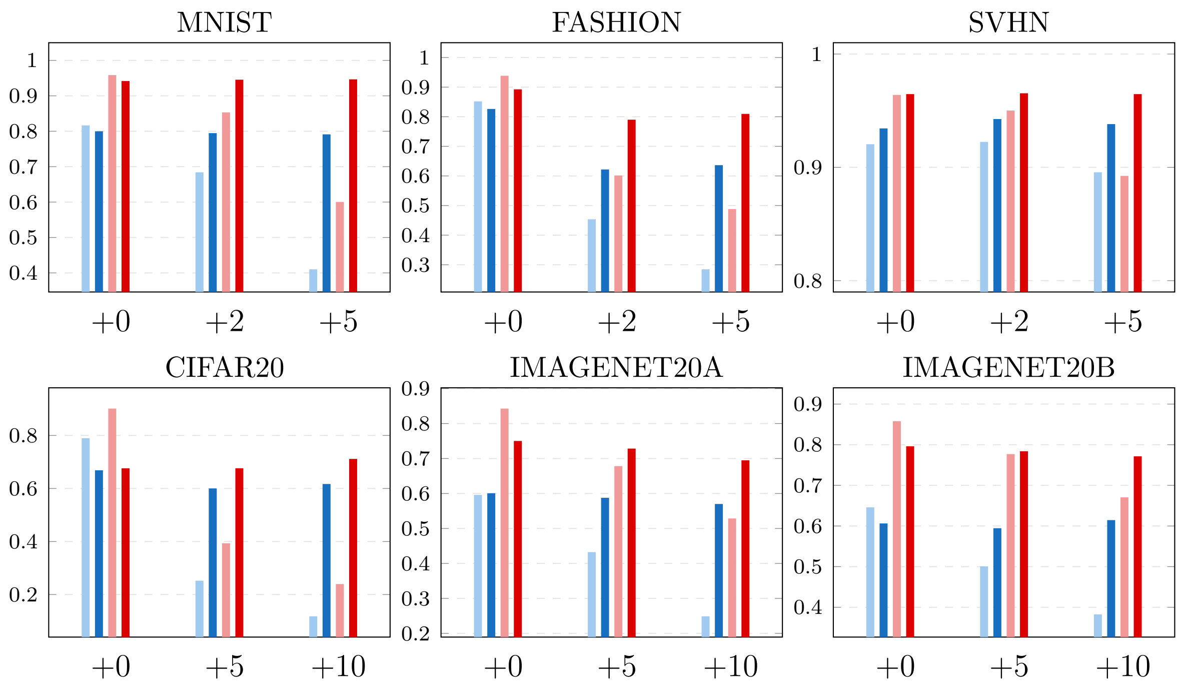


Fig. 19.: Average retention after +k class batches since the moment a class appeared for: ■ HT, ■ HT+AE, ■ IRF, ■ IRF+AE

on every previously seen class, regardless of how many new classes it has seen since then. We can see that the baseline HT and IRF offered very good performance on the newest class but drastically dropped it after seeing as few as 2 new classes. This further enforces our hypothesis that standard decision trees and their ensembles cannot avoid catastrophic forgetting and thus cannot be directly used for continual learning. However, when we enhance HT with the proposed propagation of class-conditional attribute estimators and storing the class priors, we obtain a streaming decision tree that can learn new information almost as effectively as its standard counterpart, while offering excellent robustness to catastrophic forgetting (**RQ1 answered**). Furthermore, we can see that HT+AE can be utilized as a base learner for ensemble approaches, leading to even further improvements (**RQ2 answered**).

Batch-based performance of the ensemble architectures. Fig. 20 depicts the average accuracy after each class appearing incrementally for the reference ensemble approaches. This confirms our observations from the earlier point that the ensemble architecture itself does not have any impact on the catastrophic forgetting occurrence. Reference methods use different ways of data partitioning (subsets of instances, features, or classes), but none of them allowed for better retention of old information. What is highly interesting is that HT+AE (a single decision tree) could outperform any ensemble of trees that do not use our proposed modifications. This shows the importance and significant impact of propagation of class-conditional attribute estimators and storing the class priors on the usefulness of streaming decision trees for continual learning. Therefore, catastrophic forgetting can be avoided by using a robust base learner, not changing the ensemble structure (**RQ3 answered**).

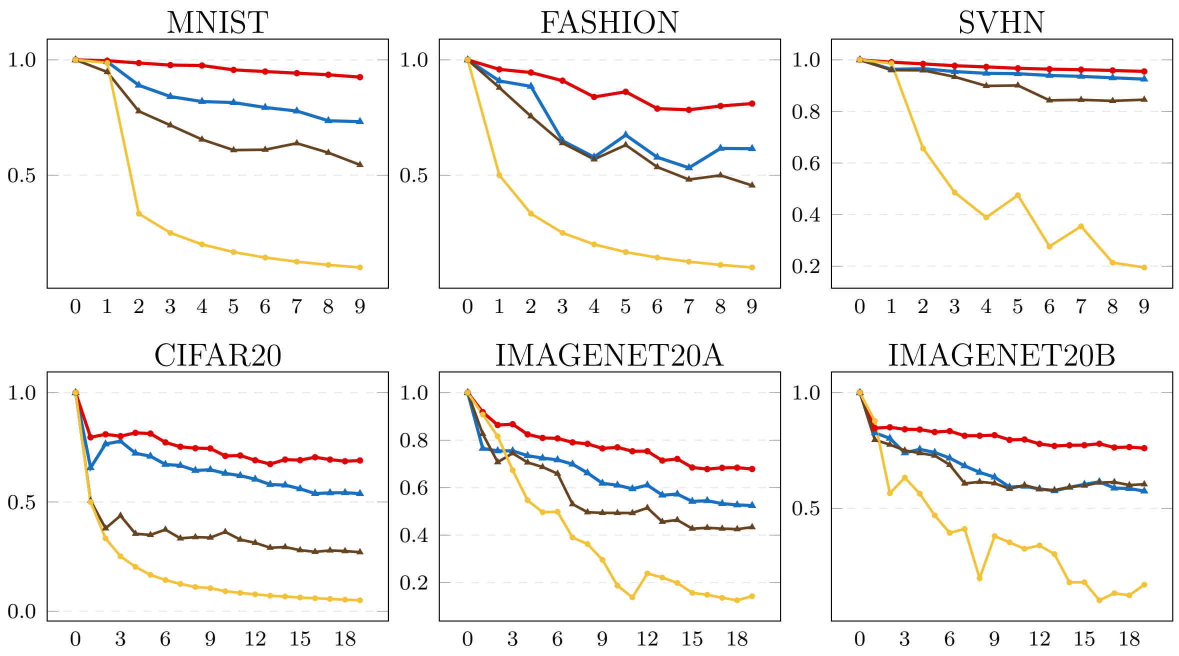


Fig. 20.: Average class accuracy for other baseline models (▲ OVA, ▲ BAG) and the proposed ones (● HT+AE, ● IRF+AE) after each class batch.

4.3 Summary

In this chapter, we identified and emphasized the issue of catastrophic forgetting that occurs when traditional streaming decision trees attempt to learn in class-incremental continual learning scenarios. Through an in-depth analysis of the Hoeffding Tree algorithm, we found out that the source of the algorithm's weakness comes from the lack of additional support for class-conditional attribute estimators, which tend to forget older classes after splits. The issue critically affects different aspects of tree-based learning, ranging from the procedure for finding new splits to the classification at leaves.

To solve the introduced problem, we proposed a rough estimation of the conditional distributions after a split, based on distributions and priors aggregated at a node before it is divided. Our extensive experimental study has shown that this simple yet effective approach is capable of providing excellent improvements for both single trees and incremental forests. As a result, we demonstrated that the proposed method turns the standard streaming trees into learners suitable for continual learning scenarios.

In future works, we plan to find more precise estimators, which may need to be supported by some local experience replay utilizing small buffers of either input instances or prototypes. In addition to that different split conditions more robust to different characteristics of data, e.g., based on the Hellinger distance [199, 200] or the McDiarmid's bound [201], can also be considered.

CHAPTER 5

INSTANCE EXPLOITATION FOR LEARNING TEMPORARY CONCEPTS FROM SPARSELY LABELED DRIFTING DATA STREAMS

In data stream mining, we assume that new information arrives continuously and our learning model must be capable of making quick decisions and incorporating new data on-the-fly. Designing machine learning methods for data streams can be viewed as a multi-criteria optimization, where one aims at reaching a trade-off between several factors that define efficacy in streaming environments [22]. Firstly, since we cannot store the entire data stream, we should carefully choose which ones to keep and which ones to discard (or, in the case of online learning, assume that each instance must be discarded after a single pass). Secondly, the trained model must quickly provide predictions for new arriving instances to avoid bottlenecks [19]. Finally, when faced with concept drift, we need not only to detect it as soon as possible, but also adapt the model to the new state of the stream with the lowest possible latency. The time between the occurrence of the concept drift and classifier re-training is known as a drift recovery rate [15].

Reducing the time when our classifier is incompetent (i.e., not adapted to the new concept) should be seen as one of the main goals of learning from non-stationary data streams. The speed of adaption can be affected by the training procedure of the classifier, as well as by the access to data from the new concept – the more instances representative for the new concept we have, the more likely we are to reduce the recovery time. However, we must be aware that fully supervised learning from data streams is an unrealistic scenario. It would require access to an oracle providing class labels for every single instance in the stream. As such labels usually come from a domain expert, we cannot assume that we can obtain all of them – we are restricted by both the budget (i.e., the time of the domain expert is costly) and human limitations (instances arrive with such a speed and in such a volume that is impossible for a human to tackle). A realistic scenario for data stream mining assumes either limited access to ground truth [163], or delayed labeling [202]. This restricted access to labeled data strengthens another problem known as underfitting, making effective drift adaptation even more challenging.

In this chapter, we address an extremely important challenge of data stream mining: how to improve learning from drifting data while having only strictly limited access to class labels. While many of the existing works focus on how to reasonably choose instances for labeling, in this work, we aim at providing further improvements based solely on the few labeled instances given to us and at no additional cost. Our assumption is that any obtainable improvement under strict labeling constraints is of vital usefulness to the underlying classifier, offering faster adaptation to new concepts. The main contributions of this work include the following.

- **Enhancing drift adaptation with instance exploitation.** Our idea is to intensively exploit labeled instances at our disposal, aiming at reinforcing adaptation to non-stationary concepts and providing a faster classifier recovery. By reusing the labeled instances we should be able to avoid underfitting and offer a faster and more accurate reaction to concept drift.

- **Exploitation techniques for improved drift adaptation.** We introduce three techniques for instance exploitation to empower active learning from sparsely labeled drifting data streams. They are based on a reactive sliding window that uses one of three probabilistic sampling techniques to select instances for a more aggressive exposure to the online classifier.

- **Ensemble architectures to avoid overfitting.** In order to minimize the risk of overfitting, we propose two simple, yet effective ensemble architectures. They are based on two paired learners, one preferring aggressive instance exploitation and the other learning in a standard way. Our architectures are capable of dynamic switching between these models, with an added procedure for improving the weaker of the two learners.

- **Flexible framework.** Our proposal is a universal wrapper that can be used to enhance any online active learning algorithm dedicated to data streams.

- **Insights into the role of enhanced drift adaptation.** We offer an exhaustive experimental study on both artificial and real data streams, paired up with an in-depth analysis that offers unique insights and a better understanding of how to efficiently improve the adaptation to concept drift by using already labeled instances without any additional labeling costs.

5.1 Learning on a budget and data dynamics

Analogously to the traditional batch-based machine learning approaches, the critical problem of limited supervision has been addressed by some semi-supervised [171, 172, 175, 174, 203] and unsupervised methods [176, 177], which aim at alleviating the lack of supervision by utilizing unlabeled instances. While utilizing unlabeled data seems perfectly adequate for scenarios involving streams, it may be surprising why the actual number of works using it is so limited. It is possible that obtaining significant improvements from unreliable unlabeled data is very difficult in dynamic environments. Indeed, one of the main assumptions of semi-supervised learning is that in order to work correctly unlabeled data have to be drawn from the same, stable distributions as observed labeled instances [204]. This may be prohibitive in many streaming scenarios, especially if we add the fundamental cluster and smoothness requirements. In fact, results from some publications suggest that either feasible improvements from unlabeled data are barely significant or very unstable and highly dependent on characteristics of a specific stream [203].

As a result of the potential weak spots, it is not a coincidence that significant attention was given to methods shifted towards supervised approaches, which are more likely to provide reliable information regardless of a state of a stream. These methods are represented mainly by active learning [162, 163, 205, 206, 207].

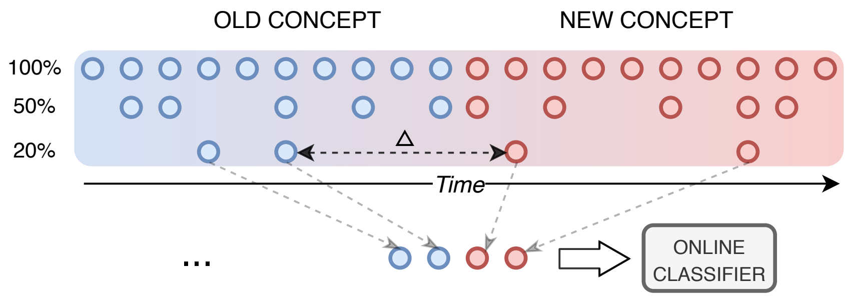


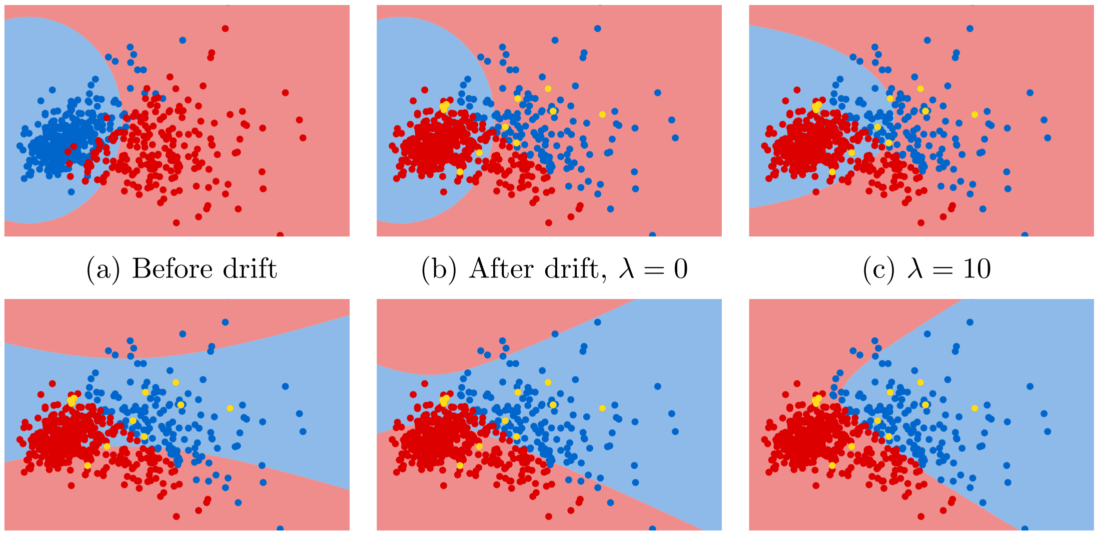
Fig. 21.: Sparse labeling problem.

Unfortunately, all of the existing active learning strategies are significantly impaired when the label query budget (i.e., the number of instances allowed to be labeled) is small. In many real problems labeling even as little as 1% of instances from the stream may be too costly. As a consequence, even if a strategy picks only valuable objects to be labeled and used, the effect of the single update may still be insufficient. Furthermore, it is crucial to remember that when we deal with evolving data streams and only low budgets are available, we usually get very sparse snapshots of observed concepts. Arriving labeled objects often come from distant parts of the stream, so if the data is non-stationary, they likely represent different distributions constituting temporary concepts (Fig. 21), especially when the rate of incoming instances is low.

As a result of these two observations, it is very likely that, while working with dynamic streams and a substantially limited labeling budget, we will encounter the underfitting problem. In fact, we may even completely overlook some of the ephemeral concepts, leaving them unnoticed. Thus, the very few labeled objects we obtain are essential for keeping our models up-to-date and we should exploit them as much as possible to avoid the waste of potential benefits. Finally, since active learning methods can be seen as exploration methods and semi-supervised ones as regularization-exploitation, approaches combining both of them are a promising research direction [176, 203]. However, due to the mentioned problems with dynamic environments and using unsupervised input, in this work, we focus on investigating how significant improvements can be obtained solely from the exploitation of scarce but reliable supervised information, provided with actively selected instances. To the best of our knowledge, it is the first such approach to the problem.

5.2 Risky adaptation

Our hypothesis is that when dealing with temporary and evolving concepts under strictly limited access to labeled data, it is reasonable to aggressively exploit the data we currently have in order to overcome underfitting and obtain efficient up-to-date models. Let us consider the following example. In Fig. 22 we can see an adaptation process for a classifier while recognizing two classes (marked as red and blue points). Fig. 22a presents a state just before a drift. The classifier has learned the previous concept with sufficient accuracy, however, after the change (Fig. 22b) its model becomes practically useless. There is a need to update it, but due to a limited budget only few labeled instances selected by the active learning strategy are available (yellow). Since we deal with data streams, we can assume that the data



(a) Before drift (b) After drift, $\lambda = 0$ (c) $\lambda = 10$

(d) $\lambda = 50$ (e) $\lambda = 100$ (f) $\lambda = 200$

Fig. 22.: Decision boundaries created by an incremental classifier, while tackling a concept drift, after learning on λ duplicates of the selected instances.

points are currently stored in a sliding window.

In the standard active learning approach, the chosen objects are used only once ($\lambda = 0$) and the effect is unacceptable – the class boundary did not change at all. This observation is connected with the nature of many incremental parameters maintained by the classifiers. Their internal estimators, cannot be updated efficiently using only few new values. The most straightforward solution is to reuse the limited labeled instances several times to influence the metrics in a more significant manner and boost the adaptation. We can clearly see that after some number of repetitions λ the decision boundary is reshaping towards a proper model – we exploit the acquired knowledge (Fig. 22c - 22f). The final result is highly dependent on the representativeness of the instances selected by active learning, but this has been already investigated in other publications [163]. While we are aware of the fact that such an approach may potentially cause overfitting to the few instances [28, 208], our assumption is that it is less important than the underfitting problem encountered before the risky adaptation was performed.

Such a wrapper can work with many online base learners, especially with those dedicated to evolving data streams. The currently most important online algorithm – Adaptive Hoeffding Tree (AHT) – may benefit from such a method since its internal splits are rearranged only if a significantly large amount of data differs from a previous concept [109]. In the case of the stochastic gradient descent classifier (SGD) or neural networks, while exposing wrongly classified instances several times, we can speed up the gradient-based optimization of a loss function. Of course, while using such algorithms we can also directly focus on the criterion, for example by adjusting the learning rate or momentum, however, such solutions are limited only to the gradient-based classifiers [209]. Finally, Naïve Bayes may also work with the introduced method since its internal estimators can be updated incrementally. It is also worth noting that we may avoid reusing instances several times by employing weight-sensitive classifiers, which can be straightforwardly adapted for this approach. In this work, due to their greater flexibility, we focus only on the more generic instance-based algorithms.

This approach can be associated with a mechanism similar to experience replay, which is a method of tackling catastrophic forgetting in deep neural networks [210]. The fundamental difference is that instead of reusing instances for stabilizing models of incoming classes, we want to erase outdated knowledge more quickly and practically *overfit* an online model to current concepts. The method is also related to resampling, known from imbalance learning, which also utilizes some instances multiple times [146].

One may notice that another popular approach, that allows for the most current instances to significantly affect a maintained model, is to simply retrain the model from scratch, using a collected batch of data [1]. However, while such methods work well with sudden concept drifts, they are not suitable for incremental changes. In addition, forgetting the whole priorly gathered information may be a wrong idea if the budget is limited and if we take into consideration the fact that streams may be characterized by hybrid drifts or changes affecting only some parts of the analyzed space. In the example above (Fig. 22), we would lose all the information about the red points at the bottom, which were not influenced by the concept drift. We do not want to completely forget previous knowledge, just quickly and incrementally adapt to the current situation when lacking labels. One well-known way to overcome this problem is to use ensembles that consist of classifiers constructed on the basis of the most current batches [122]. Still, such an approach is much more time and memory-consuming than single-classifier frameworks. Another problem is that highly limited budgets may be prohibitive in the context of building a diverse pool of classifiers. Finally, all the strategies using the retraining approach, are highly dependent on effective concept drift detectors, which increase the overall complexity of a system and are often affected by the limited access to labeled instances as well.

5.2.1 Instance exploitation

We propose a simple, generic, single-classifier framework that uses aggressive online updates based on instances selected for labeling by an active learning strategy. Our implementation utilizes a reactive sliding window that stores the most recent labeled instances and exposes them several times to the online classifier in order to exploit the knowledge that comes with them. We present three different probabilistic sampling methods that determine how the instances are selected from the window. The general assumption is that while active learning will effectively explore a decision space, by selecting the most representative seeds, the instance exploitation wrapper will sufficiently utilize them in order to enhance the adaptation of a classifier using only few labels. That should help us tackle the underfitting problem.

5.2.1.1 Framework

The generic wrapper framework is presented in Alg. 1. When a new instance $\boldsymbol{x}$ appears, the actual budget spending $\hat{b}$ is checked. Since the data stream is infinite by definition, the current spending has to be estimated as a ratio of objects that have been already labeled to the total number of registered instances [163]. If the value is below an available budget B, we use the active learning strategy to check if the instance is worth labeling. Some of the online strategies that can be used here were presented in [163]. The methods are focused on selecting instances close to the decision boundary, based on the uncertainty of a classifier. The idea is that those regions are most likely to be affected by drifts. In order to make the strategies more exploratory and sensitive to changes in other subspaces, randomization techniques can be utilized. One of the methods implementing hybrid comprehensive querying

Algorithm 1: Combining active learning with instance exploitation.

```
Data: labeling budget $B$, ActiveLearningStrategy,
      ExploitationStrategy, window size $\omega_{max}$
Result: classifier $L$ at every iteration
Initialization: $\hat{b} \leftarrow 0$, $\boldsymbol{I} \leftarrow []$, $\boldsymbol{W} \leftarrow []$
repeat
    receive incoming instance $\boldsymbol{x}$;
    if $\hat{b} < B$ and ActiveLearningStrategy $(\boldsymbol{x}) = true$ then
        request the true label $y$ of instance $\boldsymbol{x}$;
        update labeling expenses $\hat{b}$;
        update classifier $L$ and sliding window $\boldsymbol{W}$ with $(\boldsymbol{x}, y)$;
        $I \leftarrow$ ExploitationStrategy $(\boldsymbol{W})$;
        foreach $i \in \boldsymbol{I}$ do
            update classifier $L$ with $\boldsymbol{w}_i \in \boldsymbol{W}$;
until stream ends;
```

is the RandVar algorithm. In addition to the complex sampling, the strategy uses a variable threshold balancing budget spending. We select this method as our default strategy due to its theoretically sound design.

After receiving a positive response, the object is queried (we obtain its true label y), the current labeling expenses are increased and the classifier is updated with the labeled instance. Then, we activate our instance exploitation strategy, which selects indices $\boldsymbol{I}$ of previously acquired labeled instances in order to reuse them and boost the adaptation process. Since we want to adapt a predictive model to current concepts, our exploitation methods work with a batch of the most recent instances. We maintain a sliding window $\boldsymbol{W}$ that is updated with every labeled instance we receive, before using an exploitation strategy. We define the window $\boldsymbol{W}$ as a sequence of $\omega \in \langle 1, \omega_{max} \rangle$ instances $\boldsymbol{w}_i \in \boldsymbol{W}$, where $i \in \langle 1, \omega \rangle$, the oldest instance is $\boldsymbol{w}_1$ and the newest one is $\boldsymbol{w}_\omega$. It is important to remember that for very low budgets, wide windows may not be able to represent the actual concept properly, since the relatively small number of labeled instances will cover only few batches. For example, if $B = 1\%$, $\omega_{max} = 1000$ and a stream consists of 100 000 instances, then the window will not slide at all, since all labeled instances will cover exactly one such window. Therefore, in order to have a reactive sliding window, when using a low budget or dealing with a low-rate stream, its size must be adequate.

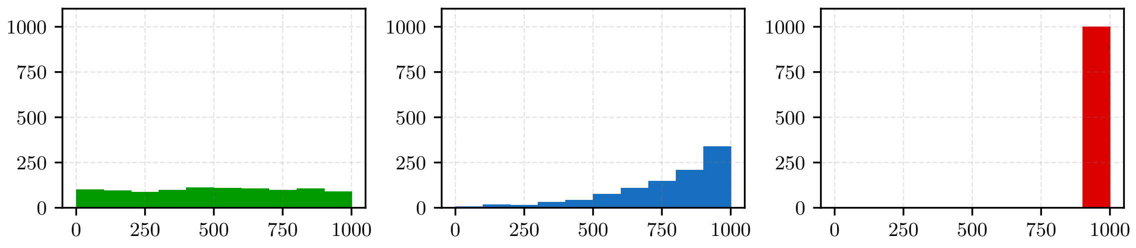
5.2.1.2 Exploitation strategies

Our exploitation methods consists of strategies that simply reuse instances selected from a sliding window. In practice, they generate a multiset consisting of indices $\boldsymbol{I} = \{i \mid i \in \langle 1, \omega \rangle\}$, where $|\boldsymbol{I}| = \lambda$. The λ parameter determines the intensity of the exploitation process. A single index is selected using the formula $i = \phi(\omega, r) = \lceil r\omega \rceil$, where r is a random variable $r \sim \mathcal{P}(0, 1)$. As a result, each strategy is modeled by a probability distribution $\mathcal{P}(0,1)$ – it determines on which parts of the window a strategy focuses. We may see the sliding window as a probabilistic or fuzzy one.

Uniform Window (**UW**) – the most straightforward solution is to select the instances uniformly, which results in each instance having an equal chance of being chosen (Fig. 23a). The probability value is given by the uniform distribution, so a single object has $1/\omega$ chance of being used again, where ω is a number of instances within the window after a new object is added. Therefore, the indices i are drawn by the strategy using $r = u \sim \mathcal{U}(0,1)$, where u is a random variable sampled from the uniform distribution $\mathcal{U}(0,1)$. This approach keeps adapting a classifier to a wider context of the maintained batch of data and by doing so it may reduce the risk of falling into local minima. On the other hand, if the sliding window is insufficiently reactive, this method may fail at handling more rapid changes, since, in such case, the strategy will keep providing outdated instances. We assume that this method may work well while dealing with gradual concept drifts, when instances for both older and newer concepts should contribute to updates. The complexity of this method is solely based on the intensity λ, so it is $O(\lambda)$.

Exponential Window (**EW**) – instead of selecting instances with an equal probability, we can focus more on the newer objects, while leaving a non-zero sampling rate for the rest of the window (Fig. 23b). We model this behavior, using the normalized exponential distribution $r = e \sim \mathcal{E}_n(0,1)$. Utilizing a truncated exponential transform, we have $e = -ln(u)/\gamma$, where γ is the parameter of the exponential distribution and $u \sim \mathcal{U}(0,1)$ [211]. This strategy assumes that a sliding window may imperfectly represent the current concepts, e.g, by having an inadequate size or by insufficient exploitation of the newer instances. It attempts to handle this problem on its own by increasing the chances for the more recent data. Theoretically, this approach may be adequate for more rapid incremental changes and, at the same time, acceptable for gradual and sudden concept drifts. We set $\gamma = 4$ as our default value, which provides a functionality balanced between the two other strategies. The complexity is the same as for the previous approach – $O(\lambda)$.

Single Exposition (**SE**) – the most extreme method is reusing only the latest instance $\boldsymbol{w}_\omega$ (Fig. 23c). Obviously, probabilities for such a strategy are: $P[i = \omega] = 1$ and $P[i \neq \omega] = 0$, so formally $\boldsymbol{I} = \bigcup_{\lambda \in \mathbb{N}} \{\omega\}$. One of the advantages of this method is that, in practice, we do not even have to maintain a sliding window. It is enough to update the classifier with the instance several times and proceed to the next incoming objects. The strategy is a fully online one. While it has indisputable advantages regarding processing performance, it is also possible that such intensive focus on a single instance may more likely lead to detrimental overfitting, which is opposite to the problem we want to solve (we spend more time on this aspect in the next sections). On the other hand, it may be a good choice for sudden drifts. The complexity of this strategy is also $O(\lambda)$.



(a) Uniform window (b) Exponential window (c) Single exposition

Fig. 23.: Histograms for selected window indices (x-axis) depending on a strategy.

5.2.1.3 Dynamic parameters

It is easy to notice that two the most important parameters of the strategies are: intensity λ and size of the sliding window ω_{max}. Generally, we may want to keep

λ relatively high for low budgets, low-rate data and unstable temporary concepts to compensate for limited exploration, and low for the opposite, since having an abundance of labeled data and a sufficiently updated model, we can rely on the mechanism less. In addition, we have to control the computation time. When it comes to ω_{max}, we most likely should do the opposite – keep the size of the window low for limited budgets, low-rate data and drifting concepts, and high for more labeled instances and stable streams. These two heuristics should give us more reactivity to changes (plasticity) in the former case and provide us with a more comprehensive and stable generalization (stability) in the latter one. In other words, depending on the situation we may either need to entirely focus on tackling underfitting or to be a bit more careful not to end up with overfitting.

In our work, we empirically analyze how the values of the parameters should change depending on an amount of labeled data, as well as we provide a dynamic control technique for adjusting the values adequately to the state of a stream in a learning process. The idea is that since a model should quickly adapt to new data when it suffers from underfitting or after a concept drift, we can control λ and ω_{max} based on a value of a current error, which should be high in both situations and which is a reliable state-of-the-art input in many drift detectors [111]. Therefore, if $\epsilon \in \langle 0, 1 \rangle$ is a currently registered error for a model learning from a stream, for the intensity we can use:

$$\lambda(\epsilon) = \epsilon \lambda_{max} \tag{5.1}$$

and for the size of a window:

$$\omega'_{max}(\epsilon) = (1 - \epsilon)\omega_{max}, \tag{5.2}$$

where λ_{max} and ω_{max} are fixed maximum values selected for a given budget.

Obviously, a new problem emerges – how should we determine the current value of ϵ? The most straightforward solution is to calculate an error within a sliding window. However, in such a case, we will, again, struggle with finding a proper window size ω_ϵ that should depend on the amount of available data and a state of a stream. Fortunately, this problem has been already addressed by the well-known ADWIN algorithm [108], which provides an adaptive window capable of calibrating its size adequately to incoming data. At its core, the algorithm checks for each subwindow $\boldsymbol{W}_0$ and $\boldsymbol{W}_1$ whether $|\epsilon_{\boldsymbol{W}_0} - \epsilon_{\boldsymbol{W}_1}| > \theta_{cut}$, where θ_{cut} is a significance threshold based on the Hoeffding bound [212] using a specified confidence value α_θ[1]. If the condition holds, the algorithm shrinks the window in order to keep only the most recent consistent values, which are used for the calculation of the estimated average ϵ_{ADW}. We use this indicator as a control signal in our methods, so we have $\epsilon = \epsilon_{ADW}$. Finally, since ADWIN provides the optimal window size in an online way, we can set $\omega_{max} = \omega_{ADW}$. We empirically show that all of those choices are reasoned, also in our scenario involving strict limitations on supervision.

5.2.2 Alleviating overfitting

The presented approach to handling underfitting while learning on a budget has one significant potential weakness – it may turn one problem into its opposite. Since our method prefers aggressive updates using few labeled instances, it becomes more likely that at some point we may encounter the overfitting problem instead of underfitting, even if we try to adequately adjust the dynamic parameters λ and ω_{max} or when we choose a safer exploitation strategy. To alleviate it, we propose

[1]Its actual implementation optimizes the computations, allowing updates with O(1) amortized and O(logW) worst-case time [108]. Therefore, it is a very efficient algorithm for streaming data.

an ensemble of two paired learners, in which one of them performs risky adaptation with instance exploitation (L_r) and another one learns in a standard way, without any additional enhancement (L_s). Such a simple ensemble can learn in two modes.

Switching – in this mode, we maintain both learners in parallel (Alg. 2) and use only the currently better one for prediction, based on their respective errors ϵ_r and ϵ_s (Alg. 3). The assumption is that for some parts of a stream the learner intensively exploiting instances will take the risk in the right direction, providing an efficient up-to-date model more quickly, while for the other parts (mostly more stable) it will fall into local minima, giving way to the basic learner. Since the latter tries to learn in a different way, it is possible that its perspective will allow it to temporarily outperform its counterpart, preventing overfitting.

Elevating – in the alternative mode, we not only temporarily switch between better approaches (Alg. 3), but also elevate the worse model (replacing it with the better one), if a difference between them is significant according to a chosen test $\delta(\epsilon_r, \epsilon_s, \alpha_e)$ (Alg. 4), where α_e determines a significance level of the test. In this case, the idea is that instead of waiting for one model to catch up with learning, we simply instantly bring it up to speed by replacing with a more effective model. One should be careful, however, since having a better model for one timestamp does not mean that it will be easier to adapt it to a new concept from there.

Algorithm 2: Switching learning mode.

```
Input: incoming instance $\mathbf{x}$, true label $y$, risky base learner $L_r$, standard
       base learner $L_s$, ExploitationStrategy
Result: up-to-date models $L_r$, $L_s$
Initialization: $\epsilon_r \leftarrow 0$, $\epsilon_s \leftarrow 0$
update $\epsilon_r$ and $\epsilon_s$;
update $L_r$ with ($\mathbf{x}$, y) using ExploitationStrategy (Algorithm 1);
update $L_s$ with ($\mathbf{x}$, y) ;
```

Algorithm 3: Switching and elevating prediction mode.

```
Data: incoming instance $\mathbf{x}$ risky base learner $L_r$, standard base learner $L_s$
Result: prediction $\hat{y}$
if $\epsilon_r < \epsilon_s$ then
    $\hat{y} \leftarrow L_r(\mathbf{x})$;
else
    $\hat{y} \leftarrow L_s(\mathbf{x})$;
return $\hat{y}$;
```

To test whether two classifiers are significantly different we can use the Welch's test, which assumes that two populations (for ϵ_r and ϵ_s) have unequal variances [213]. We find this assumption reasonable in our case. The statistic t required for the test is obtained using the formula:

$$t = \frac{\epsilon_r - \epsilon_s}{\sqrt{\dfrac{\sigma_r^2}{N_r} + \dfrac{\sigma_s^2}{N_s}}}, \tag{5.3}$$

where required average errors ϵ, their variance σ and the numbers of samples N can be easily calculated within a sliding window, including ADWIN, which we use as a default estimator. The same values are required for the degrees of freedom ν, which can be approximated using the Welch-Satterthwaite equation:

$$\nu \approx \frac{\left(\dfrac{\sigma_r^2}{N_r} + \dfrac{\sigma_s^2}{N_s}\right)^2}{\dfrac{\sigma_r^4}{N_r^2 \nu_r} + \dfrac{\sigma_s^4}{N_s^2 \nu_s}}, \tag{5.4}$$

Algorithm 4: Elevating learning mode.

```
Input: incoming instance $\mathbf{x}$, true label $y$, risky base learner $L_r$, standard
       base learner $L_s$, ExploitationStrategy, significance level $\alpha_e$
Result: up-to-date models $L_r$, $L_s$
Initialization: $\epsilon_r \leftarrow 0$, $\epsilon_s \leftarrow 0$
update $\epsilon_r$ and $\epsilon_s$;
if $\delta(\epsilon_r, \epsilon_s, \alpha_e) = True$ then
    replace the worse model L and error $\epsilon$ with their counterparts;
update $L_r$ and $L_s$ as in Algorithm 2;
```

where $\nu_r = N_r - 1$ and $\nu_s = N_s - 1$ are the degrees of freedom associated with variance estimates. Since the Welch's test can be easily implemented in our setting, we use it as our default test.

It is worth noting that the effectiveness of both modes highly depends on the precision of determining the true value of a current error, especially when supervision is strictly limited. This problem has been already mentioned in Sec. 5.2.1.3 and we empirically investigate it in the context of the proposed ensembles.

5.3 Experimental study

In this section, we present a comprehensive empirical evaluation of the proposed approaches. Our general goal was to provide a complex and, at the same time, in-depth analysis of the strategies and their parameters in different settings. We focused on showing how limiting the labeling budget affects learning from data streams with or without the exploitation wrappers. The specific research questions we asked are given as follows.

- **RQ1:** Does the instance exploitation improve classification while learning from sparsely labeled non-stationary data streams?
- **RQ2:** How the proposed methods should be configured depending on the available labeling budget and stability of a stream?
- **RQ3:** Is the ensemble technique, using the standard and risky classifier, capable of alleviating potential overfitting problems that may occur as a result of using the approach? Does it improve the overall performance?
- **RQ4:** Are our methods competitive when compared with some of the state-of-the-art classifiers dedicated to data streams?

In order to improve reproducibility of this study, the source code for the experiments presented in the following sections, along with information about benchmarks and details of configurations, have been uploaded to our public repository: *github.com/mlrep/ie-20*.

5.3.1 Data streams

For the purpose of the experiments we utilized two groups of data stream benchmarks. Each of them was dedicated to a different goal of this study.

Synthetic streams. The first one consists of streams that were created using artificial drift generators. Such benchmarks are commonly used while evaluating algorithms on dynamic data [111, 116]. Their well-defined characteristics can be utilized to evaluate the algorithms in very specific situations, for example, when adapting to stable or unstable concepts, or to different types of drifts. The streams were created using tools available in MOA [214], which provides some state-of-the-art generators that synthesize drifting streams by simulating transitions between different concepts. They are based on the following sigmoidal formula:

$$f(t) = 1/(1 + e^{-s(t-t_0)}), \tag{5.5}$$

where s controls the duration of change and t_0 is a peak of it. There are several different concepts that can be generated – gaussian clusters (RBF), decision spaces modeled by a random decision tree (TREE), linearly separated subspaces (SEA) or concepts defined by some fixed formulas (STAG). There is also a simulation of a rotating hyperplane (HYPER), which does not use the function given above. We generated 14 synthetic streams (Tab. 7), using different types of concepts and drifts. The latter include very sudden, less or more gradual and very slow changes. For the HYPER concept we used mediocre incremental rotations (defined by the rate param-

Table 7.: Summary of the used synthetic (left) and real (right) data streams.

Name	Inst	Attr	Cls	Drift	Drifts	Noise
RBF1	1m	15	5	100	3	0.05
RBF2	1m	15	5	10k	3	0.05
RBF3	1.2m	15	5	50k	2	0.05
RBF4	1.2m	15	5	100k	2	0.05
TREE1	1m	15	5	100	3	-
TREE2	1m	15	5	10k	3	-
TREE3	1.2m	15	5	50k	2	-
TREE4	1.2m	15	5	100k	2	-
SEA1	600k	3	2	100	3	0.05
SEA2	600k	3	2	10k	3	0.05
STAG1	600k	3	2	100	3	-
STAG2	600k	3	2	10k	3	-
HYPER1	500k	15	5	$\rho = 0.001$	-	0.05
HYPER2	500k	15	5	$\rho = 0.01$	-	0.05

Name	Inst	Attr	Cls
Activity	10 853	43	8
Activity-Raw	1 048 570	3	6
Connect4	67 557	42	3
Cover	581 012	54	7
Crimes	878 049	3	39
DJ30	138 166	8	30
EEG	14 980	14	2
Elec	45 312	8	2
Gas	13 910	128	6
Poker	829 201	10	10
Power	29 929	2	24
Sensor	2 219 804	5	57
Spam	9 324	499	2
Weather	1 8158	8	2

eter ρ). This diversification of benchmarks should provide a reliable generalization of observations.

Due to the deterministic character of the synthetic streams, we decided to use them in the first part of our experiments, in which we thoroughly investigate the influence of different configurations and the potential usability of the presented heuristics.

Real streams. While the artificial data provides a well-defined environment for evaluation, real-world streams may differ from them in some unspecified, unknown aspects. There is still very little research done on the general dynamics of streams, therefore, it is difficult to say how realistic are the concepts and drifts generated by the artificial sources. Due to this reason, it is always important to evaluate streaming algorithms on real examples. In our experiments, we used 14 state-of-the-art real streams that are widely used in data stream mining [117, 116]. Analogously to the synthetic streams, the chosen set represents a variety of different problems with diverse characteristics (Tab. 7). Most of the selected streams are snapshots of some continuous observations registered in industrial systems or laboratories. All of them can be found either in the UCI repository, Kaggle competitions or in popular related publications.

As opposed to the synthetic data, most of the real streams are not transparent when it comes to their internal characteristics. It means that usually we cannot say when exactly a drift occurs or what its specific type is. On the other hand, they provide a more realistic general validation. Because of that, we utilized the real streams in the second part of our experiments, conducting the final comparison between our methods (tuned in the previous step), baseline models without instance exploitation and some state-of-the-art streaming classifiers. By doing so, we also avoided fitting parameters of our algorithms to a specific testing set and fairly evaluated proposed default settings.

5.3.2 Setup

In this subsection, we present details of the setup used in the experiments. For any very specific parameters or subroutines, please, refer to the source code given in the aforementioned repository.

Overview. The experiments were conducted in the following order. First, using the synthetic streams, we evaluated how the exploitation strategies should be configured. We checked how different values of intensity (λ_{max}) and window size (ω_{max}) affect the learning process. We also investigated whether adjusting these values dynamically provides any improvements in terms of performance and utilized resources. In the last part of the tuning stage, we analyzed the behavior of the supporting ensembles, focusing on the correctness of deciding which approach (standard or risky) is currently the better one, depending on the significance threshold (α_e). Finally, we prepared the final comparison between the proposed algorithms, baseline and other well-known classifiers learning in a conservative way. It was conducted using the real streams. Last but not least, since at its core our work emphasizes the problem of limiting the access to supervision, the whole analysis was considered mainly from the perspective of having different labeling budgets (B).

Evaluated parameters. Below, we enclose values of parameters that were considered in our experiments. We evaluated all the values given in the third column. The last one presents other parameters that were predetermined before evaluating the main parameter.

Table 8.: Summary of the parameter values evaluated in the tuning experiments.

Parameter	Description	Values	Other
λ	intensity	{1,10,100,1k}	$\omega_{max} = \omega_{ADW}$, $\epsilon = \epsilon_{ADW}$
ω_{max}	sliding window size	{10,100,1k,10k}	$\lambda = 100$, $\epsilon = \epsilon_{ADW}$
α_e	elevating significance	{0.01,0.05,0.1,0.2}	$\lambda_{AHT} = \{1, 1, 1, 10, 10, 10\}$, $\lambda_{SGD} = 100$ (both dynamic) $\omega_{max} = \omega_{ADW}$, $\epsilon = \epsilon_{ADW}$

While the choice of the default value for ω_{max} and the input error ϵ was somehow justified theoretically and empirically [108], we chose λ either arbitrarily for the initial tests (ω_{max}), or based on empirical observations, for the experiments involving ensembles (α_e), which should use reasonably tuned base learners. Due to significant differences, we decided to set individual λ_{max} for both classifiers. Multiple values in the last column mean that different settings were used for different labeling budgets. In our experiments, we considered $B = \{50\%, 20\%, 10\%, 5\%, 1\%\}$ and $B = 100\%$ where it was necessary, including the final comparison. In addition to the given fixed values, we used $\alpha_\theta = \{0.05, 0.1, 0.1, 0.2, 0.2\}$ and 0.002 for $B = 100\%$ (its default value) in all cases. It is the only parameter that has to be set for ADWIN. For the final comparison we extracted recommended default configurations, based on obtained results. We present them after the tuning phase, at the beginning of the final evaluation.

Classifiers. We used two different base learners: AHT and SGD, in order to show that our framework is, indeed, able to work as a flexible wrapper. Since these classifiers use completely different learning approaches, they seem to be good candidates for providing a more general insight. As our baseline (Base) we used AHT and SGD combined with RandVar (**ALRV**) – the active learning strategy mentioned in Sec. 5.2.1. Due to the fact that we utilize the same algorithm as the default query strategy in our framework, the proposed methods differ from the baseline only by applying the exploitation strategy. During the final comparison, besides the mentioned baseline, we tested two additional variants using other active learning approaches: random query (**ALR**) and selective sampling (**ALS**) [163]. Furthermore, since our methods involve ensemble techniques, we compared them with some of the state-of-the-art committees for streaming data: online bagging (**OBAG**) [118], incremental bagging with ADWIN (**ABAG**), adaptive online boosting (**ADOB**) [215], **DWM** [121], **LNSE** [156], **AUC** [123] (only for AHT due to the implementation constraints) and **AWE** [122] (only for SGD). It is worth noting that the given ensembles represent diverse approaches to adaptation, which, to the best of our knowledge, have not been evaluated in the context of limited supervision. All of the mentioned classifiers used recommended parameter settings given in MOA.

Metrics. Most of our experiments measured the predictive performance of the considered algorithms. For this purpose, we collected kappa values in two ways. The first one involved obtaining averaged global results for whole streams (aggregated predictions), based on the test-then-train procedure [1]. The second approach was focused on registering temporal performance within a sliding window, using the prequential evaluation technique [216]. In the case of synthetic streams, it allowed us to distinguish between stable and drift periods, and to report them separately. It is important to keep in mind that the global average tends to be biased towards stable periods, which are longer in most of the streams. Due to this reason, we also reported the average of the two separated metrics, which is more balanced. Analogously to the discussion made in previous sections, we used the ADWIN-based metrics, as recommended in [216]. In the final phase of the experiments, we employed the Bonferroni-Dunn rank test ($\alpha = 0.05$) to analyze the statistical generalization and the significance of results. Finally, we find more specific metrics used in this study self-explanatory. They are briefly introduced with their presentation.

5.3.3 Results

As we mentioned in the previous sections, the first part of the study presents different configurations and versions of the proposed algorithms, evaluated using fully controlled synthetic streams. It should provide the reader with some intuition about what to expect from those techniques under specific levels of supervision. After that, we proceed to the final evaluation and conclusions that can be drawn based on all the conducted experiments.

5.3.3.1 Intensity

Fixed intensity. The first evaluation focused on studying how intense the instance exploitation should be, meaning, how many labeled instances from the sliding window should be sampled at each step, after receiving a new object.

In Tab. 9 we can see kappa values averaged over all streams for the given budgets and both classifiers. One distinctive observation is that for different classifiers

Table 9.: Average kappa given a budget for AHT and SGD using fixed intensities.

AHT	50%	20%	10%	5%	1%	**SGD**	50%	20%	10%	5%	1%
Base	0.8556	0.8311	0.8103	0.7850	0.6916	Base	0.4089	0.2892	0.2229	0.2018	0.1750
UW-1	0.8680	0.8446	0.8202	0.7944	0.7129	UW-1	0.4734	0.3806	0.2933	0.2245	0.1731
UW-10	**0.8708**	**0.8533**	**0.8333**	0.8115	0.7318	UW-10	0.6047	0.5243	0.4721	0.4088	0.2401
UW-100	0.8571	0.8428	0.8255	**0.8144**	**0.7324**	UW-100	0.6854	0.6557	0.6252	0.5774	0.4350
UW-1k	0.8359	0.8213	0.8022	0.7863	0.7059	UW-1k	**0.7069**	**0.6978**	**0.6826**	**0.6663**	**0.5573**
EW-1	0.8668	0.8461	0.8252	0.8008	0.7226	EW-1	0.4774	0.3840	0.2944	0.2239	0.1812
EW-10	**0.8720**	**0.8567**	**0.8380**	**0.8280**	0.7669	EW-10	0.6078	0.5355	0.4798	0.4139	0.2259
EW-100	0.8555	0.8427	0.8347	0.8191	**0.7753**	EW-100	0.6934	0.6670	0.6372	0.5975	0.4532
EW-1k	0.8445	0.8326	0.8184	0.8009	0.7559	EW-1k	**0.7148**	**0.7037**	**0.6955**	**0.6817**	**0.5943**
SE-1	**0.8698**	**0.8420**	**0.8270**	**0.8040**	**0.7330**	SE-1	0.4791	0.3851	0.3002	0.2335	0.1786
SE-10	0.6763	0.6648	0.6631	0.6532	0.6630	SE-10	0.6143	0.5460	0.4866	0.4195	0.2310
SE-100	0.4128	0.3986	0.3901	0.3853	0.3799	SE-100	**0.6824**	0.6594	0.6341	0.5964	0.4594
SE-1k	0.2438	0.2352	0.2289	0.2110	0.2295	SE-1k	0.6760	**0.6710**	**0.6654**	**0.6537**	**0.6024**

we can observe significantly different preferences. While with AHT majority of the best results were obtained for $\lambda < 100$, SGD worked best with the most intensive exploitation for $\lambda = 1000$. It suggests that, in general, AHT better adapts to streams even under limited supervision, while SGD severely suffers from underfitting, leaving plenty of room for improvements.

Another pattern that the results suggest is that for AHT using UW and EW there is a trend in the relation between required intensity and the available labeling budget. Indeed, if we look closer and analyze Fig. 24, which presents the ratio between our strategies and the baseline, we will notice that as supervision is getting more and more limited, we gain more and more from exploiting the labeled instances we have (from about 1.05 to 1.2 on average). Furthermore, one should notice that for lower budgets below $B = 10\%$, the difference between safer exploitation ($\lambda \leq 10$) and the riskier one ($\lambda \geq 100$) becomes much more significant in favor of the latter approach. Finally, the gain that comes with applying the strategies is higher during

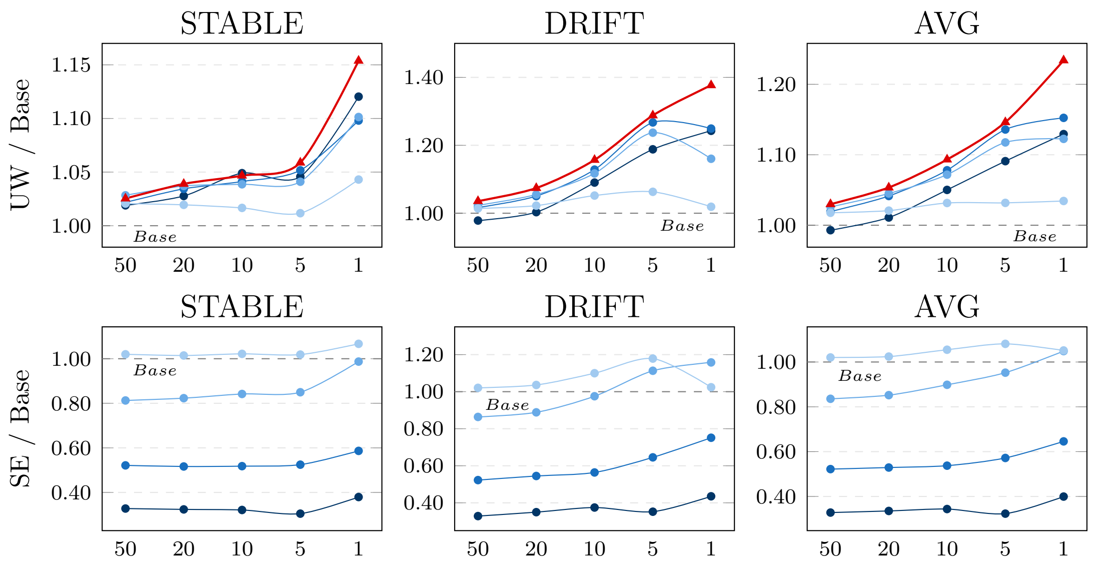


Fig. 24.: Improvement over Base given a budget for AHT using fixed intensities: ● 1k, ● 100, ● 10, ● 1, ▲ best EW.

drifts (1-1.4), when underfitting is more likely to occur regardless of the budget, than during stable periods (1.02-1.15).

One can also notice that the EW strategy (in Fig. 24, the red line represents the best value for a given budget) was slightly better than UW, especially for the lowest budget $B = 1\%$. It is not surprising since in such scenarios even a reactive sliding window may contain some relatively old instances, while it is more important to focus on the strictly limited amount of the most recent information. The EW strategy provides this additional focus.

Interestingly, in the case of AHT, the SE strategy was able to provide some small improvements (1-1.1 on average) only for lower budgets and only with $\lambda = 1$, so when using the newest instance twice. It shows that this strategy may tend to overfit more than UW or EW as it focuses entirely on one instance without replaying a wider context of instances. Nevertheless, even with $\lambda = 1$ the SE strategy provides competitive results for budgets higher than $B = 5\%$, when exploitation becomes less important.

Trends presented in Fig. 25 confirm that SGD requires much more attention that AHT, especially for lower budgets. One can easily see that the baseline using this classifier benefits enormously from employing instance exploitation. Regardless of the strategy used, our wrapper was able to improve the learning process 2-3 times over the baseline. The characteristic curvature of the trends comes from the fact that there is a sweet spot between possible improvements and available supervision. Analogously to the results for AHT, the EW strategy seems to be the best choice.

Dynamic intensity. In our next experiment, we investigated if the proposed heuristic for controlling intensity in a dynamic way – increasing it during drifts and decreasing for stable concepts, based on the current error – may provide any improvements. Tab. 10 presents the average performance of strategies using the dynamic control. We can see that general trends and relations remained the same for both classifiers.

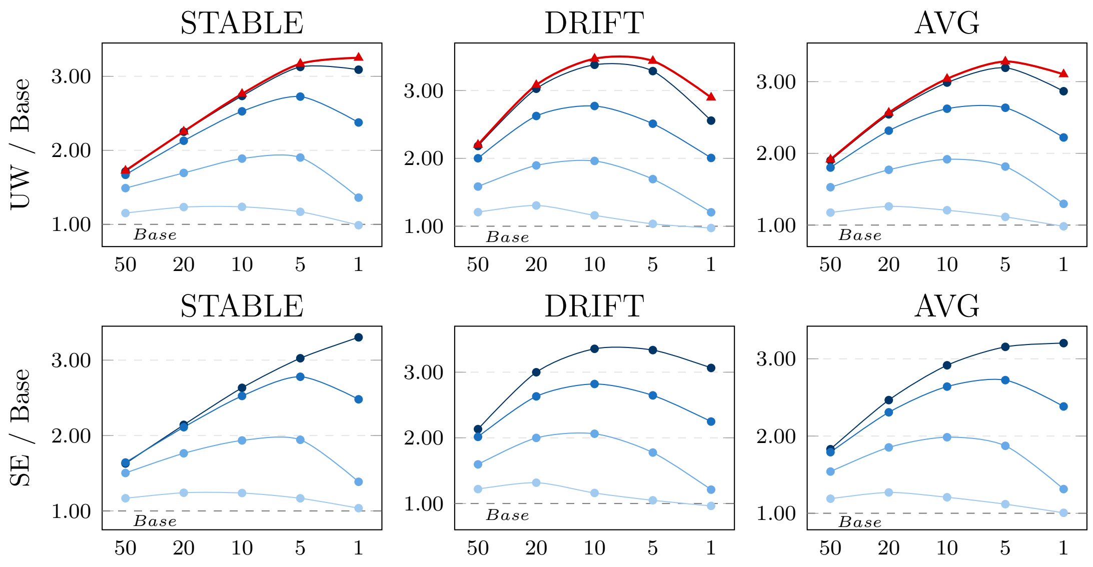


Fig. 25.: Improvement over Base given a budget for SGD using fixed intensities: ● 1k, ● 100, ● 10, ● 1, ▲ best EW.

Table 10.: Average kappa given a budget for AHT and SGD using dynamic intensities.

AHT	50%	20%	10%	5%	1%	**SGD**	50%	20%	10%	5%	1%
Base	0.8556	0.8311	0.8103	0.7850	0.6916	Base	0.4089	0.2892	0.2229	0.2018	0.1750
UW-1	**0.8677**	0.8453	0.8203	0.7966	0.7092	UW-1	0.4731	0.3810	0.2941	0.2339	0.1766
UW-10	0.8763	**0.8566**	**0.8391**	**0.8245**	**0.7500**	UW-10	0.5756	0.4924	0.4186	0.3337	0.1987
UW-100	0.8612	0.8431	0.8312	0.8060	0.7382	UW-100	0.6618	0.6292	0.5865	0.5382	0.3592
UW-1k	0.8362	0.8240	0.8067	0.7831	0.7098	UW-1k	**0.7039**	**0.6888**	**0.6650**	**0.6394**	**0.5385**
EW-1	0.8652	0.8467	0.8230	0.7994	0.7187	EW-1	0.4758	0.3819	0.2924	0.2292	0.1795
EW-10	**0.8774**	**0.8577**	**0.8431**	**0.8225**	0.7596	EW-10	0.5821	0.5020	0.4256	0.3393	0.1985
EW-100	0.8621	0.8461	0.8358	0.8207	**0.7674**	EW-100	0.6748	0.6361	0.5983	0.5547	0.3756
EW-1k	0.8472	0.8320	0.8232	0.8018	0.7467	EW-1k	**0.7101**	**0.6950**	**0.6813**	**0.6601**	**0.5651**
SE-1	**0.8668**	**0.8460**	**0.8263**	**0.8004**	**0.7201**	SE-1	0.4799	0.3816	0.2987	0.2276	0.1779
SE-10	0.7058	0.6969	0.6950	0.6852	0.6825	SE-10	0.5845	0.5059	0.4292	0.3423	0.1980
SE-100	0.4587	0.4418	0.4380	0.4328	0.4347	SE-100	0.6632	0.6309	0.5989	0.5547	0.3830
SE-1k	0.3005	0.2872	0.2777	0.2773	0.2864	SE-1k	**0.6745**	**0.6655**	**0.6544**	**0.6361**	**0.5679**

More useful information gives us Fig. 26 presenting the trade-off between obtained change in performance and computation time for less ($\lambda_{max} = \{1, 10\}$) and more risky ($\lambda_{max} = \{100, 1000\}$) exploitation, compared with its fixed-intensity counterparts. It is clear that those adjustments provide speed-up approximately proportional to the error used as the control signal (Eq. 5.1). In the case of AHT, it is the most significant for the worst-performing risky SE (more than 2.5 times faster) and lower for more reliable strategies and configurations (1.2-1.6). For SGD the speed-up is extremely high (1.5-5.5) due to the generally worse performance of the classifier. We can see that the speed-up for both base learners increases as the budget gets smaller since the performance gets worse on average when we limit supervision.

On the other hand, one should notice that the mentioned improvements of the running time come at some cost of the predictive performance. It is significant for SGD, especially on very low budgets (about 0.9 of the kappa obtained for fixed intensity), but barely noticeable for AHT with $B \leq 10\%$ (0.98 at most) and practically

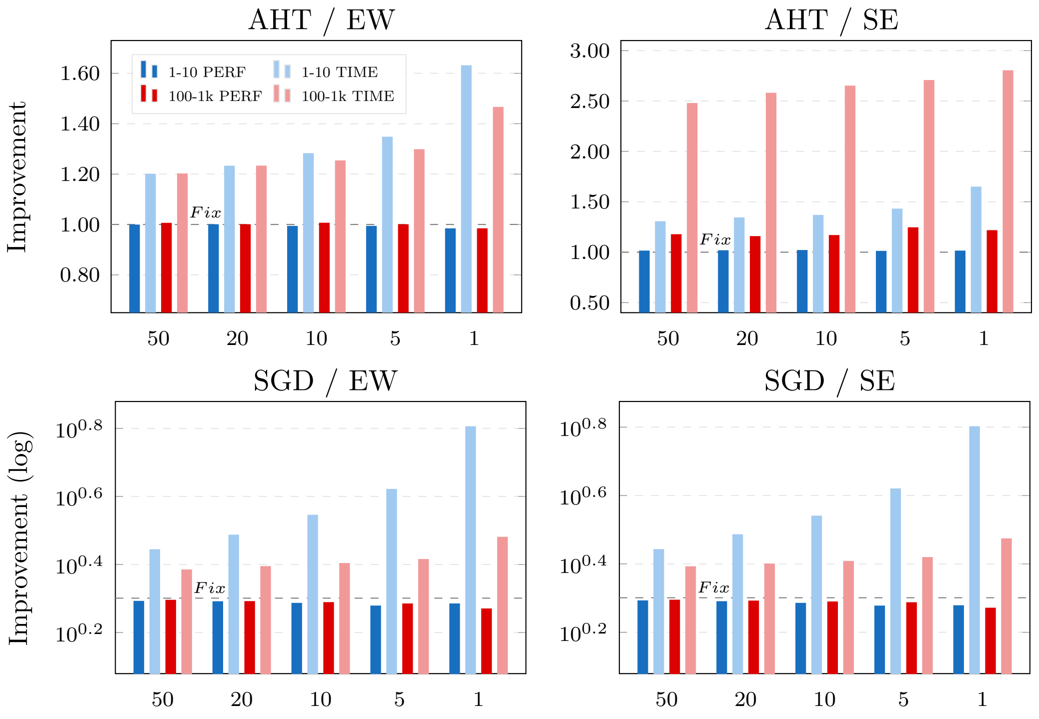


Fig. 26.: Trade-off between performance and time-consumption for AHT and SGD using dynamic intensity (vs. fixed) given a budget.

negligible for higher budgets. In fact, it even provided some improvements for SE, which exhibited the worst performance so far (which we can most likely attribute to overfitting). The explanation of these observations is simple – since the dynamic control may only lower intensity and since SGD gained a substantial amount of enhancement by enabling very intensive instance exploitation, any reduction of it will cause drops in obtained improvements. Due to the fact that AHT is less reliant on our strategies, its adaptation will be impaired to a lesser extent.

5.3.3.2 Instance window

Fixed window size. In this subsection, we investigate how the size of a sliding window of labeled instances should be configured in order to provide necessary reactivity to new concepts and better performance as a consequence. The average results presented in Tab. 11 definitely outline an easily visible trend for both classifiers – the less labeled data we have, the smaller sliding window should be used.

Table 11.: Average kappa given a budget for AHT and SGD using fixed sliding windows.

AHT	50%	20%	10%	5%	1%
UW-10	0.4966	0.4918	0.4887	0.4867	0.4864
UW-100	0.7340	0.7319	0.7286	0.7195	**0.7054**
UW-1k	0.8169	**0.8093**	**0.7991**	**0.7713**	0.6563
UW-10k	**0.8323**	0.7921	0.7220	0.6170	0.5323
EW-10	0.4752	0.4695	0.4641	0.4629	0.4534
EW-100	0.7060	0.7053	0.7040	0.7027	0.6860
EW-1k	0.8142	0.8062	**0.7981**	**0.7818**	**0.7065**
EW-10k	**0.8446**	**0.8230**	0.7786	0.7174	0.5618

SGD	50%	20%	10%	5%	1%
UW-10	0.6871	0.6650	0.6386	0.6018	0.4691
UW-100	0.6929	**0.6706**	**0.6443**	**0.6084**	**0.4697**
UW-1k	**0.6953**	0.6703	0.6400	0.5941	0.4117
UW-10k	0.6780	0.6275	0.5567	0.4614	0.2631
EW-10	0.6857	0.6634	0.6376	0.6022	0.4694
EW-100	0.6918	0.6697	0.6432	**0.6078**	**0.4731**
EW-1k	**0.6952**	**0.6718**	**0.6435**	0.6032	0.4505
EW-10k	0.6878	0.6512	0.6035	0.5349	0.3188

The differences between considered sizes for a given budget are more significant and dynamic for AHT, which is visualized in Fig. 27 showing obtained kappa for both UW and EW. It is clear and intuitive that larger windows ($\omega_{max} = 10000$ and $\omega_{max} = 1000$) are more reliable when we have more labeled data and when concepts are stable, since more labeled instances will better generalize a given problem and help prevent overfitting. Once we limit the available supervision, the rate of changes between subsequent instances is more likely to increase (Sec. 5.2). Therefore, in order to keep the representation of current concepts up-to-date we need to replace older instances with newer ones faster, so the window size should be smaller. For example, we can see that while $\omega_{max} = 10000$ is the optimal size if we can label half of a whole stream, it becomes completely useless once we limit the budget to $B = 1\%$, when even $\omega_{max} = 1000$ becomes less adequate than $\omega_{max} = 100$. An observation that the size has to be even smaller for drifting concepts is not surprising – the problem

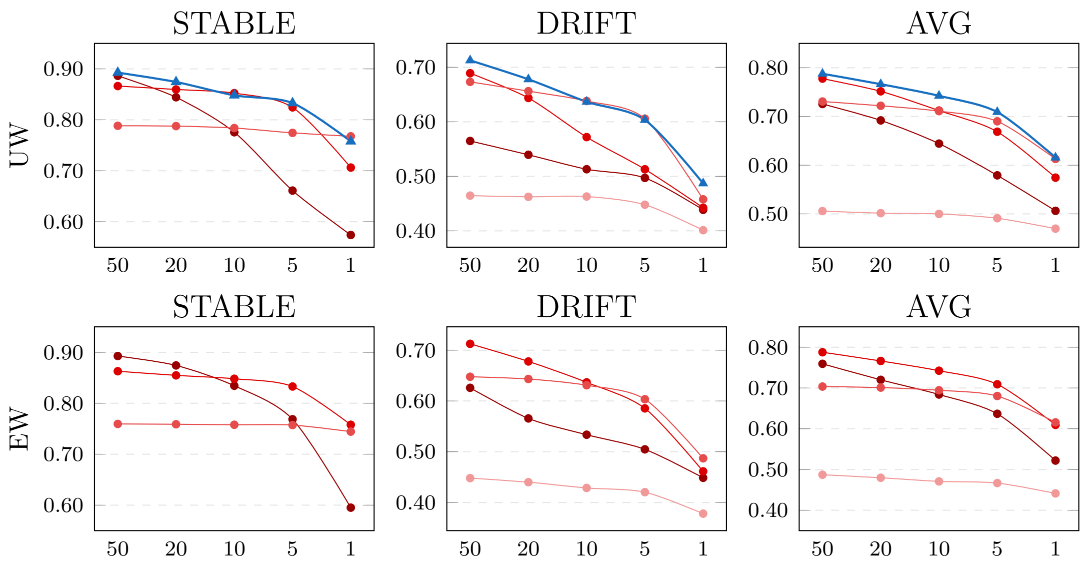


Fig. 27.: Average kappa given a budget for AHT using fixed sliding windows: ● 10k, ● 1k, ● 100, ● 10, ▲ best EW.

of developing reactive windows for drifting data has been already covered in many related publications [108].

In the case of SGD, the differences between window sizes are slightly smaller (Fig. 28), since the overall performance of this classifier is predominantly reliant on a proper choice of the level of intensity. Also, it seems that SGD works best with $\omega_{max} \leq 100$, which suggests that it prefers intensive updates based on the most recent data.

Finally, for both classifiers, the EW strategy was able to reduce the negative impact of too large window sizes for smaller budgets – one can notice that the curves for $\omega_{max} = 10000$ and $\omega_{max} = 1000$ are slightly elevated compared with UW. The reason for that is the inherent focus of EW on the most recent instances, alleviating the problem of storing too old instances, which we assumed in Sec. 5.2.1.2. Regardless of that, we did not find any significant difference between EW and UW in this experiment, which suggest that intensity is a more impactful factor.

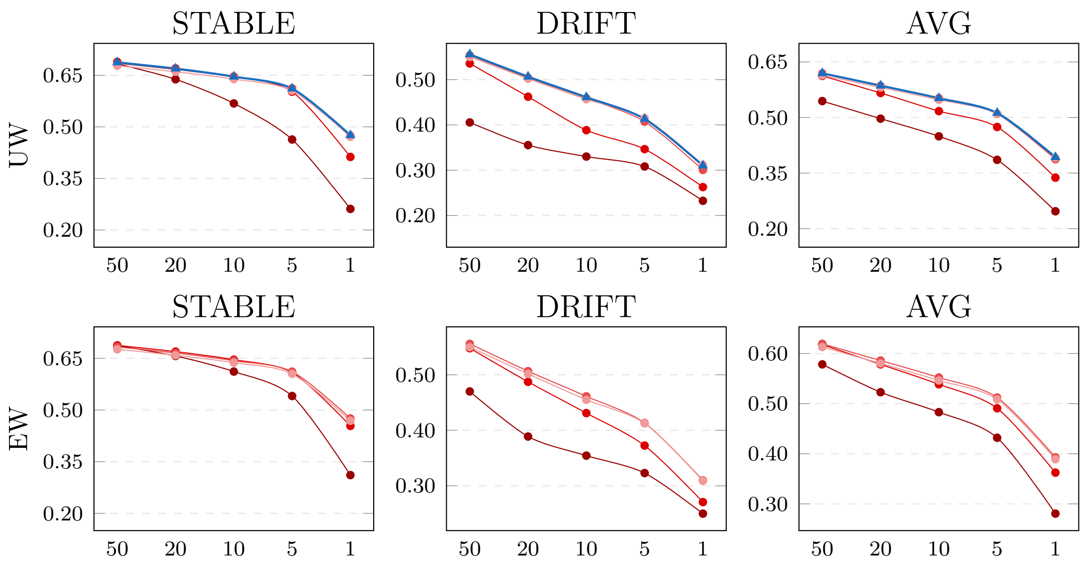


Fig. 28.: Average kappa given a budget for SGD using fixed sliding windows: ● 10k, ● 1k, ● 100, ● 10, ▲ best EW.

Dynamic window size. Analogously to the experiments focused on intensity, we studied the impact of controlling the window size in a dynamic way. In Tab. 12 one can see that while the general trends and relations remain, again, unchanged, using the size returned by ADWIN is definitely the most reliable solution for AHT, providing from 0.02 to more than 0.15 higher kappa values than any other window. For SGD differences between smaller sizes and ADWIN are hardly significant for $B \geq 20\%$ and in favor of the former for $B \leq 10\%$.

The trade-off between the quality of classification and memory consumption is presented in Fig. 29. The results for EW (UW exhibits analogous characteristics) show that while combining the dynamic adjustments with smaller windows we can use less memory (about 1.4-1.7 times less) at a cost of impaired performance (about 0.94), doing the same with larger windows results in improving both (about 1.1-1.7 for memory and 1.01-1.05 for kappa). It is intuitive, since the shrinking heuristic (Eq. 5.2), by making the small windows (containing 10-100 instances) even smaller, will

Table 12.: Average kappa given a budget for AHT and SGD using dynamic sliding windows.

AHT	50%	20%	10%	5%	1%
UW-10	0.4439	0.4295	0.4350	0.4281	0.4263
UW-100	0.7161	0.7128	0.7102	0.7068	0.6896
UW-1k	0.8161	0.8104	0.7988	0.7845	0.6976
UW-10k	0.8416	0.8078	0.7651	0.6896	0.5585
UW-ADW	**0.8603**	**0.8452**	**0.8243**	**0.8098**	**0.7418**
EW-10	0.4424	0.4252	0.4235	0.4259	0.4250
EW-100	0.6798	0.6779	0.6778	0.6748	0.6730
EW-1k	0.8081	0.8062	0.7985	0.7919	0.7356
EW-10k	0.8502	0.8306	0.8068	0.7557	0.6209
EW-ADW	**0.8598**	**0.8483**	**0.8312**	**0.8202**	**0.7717**

SGD	50%	20%	10%	5%	1%
UW-10	0.6864	0.6642	0.6395	0.6023	0.4664
UW-100	0.6940	0.6717	**0.6459**	**0.6088**	**0.4712**
UW-1k	**0.6966**	**0.6730**	0.6451	0.6041	0.4467
UW-10k	0.6831	0.6418	0.5887	0.5154	0.3180
UW-ADW	0.6892	0.6568	0.6275	0.5799	0.4379
EW-10	0.6851	0.6631	0.6373	0.6003	0.4679
EW-100	0.6918	0.6694	0.6445	0.6065	**0.4731**
EW-1k	**0.6966**	**0.6731**	**0.6474**	**0.6072**	0.4643
EW-10k	0.6908	0.6594	0.6208	0.5631	0.3823
EW-ADW	0.6941	0.6682	0.6380	0.5968	0.4572

more likely increase the chance of overfitting than efficiently improve reactivity. On the other hand, making too large windows more flexible provides enhanced reactivity to new concepts without too significant loss in generalization, especially for smaller budgets $B \leq 10\%$ for which concepts evolve at a higher rate.

When it comes to using a window size based on ADWIN, besides the good predictive performance, we also observed that this approach tends to aggregate relatively large amounts of instances, especially for high budgets. Since there is no fixed counterpart for ADWIN, in Fig. 29 we present a ratio between ADWIN and a standard window storing a comparable number of instances, which was $\omega_{max} = 10000$. Based on that, we can see that the ADWIN-driven control provides the presented performance at a quite relevant memory cost, which decreases with the budget. It is also important that even if ADWIN stores plenty of instances, it maintains a good quality of classification regardless of the number of labeled instances, doing it better than $\omega_{max} = 10000$ and $\omega_{max} = 1000$ for AHT and competitively for SGD.

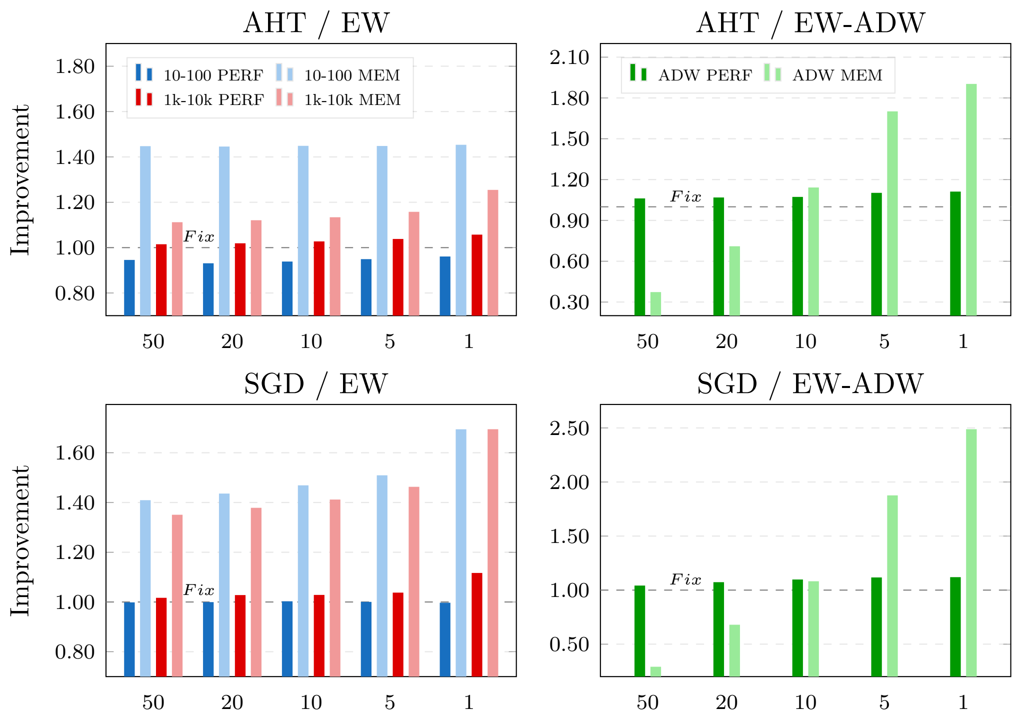


Fig. 29.: Trade-off between performance and memory given a budget for AHT and SGD using dynamic windows (vs. fixed).

5.3.3.3 Elevating significance

In the final section of the first part of the experiments, we analyze the significance level α_e, which is the only parameter that has to be set for the elevating strategy. Simultaneously, we also use this section to investigate the potential usefulness of the proposed ensemble techniques. All results for the elevating ensemble (EWE, SEE) using different α_e along with results for the switching committee (EWS, SES), baseline and single-classifier exploitation methods (EW, SE) are presented in Tab. 13 . Since EW performed at least as well as UW in all previous evaluations, we decided to omit it in the rest of the study.

The presented results clearly indicate that there is no significant difference when changing the α_e value. Furthermore, it seems that for synthetic streams the ensemble methods were not able to provide significant improvements over tuned EW or SE for both classifiers. The only exception is for SE for which we purposely set nonoptimal intensity $\lambda_{max} = 10$, slightly higher than the previously obtained results suggested. This allowed us to check whether the ensembles are able to surpass the ineffective exploitation strategy by using the alternative standard base learner when it is needed. In Fig. 30 we can see that, indeed, improperly configured SE failed for budgets $B \leq 10\%$, performing worse than the baseline for stable concepts by leading to overfitting. However, when combined with switching or elevating, the whole method was able to achieve performance at least as good as the baseline, or even to improve upon it for $B \leq 5\%$. In addition, the ensembles boosted adaptation during concept drifts (by about 0.1), which is a very important observation, since dealing with changes under strictly limited supervision is an extremely challenging task.

The elevation of results obtained for SE is a good indicator that our ensembles should be able to increase the lower bound of our approach, by providing that it will

Table 13.: Average kappa given a budget for AHT and SGD ensembles compared with Base and single-classifier models.

AHT	100%	50%	20%	10%	5%	1%	**SGD**	100%	50%	20%	10%	5%	1%
Base	0.8673	0.8556	0.8311	0.8103	0.7850	0.6916	Base	0.4772	0.4089	0.2892	0.2229	0.2018	0.1750
EW	0.8779	0.8636	0.8425	0.8402	0.8226	0.7560	EW	0.7026	0.6752	0.6424	0.6020	0.5621	0.3707
EWS	0.8798	0.8691	0.8457	0.8478	0.8318	0.7612	EWS	0.7027	0.6723	0.6415	0.6005	0.5508	0.3779
EWE-1	0.8795	0.8704	0.8451	0.8523	0.8301	0.7604	EWE-1	0.7030	0.6720	0.6365	0.5980	0.5536	0.3788
EWE-5	0.8787	0.8677	0.8440	0.8462	0.8272	0.7611	EWE-5	0.7028	0.6722	0.6401	0.5980	0.5530	0.3767
EWE-10	0.8793	0.8706	0.8462	0.8463	0.8323	0.7599	EWE-10	0.7025	0.6730	0.6389	0.5985	0.5503	0.3761
EWE-20	0.8796	0.8697	0.8449	0.8527	0.8286	0.7556	EWE-20	0.7023	0.6700	0.6371	0.5926	0.5532	0.3793
SE	0.8768	0.8695	0.8459	0.6914	0.6856	0.6800	SE	0.6913	0.6632	0.6311	0.5987	0.5543	0.3813
SES	0.8800	0.8698	0.8470	0.8136	0.7919	0.7394	SES	0.6953	0.6643	0.6311	0.5994	0.5551	0.3810
SEE-1	0.8817	0.8737	0.8457	0.8189	0.7983	0.7472	SEE-1	0.7001	0.6687	0.6325	0.6020	0.5576	0.3799
SEE-5	0.8819	0.8697	0.8424	0.8174	0.8022	0.7485	SEE-5	0.6995	0.6697	0.6385	0.6008	0.5564	0.3824
SEE-10	0.8782	0.8687	0.8441	0.8238	0.7966	0.7573	SEE-10	0.6994	0.6692	0.6309	0.6006	0.5566	0.3846
SEE-20	0.8808	0.8692	0.8450	0.8205	0.8018	0.7480	SEE-20	0.6986	0.6695	0.6320	0.5977	0.5564	0.3810

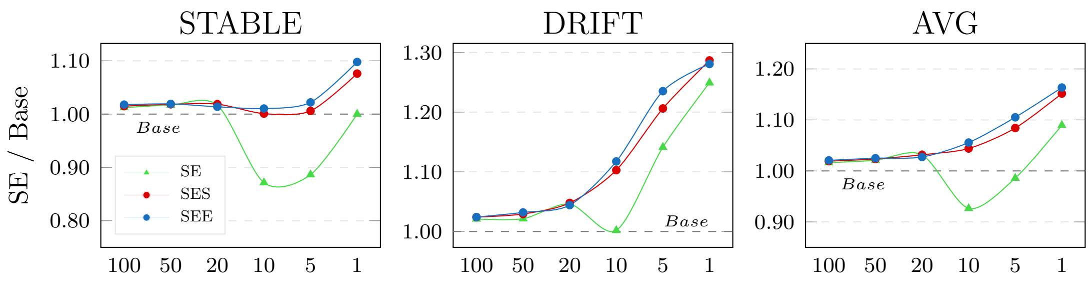


Fig. 30.: Improvement over Base given a budget for AHT using SE and the SE-based ensembles.

never be worse than the baseline. After looking at the results in Fig. 31 we can understand that this is not a coincidence. The bar plots present how many times on average either the standard base learner (Stand-TP, Stand-FP) or the risky one (Risky-TP, Risky-FP) was correctly (true positive, TP) or incorrectly (false positive, FP) elevated with respect to the available budget and used significance level α_e. The correctness depended on the precision of the error estimated based on the partial information from labeled instances. We can easily notice that a prevalent number of elevations was done correctly. The only configuration that stands out with a visible number of false positives is the one using $\alpha_e = 0.2$, which is, in fact, an unusual value for the Welch's test. Nevertheless, it still did not affect performance in a meaningful way. The results prove that we can effectively track an error even under strictly limited supervision and utilize it in switching or elevating techniques.

A more careful analysis of the average number of elevations gives us some additional observations. Firstly, the total number of elevations decreases with the budget and significance level α_e, which is intuitive. Secondly, in most of the cases, if an exploitation strategy is properly configured, we will be replacing mainly the standard learner with the risky one. This can be mostly seen for EW, however, we would observe the same relation also for SE, if it was properly configured for AHT, or for SGD

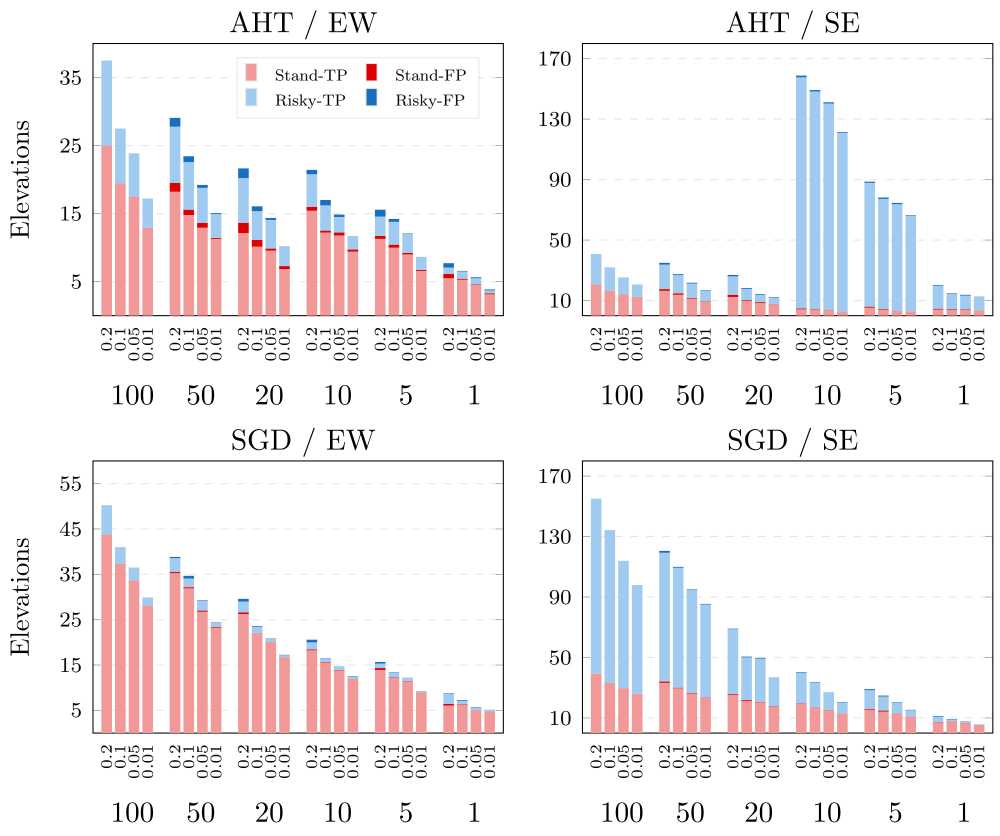


Fig. 31.: Number of elevations given a budget and significance level for ATH and SGD using elevating ensembles.

with the exclusion of the SEA stream. This was the only benchmark for which the SE strategy was constantly failing when combined with the latter classifier, resulting in more than 500 elevations and biasing the average value. We could use median instead, however, we decided to keep the average to show that the SE strategy tends to be more sensitive to improper configuration and overfitting than EW or UW, taking into consideration results for both AHT and SGD. Finally, one may wonder why even if we replace EW or SE with a standard learner multiple times, we do not obtain significant improvements over them (except for the specified cases for AHT with SE)? The reason may be that even if we elevate the exploiting learner it keeps falling into the same pitfalls again and again, as the internal characteristics of the streams do not allow any more improvements than the strategies give themselves in other parts of the streams. For a similar reason, we cannot see additional enhancements when using switching. Fortunately, the differences are more visible for real data streams, in favor of the ensembles, which makes them more than just safe lower bound providers.

5.3.3.4 Final comparison

Default configurations. Based on the observations made in the previous sections, we determined default settings for our strategies and used them in the final comparison, which was conducted using real streams. Since we found trade-offs obtained for the dynamic heuristics fair, we decided to use dynamic intensity controlled by the ADWIN error ($\epsilon = \epsilon_{ADW}$), as well as dynamic window size determined by the same algorithm ($\omega_{max} = \omega_{ADW}$). We used the same significance levels α_θ as for synthetic streams. For our elevating ensembles, due to the lack of impact, we chose one of the widely used values, $\alpha_e = 0.05$. Finally, in order to find a compromise between evidence suggesting lower values of intensity for AHT and making sure that we fully utilize the potential of our strategies, we distinguished two groups of λ_{max} values: less risky λ^l_{max} and risky λ^r_{max}. In addition to that, since AHT and SGD exhibited substantially different preferences regarding required intensity, we set: $\lambda^l_{max} = \{1, 1, 1, 1, 1, 10\}$, $\lambda^r_{max} = \{100, 100, 100, 1000, 1000, 1000\}$ for AHT, where each value corresponds to a budget ranging from $B = 100\%$ to $B = 1\%$, respectively, and $\lambda^l_{max} = 10$, $\lambda^r_{max} = 1000$ for SGD.

Results. The average performance of all considered classifiers under varying budgets is presented in Tab. 14. It provides a general overview of the final comparison. The main observation is that the proposed instance exploitation strategies are capable

Table 14.: Final average kappa for AHT and SGD on real data streams.

AHT	100%	50%	20%	10%	5%	1%	**SGD**	100%	50%	20%	10%	5%	1%
EW^l	0.5637	0.5275	0.4611	0.4116	0.3721	0.3040	EW^l	0.3397	0.3171	0.2685	0.2043	0.1568	0.0764
EWS^l	0.5831	0.5387	0.4765	0.4108	0.3880	0.3306	EWS^l	0.3751	0.3519	0.3073	0.2706	0.2357	0.1345
EWE^l	0.5638	0.5246	0.4707	0.4177	0.3780	0.3326	EWE^l	0.3684	0.3429	0.3021	0.2681	0.2304	0.1164
SE^l	0.5752	0.5378	0.4778	0.4362	0.3953	0.3401	SE^l	0.3954	0.3760	0.3363	0.3044	0.2819	0.1743
SES^l	0.5914	0.5516	0.4990	0.4463	0.4035	0.3503	SES^l	0.3956	0.3707	0.3355	0.3045	0.2804	0.1679
SEE^l	0.5838	0.5345	0.4859	0.4202	0.3842	**0.3578**	SEE^l	0.3960	0.3710	0.3410	0.3073	0.2812	0.1647
EW^r	0.6432	0.5959	0.5520	0.5035	0.4521	0.3315	EW^r	0.4124	0.3962	0.3533	0.3027	0.2711	0.1753
EWS^r	0.6492	0.6128	**0.5634**	**0.5161**	0.4525	0.3287	EWS^r	0.4351	0.4188	0.3879	0.3600	0.3184	0.2344
EWE^r	**0.6569**	**0.6182**	0.5620	0.5099	**0.4690**	0.3405	EWE^r	0.4379	0.4208	0.3878	0.3576	0.3176	0.2114
SE^r	0.5655	0.5404	0.4915	0.3900	0.3530	0.2541	SE^r	**0.4714**	**0.4455**	**0.4109**	**0.3813**	**0.3634**	**0.2891**
SES^r	0.6424	0.6052	0.5594	0.4984	0.4684	0.3500	SES^r	0.4698	0.4421	0.4042	0.3730	0.3469	0.2664
SEE^r	0.6416	0.6044	0.5557	0.5025	0.4573	0.3473	SEE^r	0.4683	0.4414	0.4044	0.3742	0.3411	0.2679
ALR	0.5403	0.4978	0.4475	0.3946	0.3527	0.2997	ALR	0.3440	0.3098	0.2644	0.2195	0.1865	0.1054
ALRV	0.5063	0.5036	0.4466	0.3953	0.3537	0.3030	ALRV	0.3151	0.3156	0.2653	0.2215	0.1863	0.1228
ALS	0.5242	0.5082	0.4424	0.3858	0.3496	0.2986	ALS	0.3204	0.3135	0.2669	0.2199	0.1800	0.1158
AUC	0.4297	0.4218	0.3680	0.2990	0.2656	0.2324	AWE	0.1289	0.1062	0.0918	0.0559	0.0427	0.0298
DWM	0.6154	0.5897	0.5309	0.4993	0.4501	0.3335	DWM	0.3689	0.3375	0.2861	0.2593	0.2369	0.1404
LNSE	0.2734	0.2986	0.3096	0.3000	0.2237	0.1717	LNSE	0.1242	0.1325	0.1163	0.0843	0.0456	0.0210
ADOB	0.2588	0.2564	0.2608	0.2705	0.2562	0.2464	ADOB	0.4553	0.4347	0.4050	0.3632	0.3271	0.2299
ABAG	0.6204	0.6003	0.5414	0.4918	0.4455	0.3203	ABAG	0.3673	0.3444	0.2873	0.2427	0.2169	0.1381
OBAG	0.5752	0.5438	0.4711	0.4129	0.3805	0.2964	OBAG	0.3343	0.3056	0.2553	0.2147	0.1845	0.1239

of providing the best predictive quality (in bold) out of all models and that the risky ones exhibited much better quality than the more conservative configurations. Most importantly, our best strategies were able to outperform baseline models using only active learning (ALR, ALRV, ALS) for both base learners (by about 0.1 kappa for AHT and 0.15 for SGD), regardless of the available budget. It shows that the depicted problem of underfitting while learning temporary drifting concepts under limited supervision is critical in real scenarios and active learning fails to solve it efficiently on its own.

In Fig. 32 we can see a gain obtained from using our exploitation strategies compared with the best baseline model using only active learning (AL) for each budget. It provides more insight into trends and relations present in the aggregated results. Firstly, as we already mentioned, the more risky approaches, applying more intense exploitation of labeled instances, outperform the safer configurations in almost all cases. We can see that EW^r and EWS^r provided about 0.1-0.2 improvement in gain over EW^l and EWS^l for both AHT and SGD. In the case of SE and SES, the enhancement remained the same for SGD and for AHT using SES. These relations are analogous for EWE and SEE. Interestingly, for AHT with SE^r we noticed that while for lower budgets the base learner did not work well with this extreme strategy, the switching technique SES^r empowered the strategy to become more efficient even than SE^l, which was initially better than SE^r. This shows that taking even a very risky adaptation while having a good lower-bound backup may still be beneficial. On the other hand, results for AHT using SE^r and SE^l indicate that we may need to be a bit more careful with more reactive classifiers. Statistical rank tests presented in Fig. 33 prove that the advantage of the more risky exploitation is significant, as the best risky strategies for AHT (EWS^r, EWE^r) and SGD (SE^r, SES^r, SEE^r) are significantly better than any less risky method.

Secondly, Fig. 32 shows that, generally and analogously to the results for the synthetic streams, the improvement over the baseline models increases as we limit available supervision. It is, again, intuitively correct, since if we have fewer labeled instances, the risk of encountering underfitting becomes higher. The only exception for this trend can be noticed when using the extremely limited budget $B = 1\%$ and the reason for that is the same as the one given for the characteristic curve for SGD in Fig. 25. The increase is more clear for SGD, which, in general, adapts less efficiently than AHT, giving more occasions for improvements. Regardless of the budget, almost all strategies turned out to be significantly better than a model

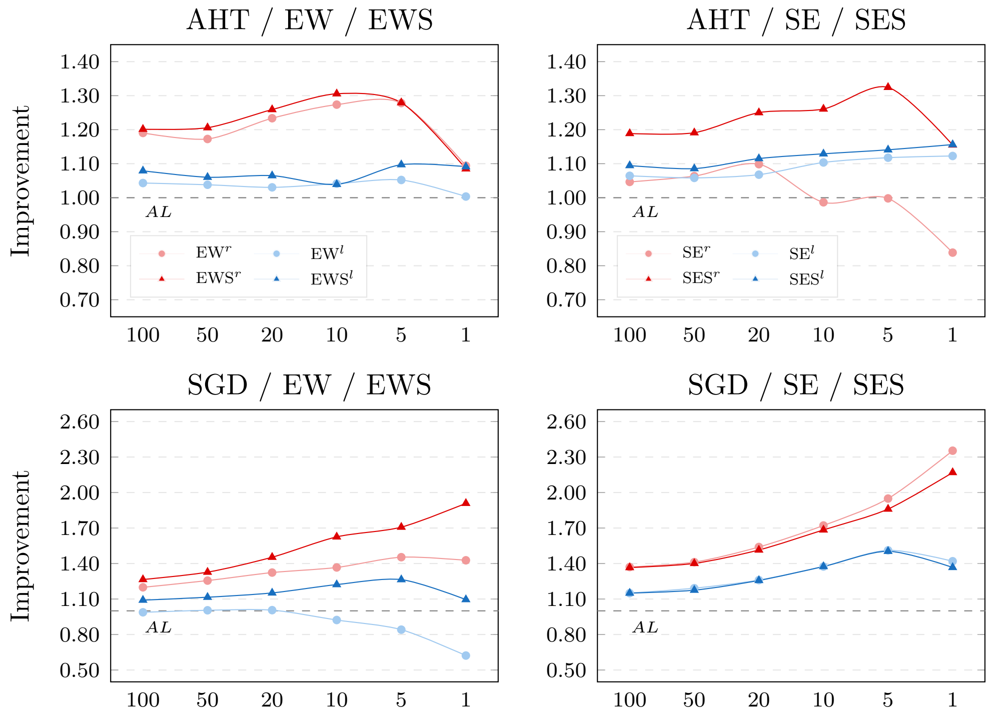


Fig. 32.: Improvements over the best AL given a budget for AHT and SGD using different strategies on real data streams.

without instance exploitation (Fig. 33, ALRV, in bold), which proves the general usefulness of the proposed approaches. Interestingly, since our methods were able to provide some improvements also for fully labeled streams ($B = 100\%$), we can assume that it may enhance adaptation to unstable data in general – limiting budget only makes the problem harder and more severe.

Finally, based on Fig. 32 and Fig. 33 we can claim that switching and elevating ensembles provide intended improvements. In almost all cases they maintained at least as good performance as the best base learner, either a single-classifier strategy (SE, EW) or a standard model without exploitation (ALRV). As a consequence, the ensembles are always at least as good as the baseline, providing noticeable improvements in almost all cases, as opposed to simpler strategies (SE^r for AHT and EW^l for

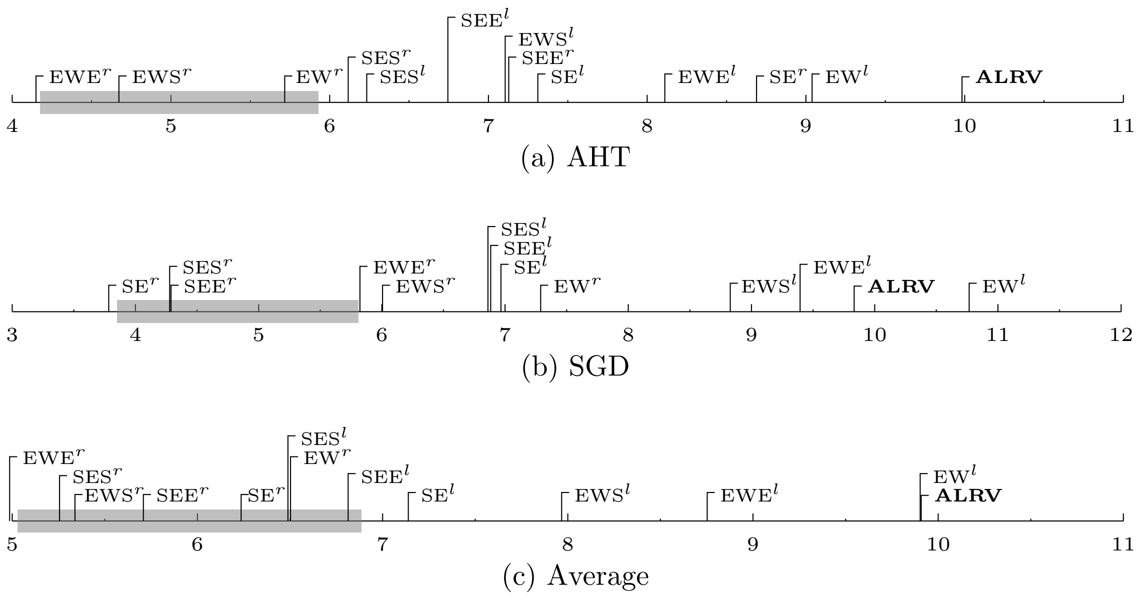
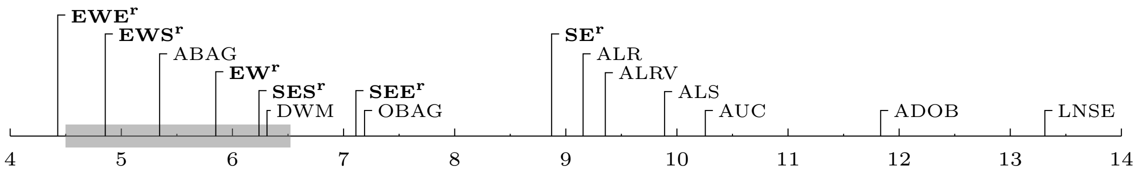


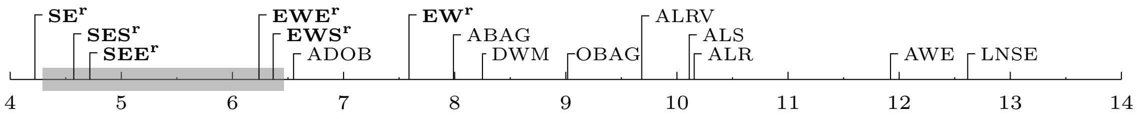
Fig. 33.: Bonferroni-Dunn test for different strategies on real data streams.

SGD). Fig. 33 suggests that the improvements over the single-classifier methods are usually on the verge of significance. The only exception is, one more time, SE^r for SGD – since it worked outstandingly well with this classifier (especially for the lowest budgets), there was a low chance that the baseline could provide better prediction and, as a consequence, any incorrect switch led to some reduction in improvements. This once again shows that even very extreme and risky exploitation makes sense. In addition, compared with results obtained for synthetic streams, the ensembles were also able to improve upon SE and EW in some cases, even if the simpler strategies already provided a gain over the baseline. This very likely shows that real streams are more unstable or complex than synthetic ones, thus requiring more intensive adaptation from classifiers. Last but not least, we did not obtain a significant difference between the switching and elevating techniques.

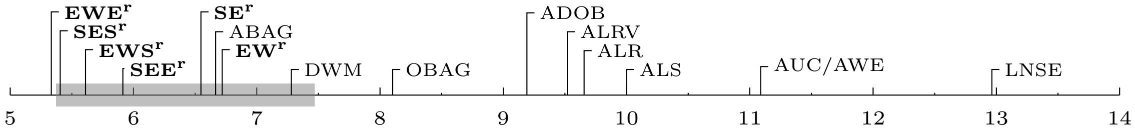
As the final part of our experimental study, we compared our best strategies (the risky ones) with some state-of-the-art classifiers, including models supported by different active learning strategies and popular ensembles. Based on the results in Tab. 14 and the statistical tests in Fig. 34, we can claim that our best strategies (in bold in the latter) are capable of significantly improving upon all baseline models using only active learning and upon most of the considered ensembles. We can distinguish EWE^r, EWS^r and SES^r methods. They outperformed all other classifiers on average, except for ABAG and DWM, which were significantly competitive, especially for AHT, even under strictly limited supervision. It exhibits their generally good resilience to such limitations. The EW-based ensembles provided the best results with AHT, while the SE-based ones dominated the highest ranks with SGD. This shows that our ensembles are able not only to improve upon single-classifier models without instance exploitation but also to compete with state-of-the-art ensembles designed for data streams. One should also keep in mind, that while all the other committees use at least 10 base learners, our ensemble utilizes only two of them.



(a) AHT



(b) SGD



(c) Average

Fig. 34.: Bonferroni-Dunn test for different strategies and other classifiers on real data streams.

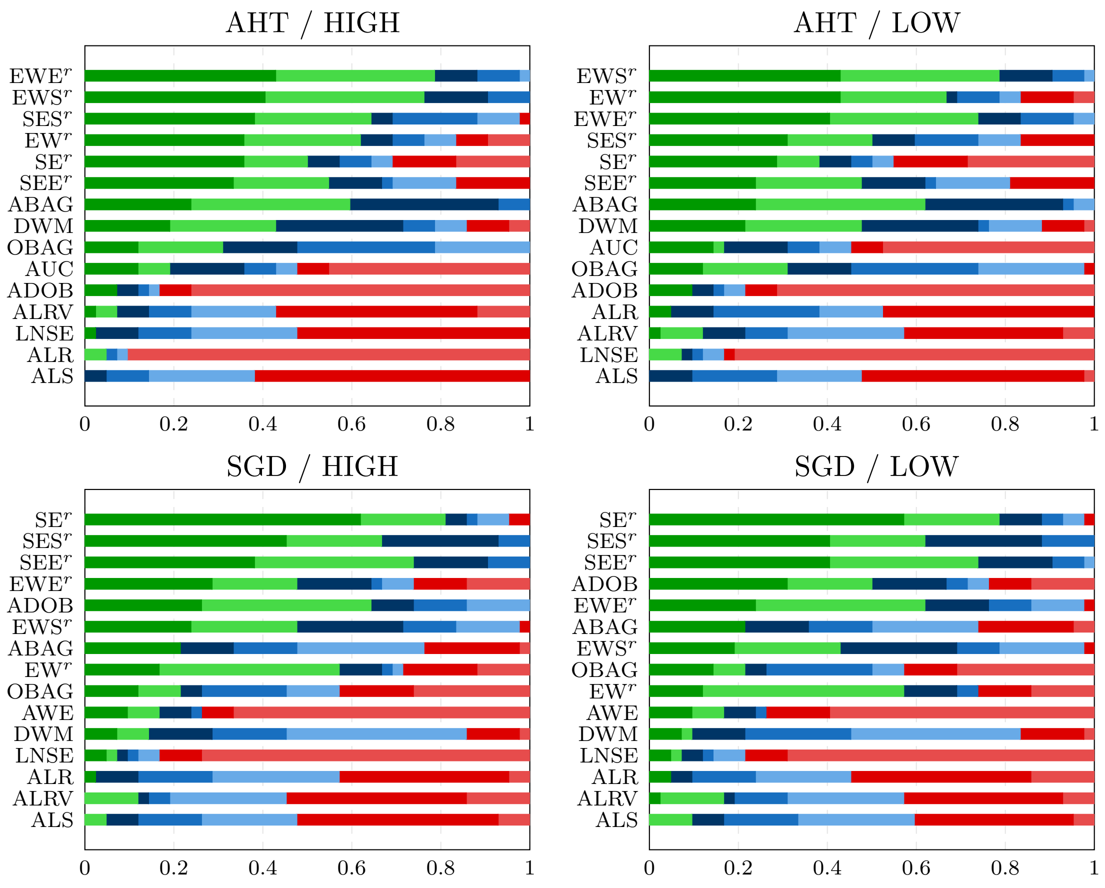


Fig. 35.: Ranks for AHT and SGD using different strategies compared with other classifiers on real data streams.

The complementary results presented in Fig. 35 show the frequencies of occupied ranks for each of the considered methods given high ($B \geq 20\%$) and low budgets ($B \leq 10\%$). Unsurprisingly, we can see that the best strategies were most frequently the best ones (green bars), while very rarely ending up as the worst ones (red bars), regardless of the available supervision. One interesting observation is that our ensembles were able to maintain safer lower bounds also for the presented ranks – they very efficiently reduced the number of lowest ranks compared with their simpler counterparts (SE^r, EW^r), bringing them down to zero for the best strategies (EWS^r and

EWE^r for AHT, SES^r and SEE^r for SGD). It is important if someone cares about the worst possible scenario.

In order to provide the reader with some specific examples of how our methods work in practice, in Fig.36 and Fig.37 we presented accuracy series (instead of kappa for readability) registered for all real streams when the budget was reasonably limited to $B = 10\%$. The best exploiting strategy, the best baseline using active learning and the best other ensemble were chosen for each stream separately. For most of the streams, we can easily notice that instance exploitation (green) was able to elevate the temporal performance of standard models that were reliant solely on active learning (red). The improvements may have diverse forms. For example, our strategies were able to provide overall a more stable, saturated and higher level of predictive performance for Activity-Raw, Covertype, EGG, Sensor or Weather, among others, for both AHT and SGD. For other streams, the enhanced adaption meant that instance exploitation was able to alleviate severe temporal drops in accuracy, most likely due to concept drifts, for example, for Gas, Poker or Spam. There were very few exceptions when none of our strategies was able to provide improvements, like Airlines and DJ30, or Crimes for SGD. Finally, when compared with the considered state-of-the-art ensembles (blue), we can also see that our strategies were, in most cases, competitive in improving adaptation in the same way as against the baseline models.

5.3.4 Lessons learned

To summarize all the presented results and observations, we finalize our study with a list of the most important conclusions. Primarily, they address the research questions introduced at the beginning of the experimental study.

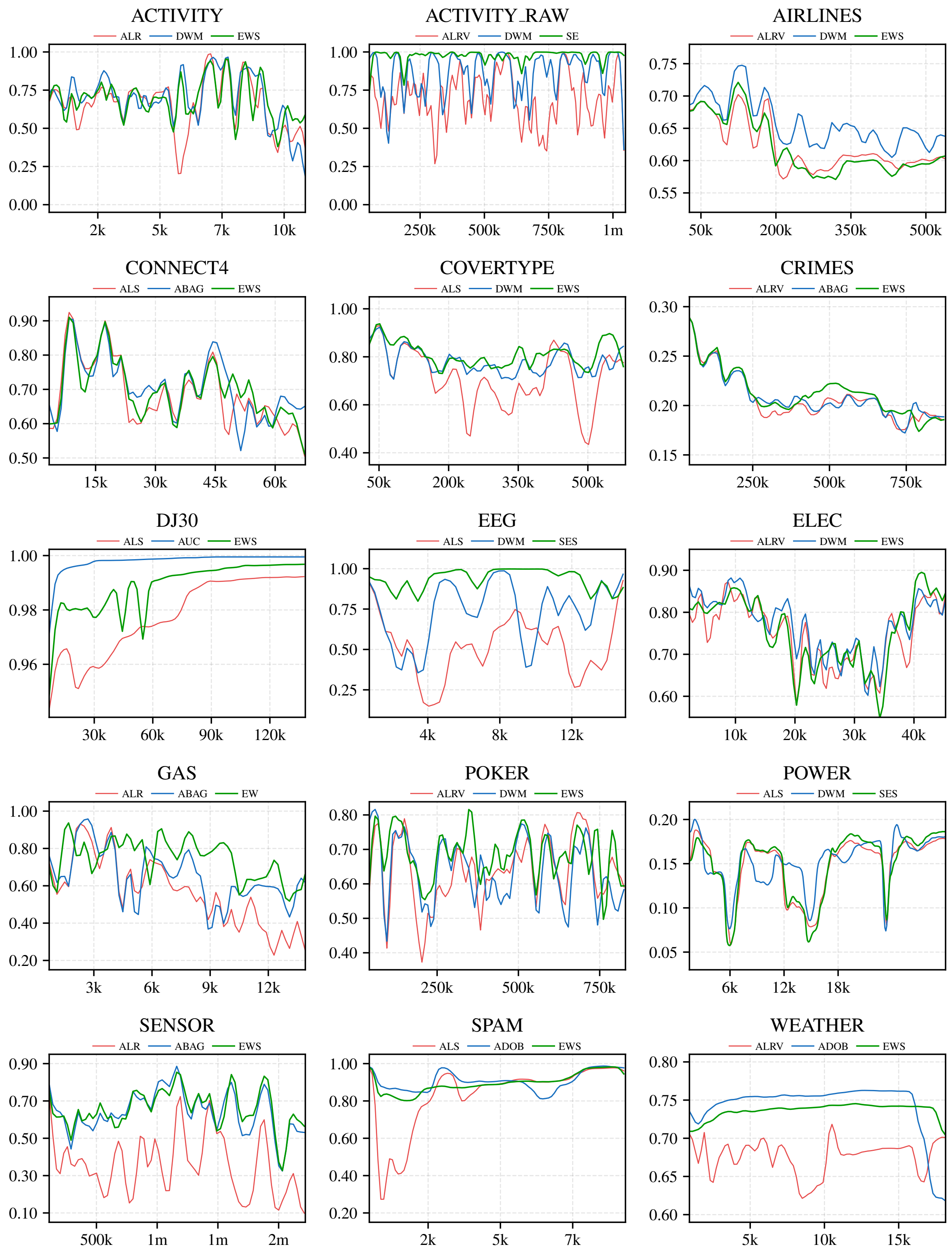


Fig. 36.: Accuracy series for AHT given $B = 10\%$: the best strategy vs. the best AL and the best (not our) ensemble.

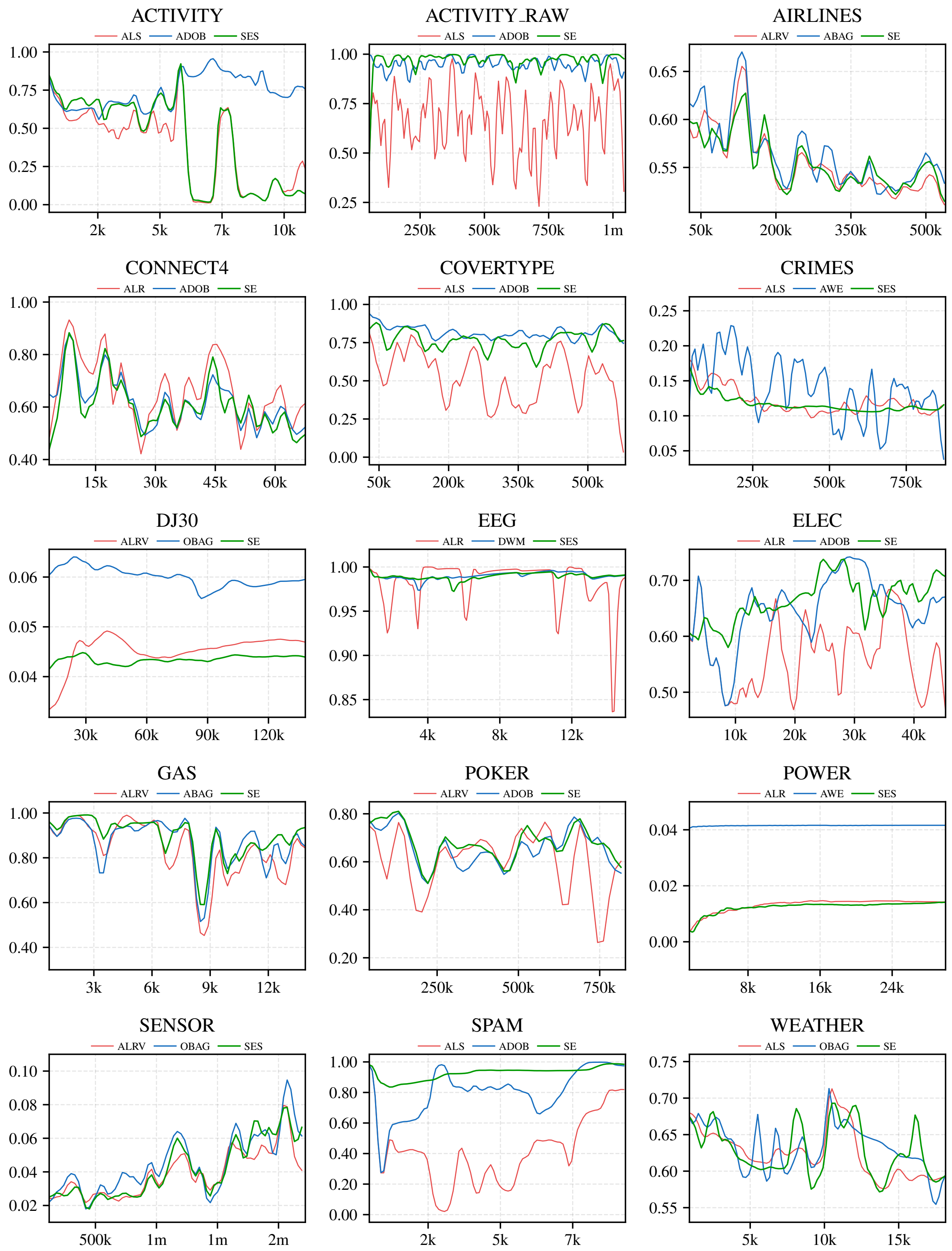


Fig. 37.: Accuracy series for SGD given $B = 10\%$: the best strategy vs. the best AL and the best (not our) ensemble.

Instance exploitation improves learning from drifting streams on a budget. In most of the considered cases, almost all strategies provided significant improvements over the baseline using only active learning and adapting in a conservative way (**RQ1**). Without more intensive learning, the base classifiers were not able to deal with strict supervision limitations and drifts at the same time, even if supported by a more strategic label query. They struggled with providing reactive up-to-date models for dynamic temporary concepts and end up with unresolved and severe underfitting. Our method addressed this problem by forcing the classifiers to take the risk of exploiting only the labeled instances they could use and enhancing the adaptation process as a result. The improvements were more impactful for real streams. With the abundance of positive examples collected from diverse measurements and statistical tests, and supported by the fact that generally higher intensity of exploitation was preferred, we prove that the problem of underfitting is critical for realistic streaming scenarios and that risking overfitting by using instance exploitation is more reasonable in practice.

No free lunch theorem for strategies and settings. We were not able to explicitly select one of the three strategies (UW/EW, SE) as the best one (**RQ2**). While EW^r worked best with AHT, SGD exhibited the best synergy with SE^r. Furthermore, although all the strategies generally worked better with the more risky settings using higher values of intensity λ_{max}, we were forced to distinguish different values of this parameter for different classifiers. Also, AHT preferred larger sliding windows than SGD. The distinction was caused by the fact that some algorithms may be more reactive to changes (AHT) than others (SGD), therefore, they may need a different amount of instance exploitation.

Dynamic control comes with a reasonable trade-off. The heuristics for controlling intensity and window size in a dynamic way do not provide improvements in predictive performance in most cases, even impairing it to some limited extent. On the other hand, they are able to reduce running time and memory consumption, respectively (**RQ2**). For the latter, adjusting the window size based on ADWIN is a contradictory exception from the observations – this algorithm provides the best quality of classification for AHT and reliable one for SGD, while requiring more memory than most of the other windows. We would recommend using fixed intensity λ_{max} and dynamic window size based on ADWIN ($\omega_{max} = \omega_{ADW}$), if someone does not have problems with the utilization of resources. Otherwise, we suggest using dynamic intensity (Eq. 5.1) and dynamic window (Eq. 5.2) with a limited size $\omega_{max} = 1000$, while utilizing ADWIN-based error as a control signal $\epsilon = \epsilon_{ADW}$.

Switching and elevating alleviates the risk of overfitting. The ensemble-based techniques for avoiding turning one problem (underfitting) into another (overfitting) provided the assumed improvements (**RQ3**). The methods are able to guarantee a more reliable lower bound on performance. Since we can efficiently track errors even under limited supervision, the ensembles can correctly determine a temporarily better approach. As a consequence in a prevalent number of cases (except for SGD using SE^r) they are at least as good as the standard or risky learner, providing additional enhancements in many scenarios. One should, however, keep in mind that the advantage of using the ensembles is usually on the brink of significance. Furthermore, we did not observe a significant difference between switching and elevating, therefore, since the latter requires very time-consuming model replacements, we recommend using switching. Finally, taking into account all considered factors, we can distinguish EWS and SES as the best methods on average.

Proposed strategies are competitive against other streaming classifiers.

The final comparison revealed that our algorithms, mainly committees, not only improve upon the baseline using only active learning, but that they can also be at least competitive against some of the state-of-the-art streaming ensembles, including bagging and boosting, and outperform them in many cases (**RQ4**). At the same time, while other ensembles utilize 10 base learners, ours use only two of them. Finally, although we could extend this comparison to more ensembles, one should notice that this study was not entirely focused on ensemble-based methods. In fact, there is a lot of potential for further improvements in this direction. We conducted only a limited comparison to place our algorithms in some recognizable context.

5.4 Summary

In this work, we have addressed a challenging, yet crucial issue in data stream mining domain – how to increase the adaptation capabilities of classifiers under concept drift while dealing with very limited access to class labels. Sparsely labeled data streams are predominant in a plethora of real-world applications and a lack of labeled instances increases the already high difficulty of recovery from changes. We have proposed a novel and flexible framework for instance exploitation in order to boost the learning from streaming data on a budget. By using a sliding window with a probabilistic sampling for selecting instances for additional exposure to the online classifier we were able to force faster adaptation rates, resulting in a significantly improved robustness to concept drift. We developed three instance exploitation strategies to alleviate the problem of underfitting of classifiers to new emerging concepts by various levels of expositions of labeled instanced obtained from active learning. In order to minimize the risk of overfitting, we have proposed two ensemble architectures that dynamically switch between learners based on aggressive and standard instance exposition. All our strategies are incorporated in a flexible wrapper framework capable of working with any active learning strategy and online classifier.

Based on an extensive experimental study, we were able to show the tremendous gains of using our strategies while learning temporary concepts from data streams and having limited access to class labels. The analysis of the results allowed us to reach in-depth and unique insights into the adaptation of classifiers under concept drift, showing how access (or lack of thereof) to an adequate number of labeled instances is of crucial importance in the dynamic settings. We formed a set of recommendations on how and when to use the proposed strategies in order to improve learning rates from sparsely labeled non-stationary data streams at no additional cost.

Our future works will concentrate on using instance exploitation to leverage ensemble learning and control ensemble diversity under concept drift, as well as to improve learning from sparsely labeled imbalanced data streams. We will also take into consideration noisy streams which may pose additional challenges when using our aggressive updates.

CHAPTER 6

ONLINE OVERSAMPLING FOR SPARSELY LABELED IMBALANCED AND NON-STATIONARY DATA STREAMS

Skewed data distributions are a very challenging topic in standard machine learning, being present there for over 25 years. In the continual learning domain, the class imbalance becomes even more difficult, as we not only need to deal with disproportion among classes but also with their evolving nature [143]. Here, the proportions between classes change dynamically, as well as class roles – minority may become a majority over time [146]. In addition to that, they may be subject to simultaneous concept drifts. As a result, we get complex and perplexing scenarios that actually occur in many real-life applications.

Although there exists a plethora of solutions for imbalanced problems [144], adapting them to data streams and concept drift is not straightforward. Many of the algorithms dedicated to imbalanced data streams concentrate on binary problems [217, 207, 143]. From the context of data streams, multi-class imbalanced problems are even more interesting, as they occur when new classes emerge, old ones disappear, or break into subconcepts [218]. They pose a bigger challenge, as relationships among classes are no longer well-defined and one cannot decompose them into binary sub-problems without losing valuable information [146]. There exist but few algorithms dedicated to multi-class imbalanced data streams, but they either focus on changing class ratios without drifts [219, 159], or on handling concept drift with static class ratios [150, 220]. Additionally, many existing solutions assume unrestricted access to class labels.

We propose a novel learning framework for multi-class data streams that addresses all of the mentioned challenges: (i) changing imbalance ratios among multiple classes; (ii) concept drift; and (iii) limited access to ground truth. We use active learning for selecting the most valuable instances for labeling and then use them to perform the multi-class oversampling. We guide our selection and instance generation procedures with a hybrid criterion that takes into account both current class ratios and the classifier error on each class independently. This allows us to effectively tackle both concept drift and class imbalance. At the same time, the fact that we use the oversampling module, generating additional instances, may be a solution to the underfitting after drifts regardless of a class imbalance, so it potentially addresses the problem of the limited budget simultaneously. An experimental study shows that our approach, based only on a single classifier, can provide sufficient solutions to the described problems, improving upon both standard active learning and more sophisticated ensembles, dedicated to learning from multi-class imbalanced data streams or simply performing well in general cases.

6.1 Deceptive majority and budget constraints

The existing algorithms for multi-class imbalanced data streams with dynamic class ratios do not take into account two crucial aspects of learning from streaming sources. The first one is that while they adapt to changes in class proportions they do not provide any explicit mechanism to handle concept drifts that are common in streams and may occur concurrently with class ratio changes. Why is it important?

Let us consider a case in which we have a majority class c_1 consisting of 80% of all instances and minority c_2 with 20% of incoming objects. Now, during some period of time not only the ratios swap but also class concepts completely change, which means that both c_1 and c_2 should now be represented by new models. If an algorithm is based only on class ratios, it oversamples only the minority class c_1, improving its adaptation, while for the entirely new majority class c_2, which also requires significant updates, we will rely only on incoming instances. One may say that it is fine since we balance learning and if c_2 turned into majority it does not require additional instances. It is reasonable until we take into consideration the second crucial facet of learning from data streams – labeling budget constraints. If the number of instances that can be used for updating is significantly limited, then not only do minority instances may suffer from underfitting and require some amount of oversampling, but also we may encounter a high error for the majority classes. We call it a *deceptive majority.* Taking this fact into account may not only help with modeling the majorities more accurately but also improve the minorities by excluding more precisely the subspaces to which they do not belong. We assume that in such scenarios potential underfitting is more likely to occur and impede learning more than overfitting.

6.2 Framework

In this section, we present our algorithms that are able to tackle all of the mentioned problems emerging during learning from imbalanced data streams. We designed an online wrapper framework given in Alg. 5.

Active learning. Our approach is, in fact, a combination of active learning and oversampling techniques. The former one (`QueryStrategy`) is used to limit the number of labeling requests for incoming instances $\boldsymbol{x}$ by asking only for valuable ones given some criteria. If a current budget spending $\hat{b}$ does not exceeds an available budget B and the method decides that a new object should be labeled, we acquire a true class c for the instance and use it to update a classifier L. The available budget B is a fraction of instances that can be labeled. Since streams are infinite by defi-

Algorithm 5: Framework for learning from imbalanced data streams.

```
Data: labeling budget $B$, QueryStrategy, OversamplingStrategy, budget
      spending $\hat{b}$, generated instances $\boldsymbol{S}$, metrics $\boldsymbol{M}$
Result: classifier $L$ at every iteration
Initialization: $\hat{b} \leftarrow 0$, $\boldsymbol{S} \leftarrow []$, $\boldsymbol{M} \leftarrow []$
repeat
    receive incoming instance $\boldsymbol{x}$;
    if $\hat{b} < B$ and QueryStrategy $(\boldsymbol{x}) = true$ then
        request the true label $c$ of instance $\boldsymbol{x}$;
        update labeling expenses $\hat{b}$;
        update classifier $L$ with $(\boldsymbol{x}, c)$;
        update class metrics $\boldsymbol{m}_c \in \boldsymbol{M}$
        $\boldsymbol{S} \leftarrow$ OversamplingStrategy $(\boldsymbol{x}, \boldsymbol{m}_c)$;
        for $i \leftarrow 1$ to $len(\boldsymbol{S})$ do
            update classifier $L$ with $(\boldsymbol{s}_i, c)$;
until stream ends;
```

nition, we approximate the current spending b with $\hat{b}$ as a ratio of labeled instances to all already acquired. A few online strategies have been already proposed that can be used in our framework [163]. They are usually based on uncertainty, like Rand-Var or selective sampling, however, to the best of our knowledge, there are no active learning strategies for multi-class imbalanced streams with dynamic ratios. We do not focus on this module in our work. Instead, we are going to show that problems with limited budgets (underfitting) and class imbalance, while relying on the active learning alone, can be effectively handled by adding an additional module responsible for oversampling.

Oversampling. While the active learning approach is a step forward to handling realistic streaming scenarios, it may still be insufficient under strict budget constraints, when a very limited number of instances is used. To handle this problem, we propose an adaptation enhancement in a form of synthetically generated instances. Since in this work we focus on skewed data distributions, we use oversampling techniques for this purpose. We generate additional instances $\boldsymbol{S}$ accordingly to a given `OversamplingStrategy`. It takes the labeled instance $\boldsymbol{x}$ as a prototype and class metrics $\boldsymbol{m}_c$ for the class c the instance belongs to. The oversampling strategy may use different: (i) generation methods that define how new instances are created; (ii) balancing strategies that determine how many instances are generated based on class metrics.

6.2.1 Generation methods

We specify two incremental generation methods that synthesize additional instances $\boldsymbol{S}$. They are rooted in the offline oversampling domain.

Single Exposition (SE) – it is a fully online approach that simply duplicates d times a given instance $\boldsymbol{x}$ that is exposed to the algorithm only once.

SMOTE (SM) – in this method we maintain a sliding window $\boldsymbol{W}$ of ω_{max} latest instances $\boldsymbol{w}$ that represent current concepts. Each class has its own window, so we keep ω_{max} already received instances for each of them. New instances are generated using the SMOTE algorithm [221]. For a labeled instance $\boldsymbol{x}$ we find its k nearest neighbors (belonging to the same class c), generate synthetic instances using Algorithm 2 (the gap is calculated using the uniform distribution $\mathcal{U}(0,1)$) and then duplicate them d times.

Algorithm 6: SMOTE single instance generation.

```
Data: new instance $\boldsymbol{x}$, window instance $\boldsymbol{w}$
Result: generated instance $\boldsymbol{s}$
$gap \leftarrow \mathcal{U}(0,1)$;
for $i \leftarrow 1$ to $len(\boldsymbol{x})$ do
    $diff \leftarrow \boldsymbol{w}[i] - \boldsymbol{x}[i]$;
    $\boldsymbol{s}[i] \leftarrow \boldsymbol{x}[i] + gap * diff$;
return $\boldsymbol{s}$
```

6.2.2 Balancing strategies

This module is responsible for balancing the learning process. In general, we want to use a function $d(\boldsymbol{m_c}) = d_{max}\gamma(\boldsymbol{m_c})$, where d_{max} is a maximal number of duplications that can be created and $\gamma(\boldsymbol{m_c})$ is a balancing function transforming metrics $\boldsymbol{m_c}$ for a class c into a value $v \in \langle 0, 1 \rangle$.

Dynamic Class Ratio (DCR). The most straightforward approach is to generate d instances in a negative relation to a ratio λ_c for a class c. We can apply the ratios directly to get:

$$\gamma(\boldsymbol{m_c}) = \gamma(\lambda_c) = 1 - \lambda_c, \tag{6.1}$$

however, in such a case, for balanced cases we would unnecessarily oversample some of the classes. A more reasonable approach is to perform oversampling relatively to the majority class λ_{max} [159]. Then for each class we can define the ratio as $\lambda'_c = \lambda_c/\lambda_{max}$, so we will try to oversample up to the *largest* class:

$$\gamma(\boldsymbol{m_c}) = \gamma(\lambda'_c) = \gamma(\lambda_c, \lambda_{max}) = 1 - \lambda_c/\lambda_{max}. \tag{6.2}$$

Since we aim to solve not only multi-class but also dynamic ratio problems, we need to apply an adaptation mechanism to the maintained class ratio values. We use the sliding window approach in our algorithms. Although the presented method is theoretically able to handle multi-class problems with dynamic ratios, it still does not take into account the problem of the *deceptive majority* (Sec. 2.1).

Dynamic Hybrid Ratio (DHR). A possible solution to the problem is adding a concept drift detector and using its indications to guide the class balancing. There are a few existing online drift detectors (e.g., DDM [126]), which indicate a change discretely (absent/present) based on registered errors. Since we want to control the number of duplications in a continuous way, we utilize a class-wise error calculated within a sliding window. Different performance metrics can be used for this purpose. In our case, we apply G-mean measure g_c that is calculated in one-vs-all manner for each class c separately [222].

For each incoming instance $\boldsymbol{x}$ we combine both metrics – the relative class ratio λ'_c and error $(1 - g_c)$ – using a simple weighted sum:

$$\gamma(\boldsymbol{m_c}) = \gamma(\lambda'_c, g_c) = \alpha_\lambda(1 - \lambda'_c) + \alpha_g(1 - g_c), \tag{6.3}$$

where $\alpha_\lambda + \alpha_g = 1$. Assuming that both coefficients are equal, one can easily see that when one class is simply a stationary majority (for example, $\lambda'_c = 0.8$ and $g_c = 1.0$), the strategy will practically ignore this class regarding oversampling, however, if a majority class is drifting (class error is expected to be high, for example, $\lambda'_c = 0.8$ and $g_c = 0.1$), the strategy will maintain some level of oversampling for this class to help a model adapt to a new class concept.

6.3 Experimental study

In our experimental study of learning from multi-class imbalanced streams with evolving class ratios, concept changes and under strict budget constraints we wanted to check the following research questions.

- **RQ1:** Does our combination of active learning and oversampling improve the former?
- **RQ2:** Do the hybrid balancing strategies outperform the simple ratio-based approaches? Is one generation method better than another?
- **RQ3:** Is our single-classifier framework competitive, in terms of classification performance and time consumption, compared with solutions proposed for learning from multi-class imbalanced data streams – MOOB/MUOB [159],

as well as with other state-of-the-art ensembles that are very efficient in general cases?

6.3.1 Data

To evaluate the given questions we utilized a set of 13 real benchmarks widely used in the data stream mining domain. Most of them come from the UCI repository (Connect4, Covertype, EEG, Gas, Poker) and Kaggle competitions (Crimes, Olympic, Tags). The rest of them are very popular in related publications. They are summarized in Tab. 15.

Table 15.: Summary of the used data streams.

Name	Instances	Att	Cls	Dyn	SC	Drifts	Δ
Activity	10 853	43	8	✓	4	1	0.89
Activity-Raw	1 048 570	3	6	✓	4	1	0.99
Connect4	67 557	42	3	-	3	2	0.84
Covertype	581 012	54	7	✓	4	1	0.97
Crimes	878 049	3	39	-	4	2	0.98
DJ30	138 166	8	30	-	4	2	0.99
EEG	14 980	14	2	✓	2	-	-
Electricity	45 312	8	2	✓	2	1	0.98
Gas	13 910	128	6	✓	3	1	0.65
Olympic	271 116	7	4	-	3	2	0.95
Poker	829 201	10	10	✓	4	2	0.98
Sensor	2 219 804	5	57	✓	4	2	0.99
Tags	164 860	4	11	-	4	2	0.98

We decided to split our evaluation into two parts. The first one is based on those **real data streams** that exhibit significant variability of class ratios over time (Dynamic). Since all of the presented data streams are supposed to consist of concept

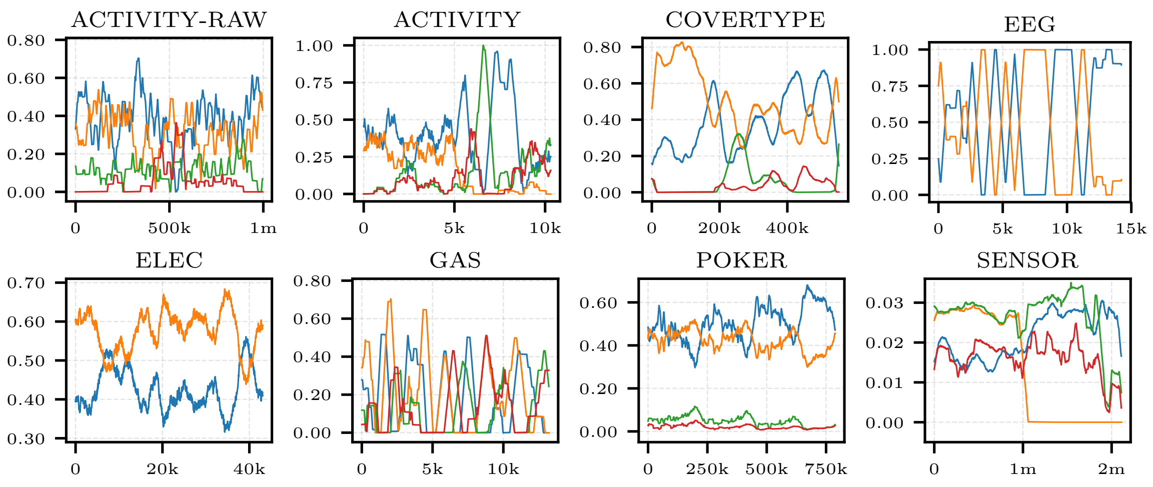


Fig. 38.: Dynamic class ratios for the used multi-class real data streams, each color represents a different class.

drifts, we can safely assume that, in at least some cases the class ratio dynamics occur simultaneously with the concept changes. We selected 8 of such data streams (Fig. 38, up to 4 classes are shown to preserve clarity).

To make sure that we evaluate our algorithms exactly in the described cases, we generated additional 12 **semi-synthetic data streams**, based on the real ones, using two fully controlled modifications. Firstly, we assigned classes in the streams to supersets (superclasses C_i), creating usually highly imbalanced majority and minority concepts. Secondly, we changed the assignments at some points to simulate class ratio and concept drifts. For example, if in Activity-Raw we assigned *Walking* and *Jogging* objects (about 70% of all instances) to a superclass C_1, and *Standing* objects (less than 5%) to C_2, then during a drift we reverse the relation, so all *Walking* and *Jogging* objects are C_2 now and all *Standing* instances become C_1. As a result, we simulate critical class ratio changes (C_1 is about 5% and C_2 is 70% after the change), as well as concept drifts, since both superclasses are represented by different distributions of objects before and after a change. The concept transitions were generated analogously to the formula from MOA, using the sigmoid function:

$$f(t) = 1/(1 + e^{-s(t-t_0)}), \tag{6.4}$$

where s controls the duration of change and t_0 is a peak of it. We created drifts of moderate lengths. All necessary details regarding the generation process can be found in our repository: github.com/mlrep/imb-drift-20.

In Tab. 15 we enclose a number of such changes (Drifts) and the proportion of objects that change concepts due to drifts (Δ). Both the concept drifts and class ratio changes are severe in almost all cases, therefore the generated data streams represent the most difficult scenarios we can encounter. If we also take into consideration the fact that, most likely, not all of the class ratio changes in the real streams occur with concept drifts at the same time, it is reasonable to say that, when it comes to handling the described dynamics, the generated streams are more challenging on average than the selected real ones. Since obtained class ratios are dynamic, we enclosed their values over time as an appendix in the repository.

6.3.2 Setup

Below we present the setup of our experiments. They can be easily reproduced, using the environment available on the given website.

Algorithms. To investigate if our combination of oversampling and active learning improves the latter (RQ1), we collected results for random **AL-R** (Random), **AL-RV** (RandVar) and **AL-S** (Sampling) without instance generation. We evaluated all combinations of our strategies **SE-DCR**, **SE-DHR**, **SM-DCR**, **SM-DHR** to check if there are substantial differences between generation and balancing strategies (RQ2). As an active learning strategy for our framework we picked theoretically universal **AL-RV** and Adaptive Hoeffding Tree (AHT) [109] as a base learner, which is a state-of-the-art classifier in the streaming data domain. Finally, we juxtaposed results for our strategies with already published ensembles for multi-class dynamic ratio streams – **MOOB** and **MUOB** (RQ3). In addition, we compared them with other well-known ensembles: Online Bagging (**OZABAG**) [223], Leveraging Bagging (**LB**) [117], Adaptive Random Forest (**ARF**) [119], Online Boosting using ADWIN (**OB-ADW**) [223] and Dynamic Weighted Majority (**DWM**) [121]. The ensembles used different versions of the Hoeffding Tree, depending on their default settings. All of them were connected with the AL-RV strategy for working on a budget.

Budgets. The algorithms were evaluated on different budgets, with a particular focus on realistic low ones, $B \in \{100\%, 50\%, 20\%, 10\%, 5\%, 1\%, 0.5\%, 0.1\%\}$.

Configurations. We varied the size of windows for SE and SM to make them reactive to the limited number of labeled instances $\omega_{max} \in \{1000, 500, 200, 100, 50, 10, 10, 10\}$. We set a fixed maximum number of duplications $d_{max} = 100$, a fixed number of nearest neighbors $k = 10$ for SM, as well as, equal coefficients for the DHR strategies: $\alpha_\lambda = \alpha_g = 0.5$. For the size of ensembles we chose 10 base learners (we have not observed significant improvement for larger ensembles). All the Hoeffding Trees used default settings. For AL-RV we selected default $\theta = 0.01$ as its variable threshold step.

Metrics. We collected classification efficacy and computing performance for all classifiers. For the former, we used the generalized multi-class form of G-mean, which is given as $G_n = \sqrt[n]{R_1 \cdot R_2 \cdot ... \cdot R_n}$, where R_i is a class-wise recall and n is a number of classes [222]. It was calculated using the prequential evaluation method. Bonferroni-Dunn ranking test with significance level $\alpha = 0.05$ was used to compare examined algorithms over multiple datasets. For the performance of computations we registered update and classification time per instance separately.

6.3.3 Results

We present the average results for all algorithms and data streams under different budget constraints in Tab. 16.

Improving active learning. The first observation is that our framework was able to enhance results over simple active learning strategies in all cases except for SM-DCR on $B = 100\%$. For the real data streams, we can see that the SM strategies provide a steady increase of the improvements, compared with the best active learning strategy for a given setting, as budget constraints are being tightened from $B = 100\%$ down to $B = 0.5\%$ (Fig. 39). It starts from about 1.1 for SM-DHR and ends at more than 1.36 for the same strategy. Results for the SE approaches exhibit the same trend for budgets higher than $B = 1\%$, with slightly lower values between approximately 1.09 and 1.29. Below the given budgets, SM and SE still provide some gain, however, they are no longer able to increase it. For the harder semi-synthetic streams the trend is even more clear (Fig. 39). The improvements for the SM strategies ranges from about 1.09 to more than 2.4 for DHR, and from 1.02 to almost 2.1 for SE using the same generation strategy. The most significant change can be observed after we limit the number of labeled instances below 5%, when the improvements become drastically higher. We suppose that the difference between results for the real streams and the semi-synthetic ones comes from the lower quality of the controlling metrics maintained by our algorithms (windowed class ratios, errors). It can be balanced by the difficulty of changes in a stream (like in the semi-synthetic ones and probably some of the real ones), when there is a higher chance that our approach will be effectively utilized.

Regardless of the quality of improvements, they are caused by the fact that the active learning strategies alone are not able to update base learners sufficiently while learning from extremely limited instances – single, sparsely labeled examples intro-

Table 16.: Average G-mean values calculated over all real streams (top) and semi-synthetic streams (bottom) for different algorithms given a budget.

REAL	100%	50%	20%	10%	5%	1%	0.5%	0.1%
AL-R	0.6415	0.6014	0.5586	0.4919	0.3966	0.3405	0.3237	0.2849
AL-RV	0.6158	0.5981	0.5327	0.4637	0.4531	0.3699	0.3175	0.2768
AL-S	0.6183	0.6034	0.5493	0.4775	0.4362	0.3802	0.3233	0.2702
SE-DCR	0.6983	0.7103	0.6580	0.6187	0.5724	0.4667	0.3741	0.3393
SE-DHR	0.7179	0.7250	0.6820	0.6282	0.5839	0.4561	0.3816	0.3318
SM-DCR	0.7048	0.7121	0.6625	0.6087	0.5706	0.4850	0.4313	0.3421
SM-DHR	0.7397	0.7395	0.6901	0.6459	0.5985	**0.4980**	**0.4424**	**0.3534**
MOOB	0.6441	0.6518	0.5756	0.5248	0.4880	0.3953	0.3619	0.3194
MUOB	0.2290	0.2191	0.2223	0.2165	0.2090	0.1774	0.2065	0.1622
OZABAG	0.6140	0.6150	0.5262	0.4852	0.4528	0.3984	0.3174	0.2757
LB	0.7404	0.7419	0.6946	0.6436	**0.6064**	0.4694	0.3461	0.2685
ARF	**0.7597**	**0.7596**	**0.7001**	**0.6524**	0.5977	0.4421	0.3453	0.2012
OB-ADW	0.6432	0.6324	0.5936	0.5538	0.5088	0.3886	0.3202	0.2498
DWM	0.7222	0.7223	0.6690	0.6181	0.5751	0.4254	0.3497	0.2866

SYNTH	100%	50%	20%	10%	5%	1%	0.5%	0.1%
AL-R	0.7229	0.6677	0.5970	0.5668	0.5088	0.3685	0.2992	0.2071
AL-RV	0.6628	0.6617	0.6047	0.5727	0.5108	0.3840	0.3079	0.1672
AL-S	0.6884	0.6713	0.6090	0.5642	0.5429	0.3882	0.2868	0.1593
SE-DCR	0.7399	0.7458	0.6973	0.6898	0.6645	0.5626	0.5434	0.4062
SE-DHR	0.7636	0.7758	0.7363	0.7144	0.6720	0.5707	0.5391	0.4330
SM-DCR	0.7198	0.7104	0.6713	0.6357	0.5954	0.5280	0.4967	0.4245
SM-DHR	0.7862	0.7938	**0.7688**	**0.7439**	**0.7142**	**0.6359**	**0.5946**	**0.5019**
MOOB	0.7369	0.7456	0.6743	0.6451	0.5922	0.4821	0.4256	0.3002
MUOB	0.4471	0.4208	0.4124	0.3895	0.3716	0.3318	0.2562	0.2444
OZABAG	0.6287	0.6262	0.5605	0.4999	0.4213	0.3208	0.2802	0.2226
LB	0.7936	0.7913	0.7312	0.6651	0.6123	0.4995	0.4264	0.2895
ARF	**0.8183**	**0.8115**	0.7603	0.7138	0.6683	0.5430	0.4644	0.3086
OB-ADW	0.7787	0.7614	0.7479	0.7115	0.6752	0.5718	0.5034	0.3603
DWM	0.7215	0.7195	0.6564	0.6459	0.6005	0.4606	0.4103	0.2656

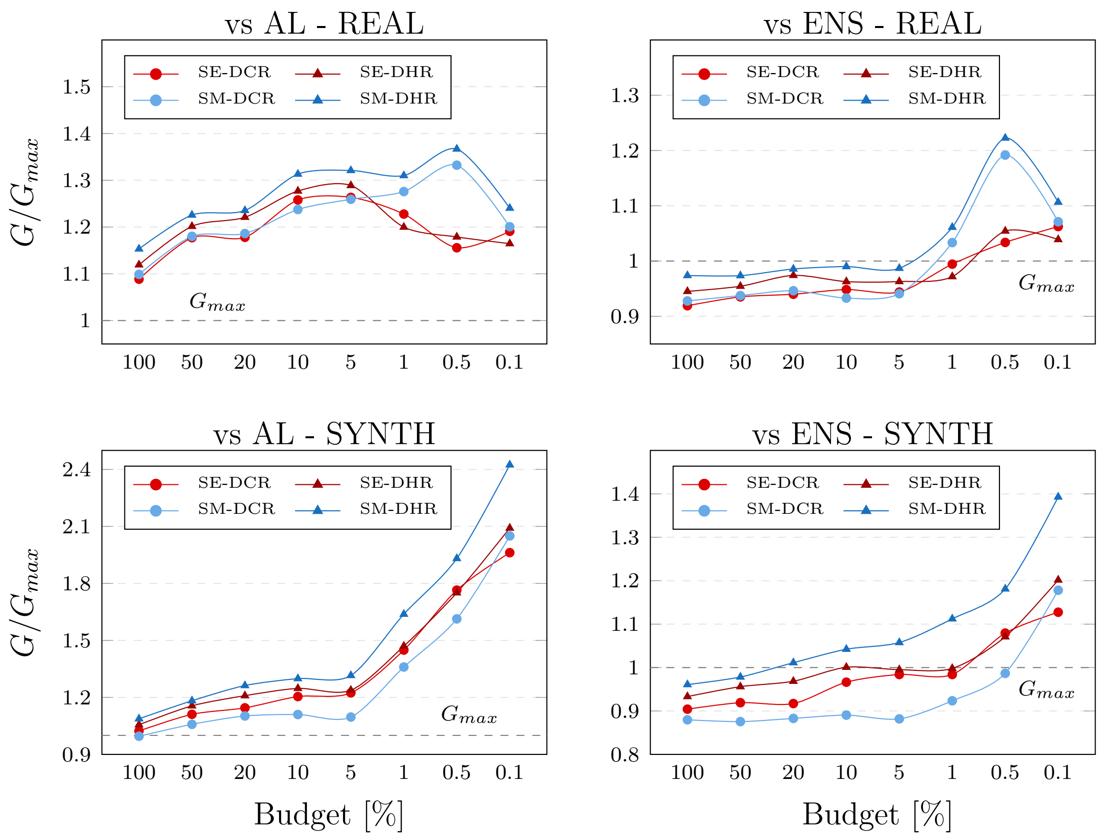


Fig. 39.: Ratios of the average G-mean for our algorithms (G) to results for the best (G_{max}) active learning (left) and ensembles (right) on given budgets.

duce inadequate changes to models in terms of reaction to skewed data distributions and severe concept drifts. Adding properly controlled oversampling helps with maintaining sufficiently balanced classifiers and prevents underfitting. The fact that we are able to increase the enhancements for lower budgets is particularly encouraging since these are the most realistic scenarios [163]. Results of ranking tests for all budgets (Fig. 40 and 41) show the significance of the differences.

Hybrid over class ratio. When we look at different combinations of our generation and balancing strategies (Tab. 16), we can conclude that methods based on DHR are generally better than those using DCR. One can also notice that the differences are

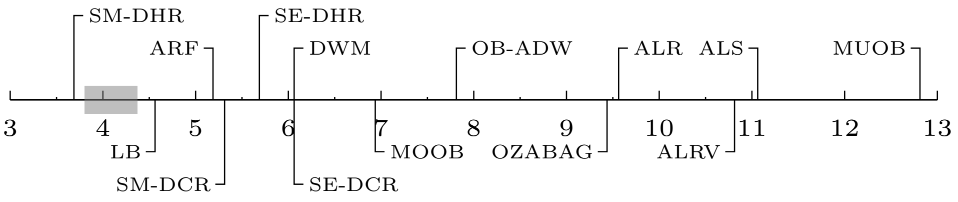


Fig. 40.: Bonferroni-Dunn test over all examined budgets for the real streams.

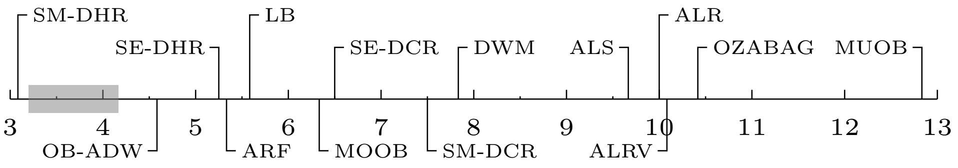


Fig. 41.: Bonferroni-Dunn test over all budgets for the semi-synthetic streams.

more substantial for SM (up to about 0.04 for the real streams and up to almost 0.12 for the semi-synthetic ones) than for SE (up to 0.02-0.03 and 0.04, respectively), for which they are on the brink of significance when averaged over all examined budgets (Fig. 40 and 41). It may mean that improvements for the very simple generation strategy are harder to achieve.

The differences occur for both groups of data streams. However, they are definitely more significant for the semi-synthetic ones. The class ratio driven approaches are not the best solutions that we can find if with the class ratio changes come severe concept drifts. In such scenarios, especially when a budget is limited, the majority classes also need to be sufficiently handled by boosting the adaptation process with additionally generated instances. The DHR strategies provide the additional objects, based on the drift indicator – an error for a class. One should also notice that even if we claim that the DHR approach is more useful when data streams are characterized by more severe simultaneous class ratio and concept changes, it almost never performs worse than the DCR strategy, regardless of the difficulty of drifts.

Generation methods. The observation that the gap between DCR and DHR is more clear for SM than for SE is correlated with the fact that the former works exquisitely well with DHR and disappointingly with DCR, especially for moderate budgets between $B = 20\%$ and $B = 5\%$. In particular, it can be seen for the semi-synthetic streams (Fig. 41). SE is more stable and combines better with DCR, however, at the same time it does not achieve as good results as SM with DHR. Eventually, we do not distinguish any generation method as significantly better than another.

Comparison with ensembles. Most importantly, although our algorithms do not improve upon ensembles for high and very high budgets above $B = 20\%$, the relation between gain and budget is similar as for the active learning strategies. In most cases, except for the lowest budget for the real data stream, we can observe that with decreasing budget our chances for improvements increase (Fig. 39), which once again, is a very important property since smaller numbers of labeled instances are more realistic.

Analogously to the results comparing our solutions with the active learning, we can observe that improvements upon the best ensembles on given budgets (usually LB or ARF) are much more clear for the results obtained from more challenging semi-synthetic streams. In particular, it can be noticed for our the most efficient combination – SM-DHR – which was better than any of the considered ensembles for the real streams on very low budgets below $B = 5\%$ (from about 1.05 to more than 1.2, Tab. 16) and for the semi-synthetic streams on low budgets below $B = 20\%$ (from 1.01 to nearly 1.4). The rest of our strategies were at least competitive on budgets lower than $B = 10\%$. One should notice that SM-DHR once again was resilient to very low budgets while compared to other algorithms. As a result, SM-

DHR turned out to be the best algorithm overall (Fig. 40 and 41), outperforming all other classifiers. Also, SM-DCR was very competitive for the real streams (most likely due to the less severe concurrent class ratio and concept changes) and SE-DHR for the semi-synthetic ones.

Furthermore, it is worth mentioning that in nearly all cases with a limited budget each of our algorithms, except for SM-DCR working on the semi-synthetic streams, was better than MOOB and MUOB (Tab. 16), which are considered state-of-the-art algorithms for the problem of learning from multi-class imbalanced data streams. In addition, one can notice that the DCR-based combinations, which use the similar balancing principle as MOOB, can also be better than the ensemble – SE-DCR exhibits higher quality for budgets lower than 50%, SM-DCR when less than 5% labeled instances are available. It is most likely caused by the fact that our strategies tend to generate many more additional instances than the bagging-based algorithms, so they are less likely to suffer from underfitting, like MUOB. Finally, the negative results for the undersampling ensemble prove that using this technique while working with highly limited budgets is not a reasonable approach.

The presented results show that our hybrid approach is adequate to the presented challenging scenarios and that currently available solutions can be meaningfully improved, especially under realistic budget constraints.

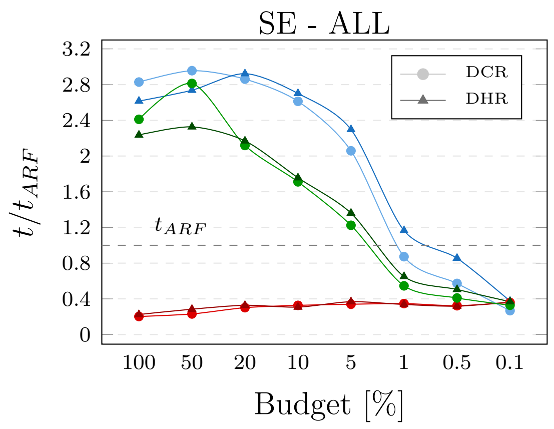
Time consumption. In Tab. 17 we enclose the average total running time per instance calculated over all data streams (real and semi-synthetic), as well as we distinguish proportions (bars in cells) of the time used for updates and classification. Interestingly, while ensembles spend more time on classification, our strategies use most of it for updates. Generally, there is also a pattern of dominating updates on higher budgets for all algorithms – it is probably caused by the nature of the base

Table 17.: Average total running time [ms] per instance for all algorithms given a budget. Update time is blue, classification is red.

Algorithm	100%	50%	20%	10%	5%	1%	0.5%	0.1%
AL-R	0.0203	0.0104	0.008	0.0099	0.0061	0.0052	0.0042	0.0041
AL-RV	0.0107	0.0124	0.0101	0.0064	0.0056	0.0051	0.0049	0.0041
AL-S	0.0135	0.0127	0.0128	0.0063	0.0059	0.0045	0.0043	0.0041
SE-DCR	0.1741	0.1596	0.0788	0.0473	0.0275	0.0093	0.0067	0.0049
SE-DHR	0.1614	0.1485	0.0807	0.0486	0.0306	0.0111	0.0083	0.0055
SM-DCR	1.1116	1.0767	0.4873	0.2579	0.1457	0.039	0.0225	0.0089
SM-DHR	0.9837	0.9403	0.4716	0.2699	0.166	0.051	0.027	0.0102
MOOB	0.1329	0.1482	0.0919	0.0689	0.0655	0.0556	0.054	0.0521
MUOB	0.0194	0.0199	0.0221	0.0197	0.0167	0.0182	0.0149	0.0114
OZABAG	0.0764	0.0827	0.0633	0.0562	0.0526	0.0471	0.0464	0.0446
LB	0.1187	0.1235	0.0822	0.0672	0.0588	0.0476	0.0446	0.0459
ARF	0.0722	0.0638	0.0372	0.0276	0.0225	0.0171	0.0164	0.015
OB-ADW	0.3092	0.2153	0.1193	0.0584	0.0467	0.0386	0.0355	0.0279
DWM	0.0833	0.0764	0.0524	0.0368	0.0284	0.0259	0.0263	0.0239

learner used (Hoeffding Trees), which for more labeled instances builds more complex structures that require more time-consuming updates.

For high budgets above 10% we can segregate the solutions into three groups – fast active learning methods and MUOB (about 0.005-0.02 ms), moderate SE along with all other ensembles (0.06-0.3 ms), and relatively very slow SM strategies (0.48-1.11 ms). The performance of the last one is caused mainly by the naive nearest neighbor search within the sliding window, which can be improved using a dedicated data structure. On the other hand, one should also notice that the differences significantly change as budgets get lower – and these are the scenarios to which we dedicate our methods.



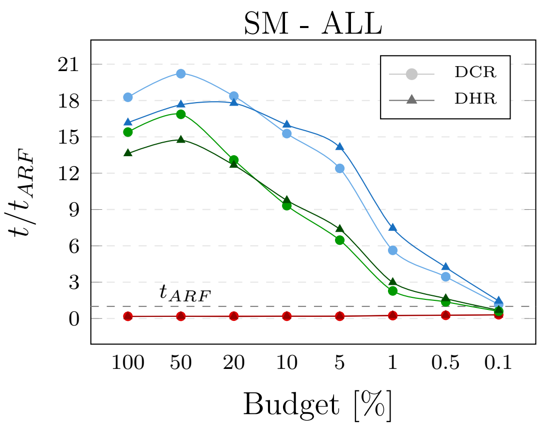


Fig. 42.: Ratios of running time per instance for our algorithms (t) to the results for ARF (t_{ARF}). The update time is blue, classification is red and the total time is green. Ratios for DHR are in darker colors than for DCR.

It is worth noting that since all our strategies depend on the number of classes (class ratio and error), we observed that the total running time for the real data streams is higher on average than for the semi-synthetic streams (larger numbers of classes).

In Fig. 42 we can observe how the ratios of the computation time for our strategies to measurements for the best ensemble – ARF (on average) – change with budget. We can clearly see that as the budget decreases the ratios decrease, in favor of our methods. The SE approaches are competitive on the highest budgets and become even faster than ARF if less than 5% labeled objects are available. The SM methods are more than 15 times slower than ARF on $B = 50\%$, however, they smoothly reduce the processing time (smaller windows, simpler AHT) and become competitive for budgets below 5%. Furthermore, for both generation methods we can see that the DCR strategies are slightly faster than DHR (probably because the latter ones tend to generate more instances). Finally, even if ratios for the update time remain unfavorable in most cases, the overall time reduces faster, since as the update time drops for all algorithms the ensemble classification time starts dominating not only in proportions (Tab. 17) but also in absolute values, compared with our single-classifier framework.

6.4 Summary

To conclude, in this chapter we presented a single-classifier framework addressing the problem of learning from multi-class imbalanced data streams with dynamic class ratios and concurrently drifting concepts. We analyzed our and referential solutions under a wide range of budgets for labeling, including very strict constraints when even less than 1% of labeled instances are available. The experimental results most importantly show the following.

- Combining active learning with oversampling improves the former by preventing underfitting and equilibrating adaptation between classes (**RQ1**).
- Hybrid balancing strategies enhance simple approaches based on class ratio when dealing with concurrent ratio and concept changes (**RQ2**).
- Our single classifier framework using the best configuration (SM-DHR) is able to outperform existing ensemble solutions, which ignore the fact that concept changes may occur simultaneously also for dominating classes, making themselves susceptible to what we call the *deceptive majority* (**RQ3**).
- Our solutions are competitive also in terms of running time per instance.

Finally, we observe that the presented strategies exhibit their primacy over active learning and the ensembles especially when the number of labeled instances is critically low – it reflects in both classification quality and computing performance. We find it essential since these are the most realistic scenarios one can encounter. Taking into account both metrics, we recommend using SE-DHR when relatively higher budgets are available (above 5%) and SM-DHR for the highly limited ones (below 5%). They provide the best quality-time improvement ratio for the given ranges of budgets.

In our future works, we will consider providing a more in-depth analysis of used parameters (window sizes, numbers of duplications, hybrid ratio weights) in the context of different concept and ratio changes, including their severity. We may also investigate other than G-mean measures for balancing strategies.

CHAPTER 7

CONCEPT DRIFT DETECTION FROM MULTI-CLASS IMBALANCED DATA STREAMS

While there exist a plethora of drift detectors proposed in the literature, most of them share two limitations: (i) they assume roughly balanced data distributions and thus are likely to omit concept drift happening in minority classes; and (ii) they monitor global data stream characteristics, thus detecting concept drifts that affect the entire stream, not particular classes or decision regions. This makes the state-of-the-art drift detectors unsuitable for mining imbalanced data streams, especially when multiple classes are involved. There is a need to develop a new drift detector that is skew-insensitive, can monitor multiple classes at once, and can rapidly adapt to changing imbalance ratios and classes switching roles, as none of the existing methods is capable of this.

In this chapter, we propose **RBM-IM**, a trainable concept drift detector for continual learning from multi-class imbalanced data streams. It is designed as a Restricted Boltzmann Machine neural network with skew-insensitive modifications of the training procedure. We use it to track the reconstruction error for each class independently and signal if any of them has been subject to a significant change over the most recent mini-batch of data. Our drift detector re-trains itself in an online fashion, allowing it to handle dynamically changing imbalance ratio, as well as evolving class roles (minority classes becoming the majority and vice versa). RBM-IM is capable of detecting drifts occurring at both global and local levels, allowing for complex monitoring of multi-class imbalanced data streams and understanding the nature of each change that takes place.

We offer the following novel contributions to the field of continual learning from data streams.

- **Robustness to class imbalance.** RBM-IM provides robustness to multi-class skewed distributions, offering excellent detection rates of drifts appearing in minority classes without being biased towards majority concepts.
- **Detecting local and global changes.** RBM-IM is capable of detecting concept drifts affecting only a subset of minority classes (even when only a single class is affected), offering a better understanding of the nature of changes than any state-of-the-art drift detector.
- **Taxonomy of multi-class imbalanced data streams.** we propose a systematic view of possible challenges that can be encountered in continual learning from multi-class imbalanced data streams and formulate three scenarios that allow us to model such changes.
- **Extensive experimental study.** we evaluate the efficacy of RBM-IM on a thoroughly designed experimental test bed using both real-world and artificial benchmarks. We introduce a novel approach towards evaluating concept drift detectors on imbalanced data streams, by measuring their reactivity to drifts occurring only in a subset of minority classes, as well as by checking their robustness to increasing imbalance ratio among multiple classes.

7.1 Challenges in learning from multi-class imbalanced data streams

In static scenarios there is a plethora of works devoted to two-class imbalanced problems, but much less attention is paid to a much more challenging multi-class imbalanced setup [146]. The same carries over to the continual learning from data streams, where most of the works focused on binary streams [22]. This is highly limiting for many modern real-world applications and thus there is a need to develop skew-insensitive techniques that can handle multiple classes [224].

There is no single universal approach to how to view and analyze multi-class imbalanced data streams. Therefore, we propose a taxonomy of the most crucial problems that can be encountered in this setting, creating three distinctive scenarios. They cover various learning difficulties that affect one or more classes and thus pose significant challenges for both drift detectors and classifiers.

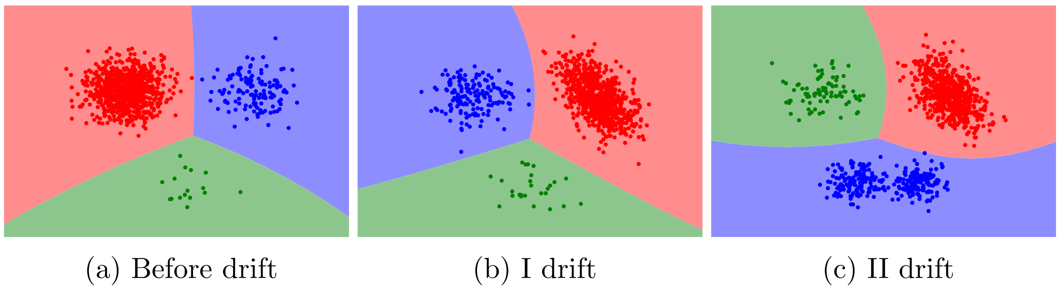

(a) Before drift (b) I drift (c) II drift

Fig. 43.: Scenario 1 – global concept drift and dynamic imbalance ratio.

Scenario 1: Global concept drift and dynamic imbalance ratio. Here we assume that all classes are subject to a real concept drift that will influence the decision boundaries. Additionally, the imbalance ratio among the classes changes together with the drift occurrences. However, class roles remain static and classes denoted as minority stay minority during the entire stream processing. This scenario poses challenges to drift detectors by varying the degree of changes in each class and how they actually impact the decision boundaries. Changes in minority classes may get overlooked by detector bias towards the majority ones, as usually they gather statistics over the entire data stream. This is depicted in Fig. 43.

Scenario 2: Global concept drift, dynamic imbalance ratio, and changing

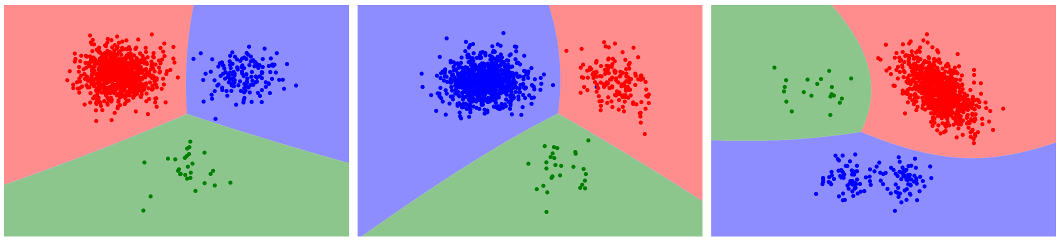

(a) Before drift (b) I drift (c) II drift

Fig. 44.: Scenario 2 – global concept drift, dynamic imbalance ratio, and changing class roles.

class roles. Here we extend Scenario 1 by adding the third learning difficulty – changing class roles. Now the imbalance ratio is subject to more significant changes and, as a result, classes may switch roles – minority may become majority and vice versa. This is especially challenging to track in a multi-class case, where relationships among classes are more complex. Drift detectors have difficulties with keeping any reliable statistics coming from classes that rapidly change their roles. This may lead to frequently switching bias towards whichever class is currently the most frequent one. This is depicted in Fig. 44.

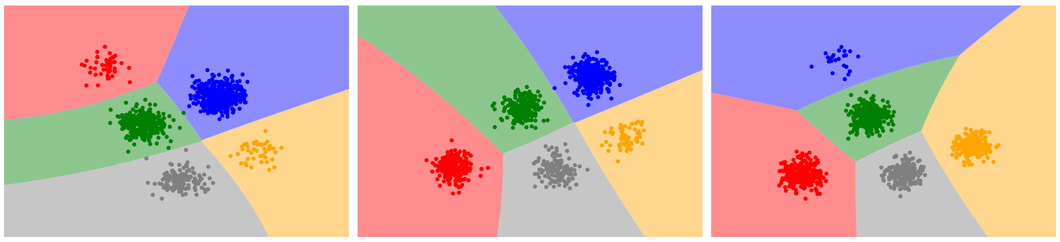

(a) Before drift (b) I drift (c) II drift

Fig. 45.: Scenario 3 – local concept drift, dynamic imbalance ratio, and changing class roles.

Scenario 3: Local concept drift, dynamic imbalance ratio, and changing class roles. This is the most challenging scenario that retains dynamic imbalance ratio and changing class roles from Scenario 2, but moves from global concept drift to the local one. That means in a given moment only a subset of classes (or even a single one) may be affected by a real concept drift, while the remaining ones are subject to no changes or a virtual concept drift that does not impact decision boundaries (see Sec. 2). In such a setting we should not only be able to tell if drift takes place but also which classes are affected. It is a big step towards understanding the dynamics of concept drift and offering classifier adaptation to specific regions of decision space (leading to savings in time and computational resources). This is the most challenging scenario for concept drift detectors, as changes happening in minority classes will remain unnoticed when a detector is biased towards the majority class. This is depicted in Fig. 45.

Real-world problems affected by multi-class imbalance and concept drift. The three defined scenarios are not only interesting from the theoretical point of view but also directly transfer to a plethora of real-world applications. In cybersecurity, we deal with multiple types of attacks that appear with varying frequencies (multi-class extremely imbalanced problems). Some of those attacks will dynamically change over time to bypass new security settings, while legal transactions will not be affected by such concept drift. In computer vision, target detection focuses on finding few specific targets, differentiating them from the information coming from a much bigger background. Targets may change their nature over time, being subject to variations, or even camouflage. In natural language processing, we must deal with constantly evolving wording/slang utilized by various minority groups, where changes in those groups will happen independently.

7.2 Restricted Boltzmann Machine for imbalanced drift detection

Overview of the proposed method. We introduce a novel concept drift detector for multi-class imbalanced data streams, implemented as a Restricted Boltzmann Machine (RBM-IM) with leveraged robustness to skewed distributions via a dedicated loss function. It is a fully trainable drift detector, capable of autonomous adaptation to the current state of a stream, imbalance ratios, and class roles, without relying on user-defined thresholds.

7.2.1 Skew-insensitive Restricted Boltzmann Machine

RBM-IM neural network architecture. Restricted Boltzmann Machines (RBMs) are generative two-layered neural networks [225] constructed using the $\boldsymbol{v}$ layer of V visible neurons and the $\boldsymbol{h}$ layer of H hidden neurons:

$$\begin{aligned} \boldsymbol{v} &= [v_1, \cdots, v_V] \in \{0,1\}^V, \\ \boldsymbol{h} &= [h_1, \cdots, h_H] \in \{0,1\}^H \end{aligned} \tag{7.1}$$

We deal with supervised continual learning from data streams (as defined in Sec. 2), thus we need to extend this two-layer RBM architecture with the third $\boldsymbol{z}$ layer for class representation. It is implemented as a continuous encoding, meaning that each neuron in $\boldsymbol{z}$ will return its real-valued support for each analyzed class (thus being responsible for the classification process). By $\boldsymbol{m}_z$ we denote the vector of RBM outputs with support returned by the z-th neuron for the m-th class. This allows to define $\boldsymbol{z}$, known also as the class layer or the softmax layer:

$$\boldsymbol{z} = [z_1, \cdots, z_Z] \in \boldsymbol{m}_1, \cdots, \boldsymbol{m}_Z. \tag{7.2}$$

This class layer uses the softmax function to estimate the probabilities of activation of each neuron in $\boldsymbol{z}$.

RBMs do not have connections between units in the same layer, which holds for $\boldsymbol{v}$, $\boldsymbol{h}$, and $\boldsymbol{z}$. Neurons in the visible layer $\boldsymbol{v}$ are connected with neurons in the hidden layer $\boldsymbol{h}$, and neurons in $\boldsymbol{h}$ are connected with those in the class layer $\boldsymbol{z}$. The weight assigned to a connection between the i-th visible neuron v_i and the j-th hidden neuron h_j is denoted as w_{ij}, while the weight assigned to a connection between the j-th hidden neuron h_j and the k-th class neuron z_k is denoted as u_{jk}. This is used to define the RBM energy function:

$$\begin{aligned} E(\boldsymbol{v},\boldsymbol{h},\boldsymbol{z}) = &-\sum_{i=1}^{V} v_i a_i - \sum_{j=1}^{H} h_j b_j - \sum_{k=1}^{Z} z_k c_k \\ &-\sum_{i=1}^{V}\sum_{j=1}^{H} v_i h_j w_{ij} - \sum_{j=1}^{H}\sum_{k=1}^{Z} h_j z_k u_{jk}, \end{aligned} \tag{7.3}$$

where a_i, b_j, and c_k are biases introduced to $\boldsymbol{v}, \boldsymbol{h}$, and $\boldsymbol{z}$ respectively. Energy formula $E(\cdot)$ for state $[\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{z}]$ is used to calculate the probability of RBM of being in a given state (i.e., assuming certain weight values), using the Boltzmann distribution:

$$P(\boldsymbol{v},\boldsymbol{h},\boldsymbol{z}) = \frac{\exp\left(-E(\boldsymbol{v},\boldsymbol{h},\boldsymbol{z})\right)}{F}, \tag{7.4}$$

where F is a partition function allowing to normalize the probability $P(\boldsymbol{v},\boldsymbol{h},\boldsymbol{z})$ to 1.

Hidden neurons in $\boldsymbol{h}$ are independent and use features given by the visible layer $\boldsymbol{v}$. The activation probability of the j-th given neuron h_j can be calculated as follows:

$$\begin{aligned} P(h_j|\boldsymbol{v},\boldsymbol{z}) &= \frac{1}{1+\exp\left(-b_j - \sum_{i=1}^{V} v_i w_{ij} - \sum_{k=1}^{Z} z_k u_{jk}\right)} \\ &= \sigma\left(b_j + \sum_{i=1}^{V} v_i w_{ij} + \sum_{k=1}^{Z} z_k u_{jk}\right), \end{aligned} \tag{7.5}$$

where $\sigma(\cdot) = 1/(1+\exp(-\cdot))$ stands for a sigmoid function.

The same assumption may be made for neurons in the visible layer $\boldsymbol{v}$, when values of neurons in the hidden layer $\boldsymbol{h}$ are known. This allows us to calculate the activation probability of the i-th visible neuron as:

$$
\begin{aligned}
P(v_i|\boldsymbol{h}) &= \frac{1}{1+\exp\left(-a_i - \sum_{j=1}^{H} h_j w_{ij}\right)} \\
&= \sigma\left(a_i + \sum_{j=1}^{H} h_j w_{ij}\right),
\end{aligned}
\tag{7.6}
$$

where one must note that given $\boldsymbol{h}$, the activation probability of neurons in $\boldsymbol{v}$ does not depend on $\boldsymbol{z}$. The activation probability of the class layer (i.e., decision on which class the object should be assigned to) is calculated using the softmax function:

$$
P(\boldsymbol{z} = \mathbf{1}_k|\boldsymbol{h}) = \frac{\exp\left(-c_k - \sum_{j=1}^{H} h_j u_{jk}\right)}{\sum_{l=1}^{Z} \exp\left(-c_l - \sum_{j=1}^{H} h_j u_{jl}\right)}, \tag{7.7}
$$

where $k \in [1, \cdots, Z]$ and $k \neq l$.

RBM training procedure. As RBM is a neural network model, we may train it using a loss function $L(\cdot)$ minimization with any gradient descent method. Standard RBM most commonly uses the negative log-likelihood of both external layers $\boldsymbol{v}$ and $\boldsymbol{z}$. However, our RBM-IM architecture must be designed to handle multiple imbalanced classes. Therefore, we need to modify this loss function to make RBM-IM skew-insensitive. We will achieve this by using the effective number of samples approach [226] that measures the contributions of instances in each class. This allows us to formulate a class-balanced negative log-likelihood loss for RBM-IM:

$$
L(\boldsymbol{v}, \boldsymbol{z}) = -\frac{1-\beta}{1-\beta_m^x} \log\left(P(\boldsymbol{v}, \boldsymbol{z})\right), \tag{7.8}
$$

where β_m^x stands for the contribution of x-th instance to the m-th class. By taking each independent weight w_{ij}, we may now calculate the gradient of the loss function:

$$\nabla L(w_{ij}) = \frac{\delta L(\boldsymbol{v},\boldsymbol{z})}{\delta w_{ij}} = \sum_{\boldsymbol{v},\boldsymbol{h},\boldsymbol{z}} P(\boldsymbol{v},\boldsymbol{h},\boldsymbol{z}) v_i h_j - \sum_{\boldsymbol{h}} P(\boldsymbol{h}|\boldsymbol{v},\boldsymbol{z}) v_i h_j. \tag{7.9}$$

This equation allows us to calculate the loss function gradient for a single instance. However, as we use RBM as a drift detector, we must be able to capture the evolving properties of a data stream. If we based our change detection on variations induced by a single new instance, we would be highly sensitive to even the smallest noise ratio. Therefore, our RBM-based drift detector must be able to work with a batch of the most recent instances in order to capture the current stream characteristics. We propose to define RBM-IM model for learning on mini-batches of instances. This will offer significant speed-up when compared to traditional batch learning used in data streams. For a mini-batch of n instances arriving in t time $\boldsymbol{M}_t = [x_1^t, \cdots, x_n^t]$, we can rewrite the gradient from Eq. 7.9 using expected values with loss function:

$$\frac{\delta L(\boldsymbol{M}_t)}{\delta w_{ij}} = E_{\text{model}}[v_i h_j] - E_{\text{data}}[v_i h_j], \tag{7.10}$$

where E_{data} is the expected value over the current mini-batch of instances and E_{model} is the expected value from the current state of RBM-IM. Of course, we cannot trace directly the value of E_{model} (as this would require immediate oracle access to ground truth), therefore we must approximate it using Contrastive Divergence with k Gibbs sampling steps to reconstruct the input data (CD-k):

$$\frac{\delta L(\boldsymbol{M}_t)}{\delta w_{ij}} \approx E_{\text{recon}}[v_i h_j] - E_{\text{data}}[v_i h_j]. \tag{7.11}$$

After processing the t-th mini-batch $\boldsymbol{M}_t$, we can update the wights in RBM-IM using any gradient descent method as follows:

$$w_{ij}^{t+1} = w_{ij}^{t} - \eta \left(E_{\text{recon}}[v_i h_j] - E_{\text{data}}[v_i h_j] \right), \tag{7.12}$$

where η stands for the learning rate of the RBM-IM neural network (responsible for the speed of model update and forgetting of old information). The way to update the a_i, b_j, and c_k biases, as well as weights u_{jk} is analogous to Eq. 7.12 and can be expressed as:

$$a_{i}^{t+1} = a_{i}^{t} - \eta \left(E_{\text{recon}}[v_i] - E_{\text{data}}[v_i] \right), \tag{7.13}$$

$$b_{j}^{t+1} = b_{j}^{t} - \eta \left(E_{\text{recon}}[h_j] - E_{\text{data}}[h_j] \right), \tag{7.14}$$

$$c_{k}^{t+1} = c_{k}^{t} - \eta \left(E_{\text{recon}}[z_k] - E_{\text{data}}[z_k] \right), \tag{7.15}$$

$$u_{jk}^{t+1} = u_{jk}^{t} - \eta \left(E_{\text{recon}}[h_j z_k] - E_{\text{data}}[h_j z_k] \right). \tag{7.16}$$

7.2.2 Drift detection with RBM-IM

While RBM-IM is a skew-insensitive generative neural network model, we can use it as an explicit drift detector. The RBM-IM model stores compressed characteristics of the distribution of data it was trained on. By using any similarity measure between the data prototypes and properties of newly arrived instances, one may evaluate if there are any changes in the distribution. This allows us to use RBM-IM as a drift detector. Our model uses an embedded similarity measure for monitoring the state of a stream and the level to which the newly arrived instances differ from the previously observed concepts. RBM-IM tracks the similarity measure for every single class independently, using the class layer continuous outputs. RBM-IM is a fully trainable and self-adaptive drift detector, capable not only of capturing the trends of changes in each class independently (versus state-of-the-art drift detectors that monitor changes in all classes with an aggregated measure), but also of learning and adapting to the current state of a stream, class imbalance ratios, and class roles. This makes it a highly attractive approach for handling multi-class imbalanced streams with various learning difficulties discussed in Sec. 4.

Measuring data similarity. In order to evaluate the similarity of newly arrived instances to old concepts stored in RBM-IM, we will use the reconstruction error metric. We can calculate it online for each new instance, by inputting a newly arrived d-dimensional instance $\boldsymbol{S}_n = [x_1^n, \cdots, x_d^n, y^n]$ to the $\boldsymbol{v}$ layer of RBM. Then values of neurons in $\boldsymbol{v}$ are calculated to reconstruct the feature values. Finally, class layer $\boldsymbol{z}$ is activated and used to reconstruct the class label. This allows us to keep track of the reconstruction error for each class independently, offering per-class drift detection capabilities. We can denote the reconstructed vector for m-th class as:

$$\tilde{\boldsymbol{S}}_n^m = [\tilde{x}_1^n, \cdots, \tilde{x}_d^n, \tilde{y}_1^n, \cdots, \tilde{y}_Z^n], \tag{7.17}$$

where the reconstructed vector features and labels are taken from probabilities calculated using the hidden layer:

$$\tilde{x}_i^n = P(v_i|h), \tag{7.18}$$

$$\tilde{y}_k^n = P(z_k|h). \tag{7.19}$$

The $\boldsymbol{h}$ layer is taken from the conditional probability, in which the $\boldsymbol{v}$ layer is identical to the input instance:

$$\boldsymbol{h} \sim P(\boldsymbol{h}|\boldsymbol{v} = x^n, \boldsymbol{z} = \mathbf{1}_{y_n}). \tag{7.20}$$

This allows us to write the reconstruction error in a form of the mean squared error between the true and reconstructed instance for the m-th class:

$$R(\boldsymbol{S}_n^m) = \sqrt{\sum_{i=1}^{d} (x_i^n - \tilde{x}_i^n)^2 + \sum_{k=1}^{Z} (\mathbf{1}_k^{y_n} - \tilde{y}_k^n)^2}. \tag{7.21}$$

For the purpose of obtaining a stable concept drift detector, we do not look for a change in distribution over a single instance, but for the change over the newly arriving mini-batch of instances. Therefore, we need to calculate the average reconstruction error over the recent mini-batch of data for the m-th class:

$$R(\boldsymbol{M}_t^m) = \frac{1}{n} \sum_{m=1}^{n} R(x_m^t). \tag{7.22}$$

Adapting reconstruction error to drift detection. In order to make the reconstruction error a practical measure for detecting the presence of concept drift, we propose to measure the evolution of this measure (i.e., its trends) over arriving mini-batches of instances. The analysis of the trends is done for each class independently, allowing us to effectively detect local concept drifts. We achieve this by using the well-known sliding window technique that will move over the arriving mini-batches. Let us denote the trend of reconstruction error for the m-th class over time as $Q_r(t)^m$ and calculate it using the following equation:

$$Q_r(t)^m = \frac{\bar{n^m}_t \bar{TR}_t - \bar{T}_t \bar{R}_t}{\bar{n}_t \bar{T^2}_t - (\bar{T}_t)^2}. \tag{7.23}$$

The trend over time can be computed using a simple linear regression, with the terms in Eq. 7.23 being simply sums over time as follows:

$$\bar{TR}_t = \bar{TR}_{t-1} + tR(\boldsymbol{M}_t^m), \tag{7.24}$$

$$\bar{T}_t = \bar{T}_{t-1} + t, \tag{7.25}$$

$$\bar{R}_t = \bar{R}_{t-1} + R(\boldsymbol{M}_t^m), \tag{7.26}$$

$$\bar{T^2}_t = \bar{T^2}_{t-1} + t^2, \tag{7.27}$$

where $\bar{TR}_0 = 0$, $\bar{T}_0 = 0$, $\bar{R}_0 = 0$, and $\bar{T^2}_0 = 0$. We capture those statistics for each class using a sliding window of size W. Instead of using a manually set size, which is inefficient for drifting data streams, we propose to use a self-adaptive window size [108]. This eliminates the need for manual tuning of the window size that is used for drift detection. To allow flexible learning from various sizes of mini-batches, we must consider a case where $t > W$. Here, we must compute the terms for the trend regression using the following equations:

$$\bar{TR}_t = \bar{TR}_{t-1} + tR(\boldsymbol{M}_t) - (t-w)R(\boldsymbol{M}^m_{t-W}), \tag{7.28}$$

$$\bar{T}_t = \bar{T}_{t-1} + t - (t-W), \tag{7.29}$$

$$\bar{R}_t = \bar{R}_{t-1} + R(\boldsymbol{M}_t) - R(\boldsymbol{M}^m_{t-W}), \tag{7.30}$$

$$\bar{T^2}_t = \bar{T^2}_{t-1} + t^2 - (t-W)^2. \tag{7.31}$$

The required number of instances $\bar{n}^m_t$ to compute the trend of $Q_r(t)^m$ for m-th class as time t is given as follows:

$$\bar{n}_t = \begin{cases} t & \text{if } t \leq W \\ W & \text{if } t > W. \end{cases} \tag{7.32}$$

Drift detection. The above Eq. 7.23 allows us to compute the trends for every analyzed mini-batch of data. In order to detect the presence of drift we need to have the capability of checking if the new mini-batch differs significantly from the previous one for each analyzed class. Our RBM-IM uses Granger causality test [227] on trends from subsequent mini-batches of data for each class $Q_r(\boldsymbol{M}^m_t)$ and $Q_r(\boldsymbol{M}^m_{t+1})$. This is a statistical test that determines whether one trend is useful in forecasting another. As we deal with non-stationary processes we perform the variation of Granger causality test based on first differences [228]. Accepted hypothesis means that it is assumed that there exists a Granger causality relationship between $Q_r(\boldsymbol{M}^m_t)$ and $Q_r(\boldsymbol{M}^m_{t+1})$, which means there is no concept drift on the m-th class. If the hypothesis is rejected, RBM-IM signals the presence of concept drift on the m-th class.

7.3 Experimental study

In this section, we present the experimental study used to evaluate the quality of RBM-IM. It was carefully designed to offer an in-depth analysis of the proposed method and gain insights into its behavior in various multi-class imbalanced data stream scenarios. We tailored this study to answer the following research questions.

- **RQ1:** Does RBM-IM offer better concept drift detection than state-of-the-art drift detectors designed for standard data streams?
- **RQ2:** Does RBM-IM offer better concept drift detection than state-of-the-art skew-insensitive drift detectors designed for imbalanced data streams?
- **RQ3:** What is the capability of RBM–IBM to detect local drifts that affect a subset of minority classes?
- **RQ4:** What robustness to increasing imbalance ratio is offered by RBM-IM?

All methods and experiments were implemented in MOA environment [214] and run on Intel Core i7-8365u with 64GB DDR4 RAM.

7.3.1 Data stream benchmarks

For the purpose of this experimental study, we selected 24 benchmark data streams: 12 come from real-world domains and 12 were generated artificially using the MOA environment [214]. Such a diverse mix allowed us to evaluate the effectiveness of RBM-IM over a plethora of scenarios. Using artificial data streams allows us to control the specific nature of drift and class imbalance, as well as to inject local concept drift into selected minority classes. Artificial data streams use a dynamic imbalance ratio that both increases and decreases over time. Real-world streams offer challenging problems that are characterized by a mix of different learning difficulties. Properties of the data stream benchmarks are given in Tab. 18. We report the highest imbalance ratio among all the classes, i.e., the ratio between the biggest and the smallest class.

7.3.2 Setup

Reference concept drift detectors. As reference methods to the proposed **RBM-IM**, we have selected three state-of-the-art concept drift detectors for standard data: **WSTD** [130], **RDDM** [128], and **FHDDM** [134]; as well as two state-of-the-art drift detectors for imbalanced data streams: **PerfSim** and **DDM-CI**. Parameters of all the six drift detectors are given in Tab. 19.

Parameter tuning. In order to offer a fair and thorough comparison, we performed parameter tuning for every drift detector and for every data stream benchmark. As we deal with a streaming scenario, we used self hyper-parameter tuning [231] that is based on the online Nelder-Mead optimization.

Base classifier. In order to ensure fairness when comparing the examined drift detectors they all use Adaptive Cost-Sensitive Perceptron Trees [201] as a base classifier. This is a skew-insensitive and efficient classifier capable of handling both binary and multi-class imbalanced data streams, but is strongly dependent on an attached concept drift detection component. Therefore, it offers an excellent backbone for our experiments, allowing us to directly measure how a given drift detector impacts the classification quality.

RBM-IM training. Our drift detector uses the first instance batch to train itself

Table 18.: Properties of real-world (top) and artificial (bottom) imbalanced data stream benchmarks.

Dataset	Instances	Attr	Cls	IR	Drift
Activity-Raw	1 048 570	3	6	128.93	yes
Connect4	67 557	42	3	45.81	unknown
Covertype	581 012	54	7	96.14	unknown
Crimes	878 049	3	39	106.72	unknown
DJ30	138 166	8	30	204.66	yes
EEG	14 980	14	2	29.88	yes
Electricity	45 312	8	2	17.54	yes
Gas	13 910	128	6	138.03	yes
Olympic	271 116	7	4	66.82	unknown
Poker	829 201	10	10	144.00	yes
IntelSensors	2 219 804	5	57	348.26	yes
Tags	164 860	4	11	194.28	unknown
Aggrawal5	1 000 000	20	5	50.00	incremental
Aggrawal10	1 000 000	40	10	80.00	incremental
Aggrawal20	2 000 000	80	20	100.00	incremental
Hyperplane5	1 000 000	20	5	100.00	gradual
Hyperplane10	1 000 000	40	10	200.00	gradual
Hyperplane20	2 000 000	80	20	300.00	gradual
RBF5	1 000 000	20	5	100.00	sudden
RBF10	1 000 000	40	10	200.00	sudden
RBF20	2 000 000	80	20	300.00	sudden
RandomTree5	1 000 000	20	5	100.00	sudden
RandomTree10	1 000 000	40	10	200.00	sudden
RandomTree20	2 000 000	80	20	300.00	sudden

at the beginning of the stream processing. It continuously updates itself in an online fashion together with the base classifier.

Evaluation metrics. As we deal with multi-class imbalanced and drifting data streams, we evaluated the examined algorithms using prequential multi-class AUC

Table 19.: Examined drift detectors and their parameters.

Abbr.	Name	Parameters
WSTD [130]	Wilcoxon Rank Sum Test	sliding window size $\omega \in \{25, 50, 75, 100\}$
	Drift Detection	warning significance $\alpha_w \in \{0.01, 0.03, 0.05, 0.07\}$
		drift significance $\alpha_d \in \{0.001, 0.003, 0.005, 0.007\}$
		max. no of old instances min $\in \{1000, 2000, 3000, 4000\}$
RDDM [128]	Reactive Drift Detection	warning threshold $\alpha_w \in \{0.90, 0.92, 0.95, 0.98\}$
		drift threshold $\alpha_d \in \{0.80, 0.85, 0.90.0.95\}$
		min. no. of errors $e \in \{10, 30, 50, 70\}$
		min. no. of instances min $\in \{3000, 5000, 7000, 9000\}$
		max. no. of instances max $\in \{10000, 20000, 30000, 40000\}$
		warning limit $wL \in \{800, 1000, 1200, 1400\}$
FHDDM [134]	Fast Hoeffding Drift Detection	sliding window size $\omega \in \{25, 50, 75, 100\}$
		allowed error $\delta \in \{0.000001, 0.00001, 0.0001, 0.001\}$
PerfSim [229]	Performance Similarity	differentiation weights $\lambda \in \{0.1, 0.2, 0.3, 0.4\}$
		min. no. of errors $n = \{10, 30, 50, 70\}$
DDM–CI [230]	Drift Detection Method	warning threshold $\alpha_w \in \{0.90, 0.92, 0.95, 0.98\}$
	for online class imbalance	drift threshold $\alpha_d \in \{0.80, 0.85, 0.90.0.95\}$
		min. no. of errors $e \in \{10, 30, 50, 70\}$
RBM-IM	RBM Drift Detection	mini–batch size $\boldsymbol{M} \in \{25, 50, 75, 100\}$
	for imbalanced data streams	visible neurons $\boldsymbol{V} =$ no. of features
		hidden neurons $\boldsymbol{H} \in \{0.25\boldsymbol{V}, 0.5\boldsymbol{V}, 0.75\boldsymbol{V}, \boldsymbol{V}\}$
		class neurons $\boldsymbol{Z} =$ no. of classes
		learning rate $\eta \in \{0.01, 0.03, 0.05, 0.07\}$
		Gibbs sampling steps $k \in \{1, 2, 3, 4\}$

(pmAUC) [230] and prequential multi-class G-mean (pmGM) [232].

Windows. We used a window size $W = 1000$ for calculating the prequential metrics. ADWIN self-adapting window was used for both RBM-IM and reference drift detectors to alleviate the need for manual window size tuning [233].

Statistical analysis. We used the Friedman ranking test with Bonferroni-Dunn post-hoc and Bayesian signed test [234] for statistical significance over multiple comparison with significance level $\alpha = 0.05$.

Drift injection. For experiment 2, we inject local concept drift starting with the smallest minority class and then add classes according to their increasing size. This allows us to consider the most difficult scenarios, where the smallest classes are affected by the local concept drift and thus most likely to be neglected.

7.3.3 Experiment 1: Drift detectors comparison

The first experiment was designed to analyze the behavior of the six examined drift detectors under two different metrics measured on all 24 benchmark data streams. This will allow us to evaluate how competitive is RBM-IM as compared with the state-of-the-art reference methods. Results according to pmAUC and pmGM are

Table 20.: Results according to pmAUC and pmGM for the examined concept drift detectors.

Dataset	pmAUC						pmGM					
	WSTD	RDDM	FHDDM	PerfSim	DDM–CI	RBM-IM	WSTD	RDDM	FHDDM	PerfSim	DDM–CI	RBM-IM
Activity-Raw	45.43	46.23	48.45	72.81	74.29	**79.92**	51.06	54.10	55.82	76.11	78.59	**82.04**
Connect4	54.19	53.48	55.27	64.19	69.10	**75.04**	55.03	55.39	56.29	66.08	70.21	**77.92**
Covertype	33.19	34.12	35.72	41.24	40.58	**53.98**	32.45	33.10	35.98	40.19	41.02	**54.02**
Crimes	19.93	20.04	22.11	28.56	30.02	**64.59**	21.88	23.92	26.01	30.99	32.07	**69.58**
DJ30	26.94	25.98	26.02	34.11	33.98	**59.04**	27.45	27.11	28.73	36.71	35.48	**61.29**
EEG	58.14	59.98	62.29	70.08	**74.22**	72.03	59.85	60.98	64.67	72.93	**77.29**	74.13
Electricity	68.94	72.10	73.45	80.04	**83.20**	79.39	70.45	75.90	77.28	83.92	**85.44**	81.99
Gas	48.83	47.23	46.92	63.59	**67.54**	64.20	50.05	49.54	49.17	65.98	**70.02**	66.13
Olympic	72.98	70.34	74.53	80.08	83.19	**87.01**	73.95	71.91	76.02	83.19	86.88	**89.24**
Poker	72.11	69.65	72.98	84.65	87.91	**91.03**	74.46	70.97	74.52	87.11	89.34	**93.06**
IntelSensors	9.45	11.45	13.99	36.23	37.08	**58.10**	10.02	13.01	14.38	37.82	38.03	**60.39**
Tags	30.45	28.67	29.45	**42.68**	40.18	39.04	33.10	30.08	31.14	**45.28**	43.21	41.02
Aggrawal5	78.34	77.45	80.41	84.92	88.34	**90.38**	77.19	79.02	80.93	85.99	90.02	**93.01**
Aggrawal10	70.12	68.34	70.23	74.99	78.32	**88.02**	71.04	70.16	71.88	75.38	79.14	**90.49**
Aggrawal20	55.62	56.23	58.93	65.76	66.98	**83.87**	56.45	57.22	59.39	66.28	67.57	**85.09**
Hyperplane5	62.05	63.66	62.07	70.45	73.98	**75.06**	65.39	67.20	66.14	74.82	78.05	**81.80**
Hyperplane10	53.56	54.37	54.02	63.74	66.59	**72.30**	56.93	59.14	57.92	66.72	70.56	**78.03**
Hyperplane20	40.04	38.45	42.19	50.10	57.67	**66.48**	42.06	41.99	40.86	52.19	59.37	**68.27**
RBF5	80.18	78.56	82.40	90.48	92.36	**92.78**	83.47	81.59	84.99	92.12	94.82	**94.97**
RBF10	69.45	67.84	73.29	82.19	84.48	**88.82**	72.19	70.48	76.44	85.11	87.81	**90.26**
RBF20	53.18	52.88	54.01	70.24	71.93	**83.08**	55.98	54.90	57.73	73.89	74.84	**85.30**
RandomTree5	45.29	47.21	47.93	58.90	64.32	**67.98**	46.12	48.52	49.11	60.05	66.30	**69.93**
RandomTree10	31.63	33.19	35.02	50.02	53.87	**63.01**	32.79	33.90	36.14	51.58	55.20	**64.97**
RandomTree20	19.83	20.04	21.38	36.29	43.22	**59.42**	20.02	20.88	22.94	38.01	44.87	**60.33**
ranks	5.46	4.78	3.84	2.97	2.56	1.39	5.80	5.05	4.15	2.45	2.29	1.26

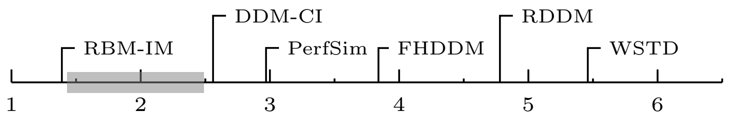


Fig. 46.: Bonferroni-Dunn test (pmAUC).

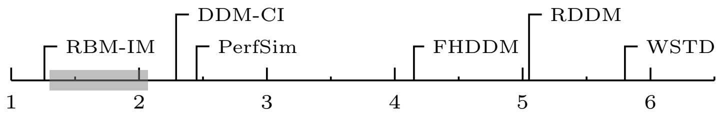


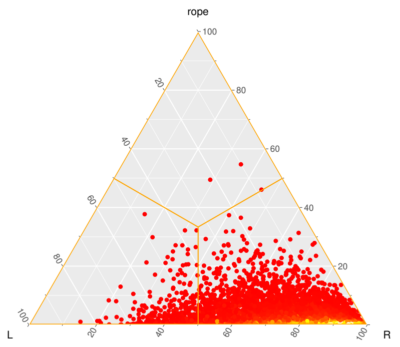
Fig. 47.: Bonferroni-Dunn test (pmGM).

given in Tab. 20, Fig. 46 and 47 depict the outcomes of the post-hoc statistical tests of significance, while Fig.21 presents average processing time per batch. Fig. 48 and 49 present visualizations of the Bayesian signed test for pairwise comparisons with two best performing reference detectors.

Comparison with standard drift detectors. The standard drift detectors return unsatisfactory performance for all of the examined multi-class imbalanced data streams. This shows that the metrics collected by them are unsuitable to monitor skewed data streams. This also indicates that drift detectors, despite not being actually trainable models, are still prone to class imbalance. Despite the fact that the underlying classifier used was designed for imbalanced data streams, it could not offer accurate predictions when being fed incorrect information from the drift detectors. Especially in the case of datasets with a high number of classes (such as Crimes, DJ20,

Table 21.: Results according to average processing times [s] per batch for the examined concept drift detectors.

Time	WSTD	RDDM	FHDDM	PerfSim	DDM–CI	RBM-IM
Test	17.26±3.11	18.11±4.72	16.54±2.98	8.92±3.07	9.78±4.14	6.28±1.08
Update	0.02±0.01	0.08±0.02	0.11±0.05	19.83±6.98	18.54±7.82	12.22±0.92



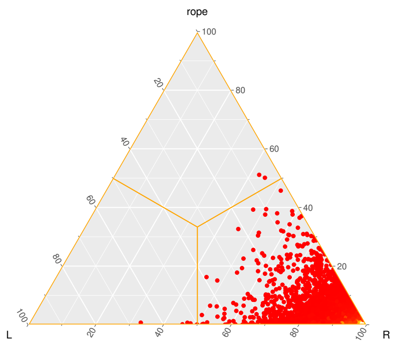
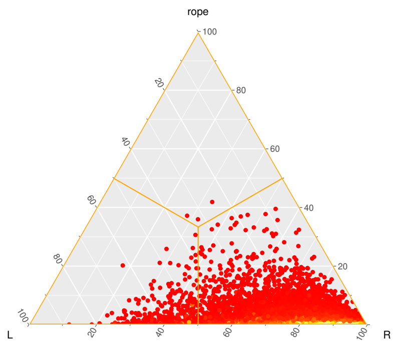


Fig. 48.: Visualizations of the Bayesian signed test for comparison between PerfSim and RBM-IM for pmAUC (left) and pmGM (right).

IntelSensor, or the artificial ones) standard drift detectors returned performance only slightly above a random guess. Those detectors were not capable of capturing changes affecting at the same time multiple class distributions and imbalance ratios. RBM-IM alleviated those limitations while displaying comparable computational complexity.

Answer to RQ1: Yes, RBM-IM offers significant improvements over standard drift detectors when applied to monitoring multi-class imbalanced data streams. Standard detectors cannot handle both a high number of classes and simultaneous changes in distributions and imbalance ratios. This shows that we need to have dedicated drift detectors for such difficult scenarios.

Comparison with skew-insensitive drift detectors. Skew-insensitive detectors performed significantly better when compared with their standard counterparts. However, for most of the real-world benchmarks and for all the artificial ones they still could not compete with RBM-IM. The only four datasets on which they returned a slightly better performance were EEG, Electricity, Gas, and Tags. All of them are relatively small and have a low number of classes. Especially the former factor



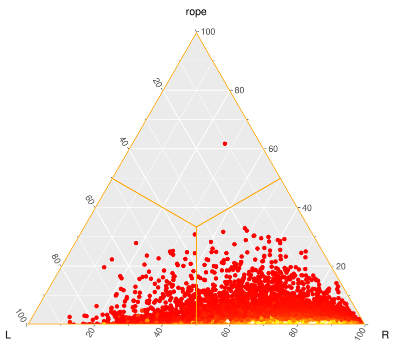


Fig. 49.: Visualizations of the Bayesian signed test for comparison between DDM-CI and RBM-IM for pmAUC (left) and pmGM (right).

might have had a strong impact on RBM-IM. As this is a trainable drift detector, it probably suffered from the problem of underfitting when learning from small data streams. This could be potentially alleviated by combining RBM-IM with transfer learning or instance exploitation techniques, which we will investigate in our future works. For all the remaining 20 data stream benchmarks RBM-IM outperformed in a statistically significant manner both PerfSim and DDM-CI. This can be contributed to the compressed information about the current concept for each class stored within the RBM-IM structure, which allowed for a significantly more informative analysis of the changing properties of incoming instances.

Answer to RQ2: Yes, RBM-IM is capable of outperforming state-of-the-art skew-insensitive drift detectors, while additionally offering faster detection and update times. This is especially visible on datasets with a high number of classes, where monitoring simple performance measures is not enough to accurately and timely detect occurrences of drifts. Moreover, by being a trainable detector RBM-IM can better adapt to changes in data streams, allowing fine-tuned encapsulation of the definition of what currently is considered a temporal concept.

7.3.4 Experiment 2: Detection of local concept drifts

This experiment was designed to understand if and how the examined drift detectors can handle the appearance of local concept drifts on top of changing imbalance ratios and class roles (see Sec. 3 – Scenario 3 for more details). We carried this experiment only on artificial benchmarks, as they allowed us to directly inject concept drift into a selected number of classes. We evaluated how the performance of drift detectors changes with the decrease in the number of classes being affected by the concept drift. For each of the 12 benchmark data streams, we created scenarios where from 1 to M classes are being affected by the drift, the M case standing for every single class in the stream being subject to the concept drift. Fig. 50 depicts the behavior of all the six drift detectors under various levels of the local concept drift for the pmAUC metric. We do not show plots for pmGM as they have very similar characteristics and would not provide any additional insights. Please note that the smaller number of classes that are subject to concept drift, the more difficult its detection becomes.

Comparison with standard drift detectors. Unsurprisingly, standard detectors completely failed when facing the task of local drift detection. When the number of classes subject to concept drift dropped below 80%, we could see significant drops in their pmAUC. When the number of affected classes dropped below 50%, all three detectors started to completely ignore the presence of any drift. This crucially impacted the underlying classifier that lost any adaptation capabilities, as drift detectors were never signaling any change being present. Such results clearly support our earlier statement that standard drift detectors cannot handle local changes, as statistics they monitor relate to the entire stream, not specific classes. Furthermore, in the case of imbalanced multi-class drifting streams, the underlying bias toward the majority

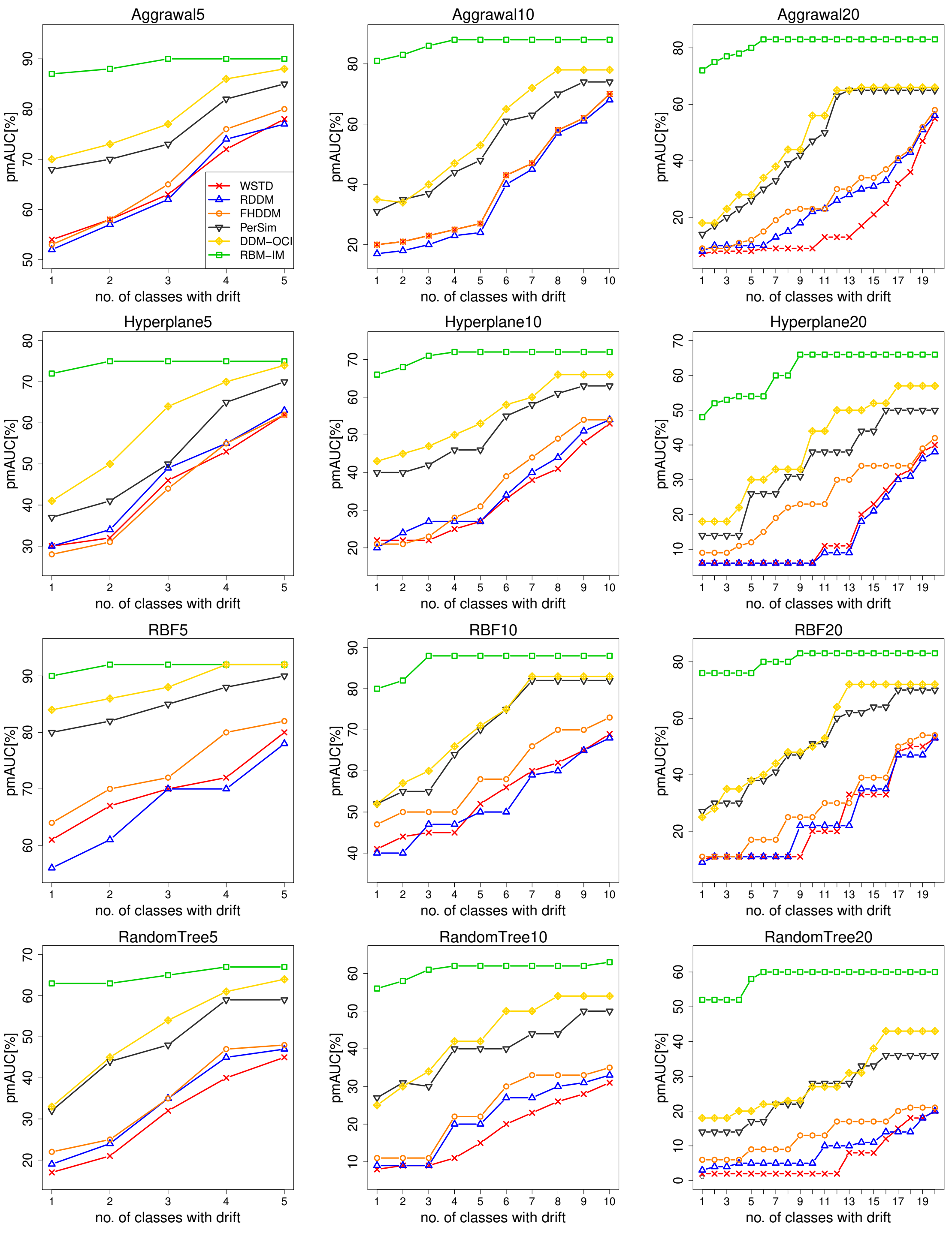


Fig. 50.: Relationship between pmAUC and the number of classes affected by the local drift for the artificial benchmarks. The lower the number of classes subject to concept drift, the more difficult its detection.

class had a strong impact on those statistics. This damaged the reactivity of those detectors to an even greater degree, as changes happening in minority classes were obscured by static properties of the majority class.

Comparison with skew-insensitive drift detectors. This experiment showed the weak side of the skew-insensitive drift detectors published so far. While they can display some robustness to changing class ratios and global concept drift, they did not perform significantly better than standard detectors when facing local drifts. For more than 90% of classes being affected by drift, both PerfSim and DDM-CI returned satisfactory performance. Their quality started degrading when less than 70% of classes were being affected, reaching the lowest plateau for less than 30% of classes being affected. This shows that despite the fact of monitoring some performance metrics for each class (like DDM-CI monitors recall) they do not extract strong enough properties of those classes to properly detect local drifts. Only when the majority of classes become subject to concept drift those detectors can pick up local changes.

RBM-IM sensitivity to local drifts. RBM-IM displayed an excellent sensitivity to local drifts, even when they affected only a single class. This observation holds for any dataset, any imbalance ratio, and any total number of classes. This can be contributed to the effectiveness of the reconstruction error, used as a change detection metric, combined with storing compressed information about each class independently, and being able to compare reconstruction error for each class individually. This allows RBM-IM to detect local drifts that at a given moment affect any number of classes.

Answer to RQ3: RBM-IM is the only drift detector among the examined ones that can correctly detect local concept drifts, even when they affect only a single minority class. This allows to gain a better understanding of what is the exact nature of changes affecting the data stream and which classes should be more carefully analyzed to discover useful knowledge. This RBM-IM's capability of offering at the same time global and local concept drift detection is a crucial step towards explainable drift detection and gaining deeper insights into dynamics behind data streams, especially those imbalanced.

7.3.5 Experiment 3: Robustness to changing imbalance ratio

The third experiment was designed for evaluating the robustness of the examined drift detectors to changing imbalance ratio, especially for extremely imbalanced cases (IR > 400). This will allow us to test the flexibility and trustworthiness of skew-insensitive mechanisms used in the detectors and to see how reliable they are. For each of 12 benchmark data streams, we created scenarios in which we generate varying imbalance ratios from 50 to 500. Fig. 51 depicts the behavior of the six drift detectors under various levels of class imbalance for the pmAUC metric. We do not show plots for pmGM as, analogously to the previous experiment, they are very similar.

Analyzing robustness to changing imbalance ratios. As expected the standard drift detectors cannot handle any class imbalance ratios and do not return any acceptable results, omitting drift detection. This can be seen in the extremely poor performance of the underlying classifier that stopped being updated and could not handle new incoming concepts. Two reference skew-insensitive detectors maintain acceptable robustness to small and medium imbalance ratios (IR < 200), but start to critically fail with further increasing IR. At extreme levels of IR their performance becomes similar to standard detectors. This shows that none of the existing detectors can handle high imbalance ratios in multi-class data streams. RBM-IM offers excellent and stable robustness, filling the gap and providing a sought-after robust drift detection approach. We can contribute this to a combination of the used loss function and the ability of RBM-IM to continually learn from the stream. This is a

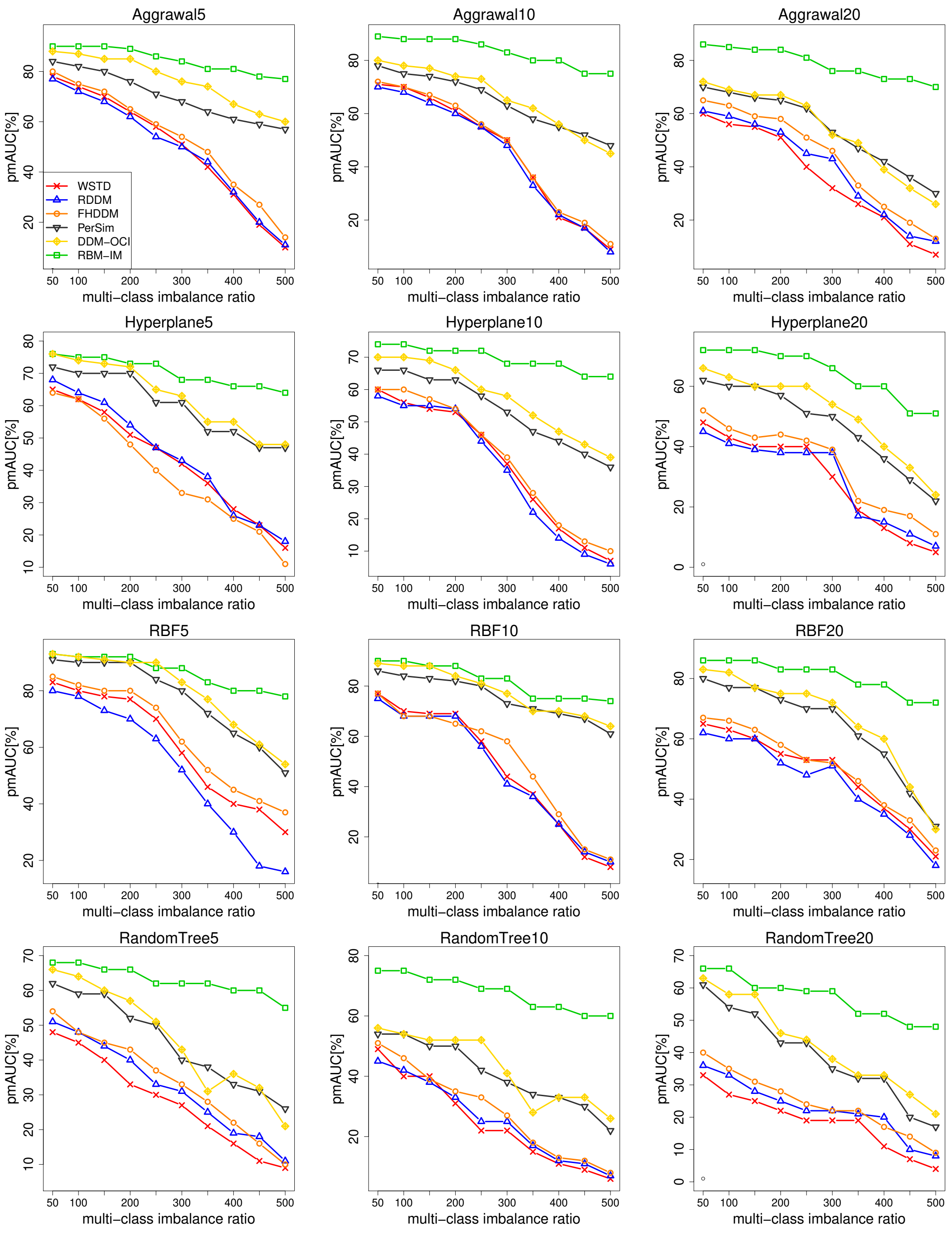


Fig. 51.: Relationship between pmAUC and changing imbalance ratio for the artificial benchmarks. The higher the imbalance ratio, the higher the disproportions among multiple classes.

massive advantage, as all other drift detectors are using some preset rules for deciding if the drift is present or not. RBM-IM can learn the current distribution in a skew-insensitive manner, making its drift detection much more accurate and not affected by the imbalance ratio.

Answer to RQ4: RBM-IM offers excellent robustness to various levels of dynamic imbalance ratio in multi-class scenarios. Due to its trainable nature, RBM-IM is capable of quickly adapting to the current state of any stream and re-aligning its own structure regarding class ratios and class roles. This is the only drift detector displaying robustness to extremely high levels of class imbalance (IR > 400).

7.4 Lessons learned

Let us now present a short summary of insights and conclusions that were drawn from both the theoretical and experimental parts of this work.

Unified view on challenges in imbalanced multi-class data streams. Continual learning from non-stationary and skewed multiple distributions is a challenging topic that requires more attention from the research community. It offers an excellent field for developing and evaluating novel learning algorithms while calling for enhancing our models with various valuable robust characteristics. Three mutually complementary scenarios were identified by us, each dealing with different learning difficulties embedded in the nature of data. One must remember that in multi-class imbalanced streams both distributions, imbalance ratios, and class roles may change over time and our models must be capable of swift adaptation to such evolving data.

Advantages of trainable drift detector. To the best of our knowledge, the existing state-of-the-art drift detectors are realized as external modules that track some properties of the stream and use them to decide if a drift should be detected or not. However, those models use static rules for determining the degree of change that constitutes drift presence. This significantly limits them in capturing the unique properties of each concept and thus may negatively impact their reactivity to changes. We propose to use a trainable drift detector that can extract and store the most important characteristics of the current state of the stream and use them to make an informative and guided decision on deciding whether the underlying classifier should be retrained or not.

Handling global and local drifts. Most of the works in drift detection focus on detecting global drifts that affect the entire stream. Detectors gather information from every single instance and use those statistics to make a decision. However, this makes them less sensitive to local drifts that affect only certain classes. The situation becomes even more challenging when combined with multi-class imbalanced distributions. Here, local drifts affecting the minority classes would go unnoticed, as gathered statistics will be biased towards the majority classes. This shows the importance of monitoring each individual class for local changes. This also provides us with valuable insights into the nature of concept drifts and helps us understand dynamics of changes. RBM-IM stores information about each class independently, allowing for precise drift detection even if it affects only a single minority class.

Impact of class imbalance on drift detection. Not enough attention has been given to the interplay between the concept drift and class imbalance. We observed that imbalanced distributions will directly affect each drift detector in two possible ways: (i) enhancing the presence of small changes in the majority classes; and (ii) diminishing the importance of changes in the minority classes. The former problem is caused by statistics gathered from more abundant classes that will dominate the detector and thus may cause false alarms, as even small changes will be magnified by the sheer disproportion among classes. The latter problem is caused by the minority classes not contributing enough to the drift detector statistics and thus not being able to trigger it to cause an alarm. We showed that by enhancing RBM-IM with a skew-insensitive loss function we are able to handle a high range of imbalance ratios in multi-class data streams.

7.5 Summary

In this chapter, we have discussed an important area of learning from multi-class imbalanced data streams under concept drift. We proposed a unifying taxonomy of challenges that may be encountered when learning from such data, and identified three realistic scenarios representing various types of learning difficulties. This was the first complete attempt to understand and organize challenges arising in this area of machine learning. We introduced RBM-IM, a novel and trainable drift detector for monitoring changes for continual learning from multi-class imbalanced data streams. Our research was motivated by an apparent lack of drift detection methods designed for skewed multi-class and evolving streams. We developed our drift detector on the basis of the Restricted Boltzmann Machine neural network with a skew-sensitivities loss function. We used it to store compressed information about each class independently and use the reconstruction error over mini-batches of data to detect concept drift per class. This, combined with the loss function robust to imbalanced data, allowed RBM-IM to be highly sensitive and reactive to local concept drifts that affect only a small subset of minority classes.

In our future works, we plan to combine RBM-IM with techniques for handling underfitting (to make it applicable to small data streams), as well as make it robust to adversarial concept drifts that may be injected by a malicious party as a poisoning attack.

CHAPTER 8

UNSUPERVISED DRIFT DETECTOR ENSEMBLES FOR DATA STREAM MINING

The blind adaptation to an incoming data stream may very easily turn out to be impractical, if we take into consideration a substantial constraint put on every supervised adaptive machine learning model – limited labeling budget [163]. Also, spending the budget on new data points, when there is no change, will inevitably lead to high unnecessary costs that could be avoided. On the other hand, informed methods update a model only when it is really needed, so after a drift occurred. In such a setting, a reliable drift detector is a core element of such a system.

Unsupervised detectors do not require any additional labels, so by utilizing them we may avoid the inconvenience of *spending budget for saving it*, which occurs in the case of the supervised drift detectors. Furthermore, by applying more sophisticated and meticulous detection [141] we may guide our adaptation even more precisely and, as a result, achieve even more reasonable budget spending. In this context, interesting ideas include local detection, which keeps attention to regional changes in features [235] and ensemble techniques, which have been proven to be very effective in cases when we have multiple weak models [22]. The latter may be the case when we have to limit our labeling and we are forced to rely on uncertain information from unlabeled data.

In this chapter, we propose an unsupervised incremental detector that reliably recognizes changes in feature subspaces, utilizing ensemble techniques. To the best of our knowledge, it is the first attempt to use unsupervised ensembles of detectors for the purpose of local detection. We address the problem of budget spending by including the whole detector in a streaming framework in which an adaptive classifier is updated with some portion of labeled instances only when a substantial local change is detected by the committees.

8.1 Detection under labeling constraints

Strictly unsupervised drift detectors aim at finding changes only in unlabeled data without requiring additional supervision. Such approaches are usually implemented in a form of a statistical comparison of two samples of data. Well-known tests like Kolmogorov-Smirnov, two-sample t-test, Wilcoxon rank sum or Wald-Wolfowitz can be used directly in a univariate way, by applying them to individual features and combining the results [139]. In fact, we create an ensemble of detectors by doing this, and various combinations of unsupervised detectors can be used for this purpose [135]. Another interesting framework (LDCNet) has been proposed in [236]. This algorithm utilizes prior information about possible data distributions and using an ensemble of unsupervised detectors, decides if a current stream switched to one of the given concepts.

The main problem with such approaches is that they ignore dependencies between features (univariate tests), making themselves susceptible to increased false positive rates. On the other hand, using much more complex multivariate approaches may be prohibitive in the data streams domain, since even simple univariate methods tend to be significantly slower than the supervised algorithms [237]. An intermediate solution based on feature subspaces may be a good approach to balancing quality and complexity.

We say that as opposed to the time-based detectors, which focus on finding a moment of a drift, region-based detectors are oriented on the spatial search. In general, these algorithms monitor local partitions of data and detect dissimilarities between old and recent states. One of such algorithms is NN-DVI [235], which applies a nearest-neighbors-based partitioning and checks whether distribution within each created partition is stationary. Various partitioning schemes [140] or dissimilarity measures [141] can be applied. The most crucial aspect of such methods is the fact that we get very precise information about drifts, namely their localization, and we can utilize it to intensify adaptation in specific regions [141]. Having such an insight into data may also help us with maintaining a high quality of detection and, by limiting false positives, it may limit the number of labeled instances we ask for. It is also worth mentioning that the mentioned ensembles of univariate detectors can be treated as a special case of spatial algorithms – in this case, partitioning is defined by features.

Our approach focuses on combining the potential strength of ensembles with local detection. We build our fully unsupervised solution based on the incremental Kolmogorov-Smirnov test [238]. In order to limit the number of false positives generated by sensitive univariate detections and at the same time to avoid complex multivariate computations, we encapsulate multiple detectors in ensembles specialized in different diversified feature subspaces. In the following sections, we describe the motivations for our choices in more detail, as well as present an empirical evaluation of our algorithm.

8.2 Incremental Kolmogorov-Smirnov test

In this section, we will present the standard Kolmogorov-Smirnov test and discuss its incremental version and how can it be applied for the purpose of unsupervised drift detection.

Kolmogorov-Smirnov test. Kolmogorov-Smirnov (KS) test assumes that two samples A and B contain univariate observations. KS aims at deciding if we can reject the null hypothesis that both A and B originate from the same distribution, with significance level α. KS requires no a priori information about data distribution, apart from the assumption that data is i.i.d.

The null hypothesis can be rejected at level α if the following is satisfied:

$$D > c(\alpha)\sqrt{\frac{n+m}{nm}}, \tag{8.1}$$

where $c(\alpha)$ can be retrieved from a known statistical table, while n and m are the number of observations in A and B respectively. On the right side of this inequality we have the target p-value, while D is the Kolmogorov-Smirnov statistic (obtained p-value). It can be calculated as follows:

$$D = \sup_x |F_A(x) - F_B(x)|, \tag{8.2}$$

where

$$F_C(x) = \frac{1}{|C|}\sum_{c \in C, c \leq x} 1. \tag{8.3}$$

And it is important to note that there exist an efficient way of computing D:

$$D = \max_{x \in A \cup B} |F_A(x) - F_B(x)|. \tag{8.4}$$

KS test is a popular method for analyzing univariate samples and finds common applications in machine learning and data mining. However, it does not work with data streams and thus must be adapted to incremental domains.

Incremental Kolmogorov-Smirnov test. This version of KS assumes that A and B may change over time, thus fulfilling requirements for data stream mining methods. Incremental Kolmogorov-Smirnov (IKS) test [238] creates a data structure storing the current state of samples and is based on five operations: (i) inserting new observation into A; (ii) inserting new observation into B; (iii) removing old observation from A; (iv) removing old observation from B; (v) performing KS test. The first four operations must be done in a logarithmic time to the total number of observations to make it applicable to data streams. The fifth operation time is constant.

IKS uses $|A|$ and $|B|$ along m and n. The distinction lies in $|A|$ and $|B|$ standing for the number of observations inserted into the data structure, while m and n stand for the total number of elements in these samples. This allows for controlling how many instances incoming from the data stream are actually used to update A and B, thus allowing for discarding potentially noisy or irrelevant observations.

IKS assumes that D is being computed in situations when $|A| = r|B|$, where $r \in \mathbb{R}$ is a constant parameter used during stream processing. This allows for setting the relationships between the two samples being compared, which in data stream context are realized as two sliding windows – one storing instances from the previous stream state and the other with newly arriving instances. As in most cases, we want both windows to be of identical size, then $r = 1$. However, changing the relationships between the sizes of those two windows may be useful for detecting specific types of drifts.

We can rewrite Eq. 8.4 to incorporate $|A| = r|B|$:

$$D = \frac{1}{|A|} \max_{x \in A \cup B} |F'_A(x) - F'_B(x)|, \tag{8.5}$$

where F'_A is a sum of ones:

$$F'_A(x) = \sum_{a \in A, a \leq x} 1, \tag{8.6}$$

and F'_B is a sum of r's:

$$F'_B(x) = \sum_{b \in B, b \leq x} r. \tag{8.7}$$

Furthermore, we can define $G(x) = F'_A(x) - F'_B(x)$, thus obtaining a new formulation of D:

$$D = \frac{1}{|A|} \max \left\{ \left(\max_{x \in A \cup B} G(x) \right), - \left(\min_{x \in A \cup B} G(x) \right) \right\} \tag{8.8}$$

We should have a data structure (e.g., array) that stores all the observations $o_i \in A \cup B$, sorted that $o_i \leq o_{i+1}$ [238]. Additionally, for each observation o_i we have a corresponding value $g_i = G(o_i)$. Such a data structure is depicted in Table 22.

Table 22.: Array of $G(x)$ for each observation $x \in A \cup B$, sorted so $o_i \leq o_{i+1}$.

Index	1	2	$\cdots$	$\|A\| + \|B\| - 1$	$\|A\| + \|B\|$
o_i	o_1	o_2	$\cdots$	$o_{\|A\|+\|B\|-1}$	$o_{\|A\|+\|B\|}$
$G(o_i)$	g_1	g_2	$\cdots$	$g_{\|A\|+\|B\|-1}$	$g_{\|A\|+\|B\|}$

IKS assumes that each new observation is inserted into the data structure following an order $o_{i-1} < o_i \leq o_{i+1}$ [238]. This means that all older observations with the same or higher value are kept on the right side of the new observation. This in consequence leads to $g_i = g_{i-1} + v$, where $v = 1$ if $o_i \in A$ or $v = -r$ if $o_i \in B$. Adding a new observation or removing an old one from such an IKS data structure can be done in $O(\log |A| + |B|)$.

Drift detection. IKS is a univariate test and thus must be applied independently on each feature describing an instance. This is done by storing two sliding windows, one for the current state of the stream (one on which the classifier is being trained) and one for the incoming instances. This allows us to use IKS to compare the previous chunk of data with a new one and determine if they come from the same distribution or not. If there is at least a change in a single feature, then it is considered a presence of drift. After the detection, the system requires true labels for the instances from a new window and uses them to rebuild the classifier [238]. This is done regardless of how many features were denoted as drifting ones.

8.3 Unsupervised ensemble drift detection with feature subspaces

In this section, we will describe the motivations and details behind the proposed ensemble drift detection with the feature subspaces (**EDFS**) method.

Limitations of IKS. The concept of IKS lies in detecting a drift on every single feature independently. This is a strong oversimplification, as concept drift may be of complex nature and may affect a combination of multiple features. In such scenarios, it may be impossible to detect a drift as affecting a specific feature only and multivariate tests must be used to actually effectively detect a change. They are, however, computationally costly and require a lot of instances to work properly. This also makes IKS sensitive to false alarms and forces the classifier to be rebuilt too often. Furthermore, IKS assumes that the entire model must be rebuilt if drift is detected on at least one feature. This poses additional computational costs by treating local drifts as global ones. A more local approach that will inform only on what parts of the model should be rebuilt can be seen as a more attractive one.

Using feature subspaces for drift detection. We propose to combine IKS with creating feature subspaces. Instead of working on individual features, our proposed EDFS approach creates a set of feature subsets. In each of them, a voting scheme is applied among all univariate IKS runs. Drift is detected only if a majority of univariate IKS tests vote for change being detected. This increases the robustness to false alarms and noisy features, allowing a more reliable detection. We obtain an ensemble of drift detectors, each with local specialization. This can be seen as a hybrid approach that alleviates the drawbacks of univariate IKS, while not requiring the complex multivariate approach over the entire feature space.

Creating meaningful feature subspaces. Usage of feature subspaces is popular in machine learning, where ensembles are built on randomly drawn combinations of features. Let us note that this approach is not suitable for drift detection, as we aim at detecting local drifts that may affect multiple features at once. Random subspaces are not stable and do not guarantee significantly improved robustness to noise over using the univariate approach. Therefore, we need a method for constructing meaningful subspaces that will consist of features that have a high probability of being affected by drift at the same time.

The proposed EDFS method creates feature subspaces in a guided manner that utilizes two criteria: (i) quality of individual feature subspaces; and (ii) diversity among the created subspaces. As we want EDFS to be computationally feasible for high-speed data streams, we use several simplifications to the algorithm design. Firstly, subspaces are being created in a greedy manner with a round-robin strategy. This may lead to a non-optimal solution, yet offers a significant computational speed-up. Secondly, we make a strong assumption that a subspace consisting of individually strong features is itself of high quality. The proposed algorithm has two parameters: (i) the number of subspaces to be created k; and (ii) the number of features selected for every subspace n. The details are outlined in Algorithm 7.

Feature subspaces quality. As mentioned before, the estimation of subspace quality is based on the strength of the individual predictors. Using only the individually strong features not necessarily will improve the discriminative power of the subspace, or even more so of the whole ensemble. However, we claim that reducing the frequency of the occurrence of the weak predictors will, on average, result in increased performance.

Let us denote the y-th class label, encoded as an integer, as $y \in \mathcal{M} = \{1, 2, ..., M\}$. Furthermore, let $\mathcal{LS} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$ be the learning set consisting of n observations, $\boldsymbol{x}^{(f)} = [x_1^{(f)}, x_2^{(f)}, ..., x_n^{(f)}]$ be the vector of observations of feature $x^{(f)}$, and $\boldsymbol{y} = [y_1, y_2, ..., y_n]$ be the vector of class labels associated with observations.

Algorithm 7: EDFS subspace creation algorithm.

```
Data: number of subspaces $k$, number of features per subspace $n$, set of
      features $\mathcal{X} = \{x^{(1)}, x^{(2)}, ..., x^{(d)}\}$
Result: feature subspaces $S$
for $i \leftarrow 1$ to $k$ do
    $S_i \longleftarrow \emptyset$;
repeat
    for $i \leftarrow 1$ to $k$ do
        for $f \leftarrow 1$ to $d$ do
            if $x^{(f)} \notin S_i$ then
                $f_{score}(x^{(f)}) \longleftarrow qual\_m(x^{(f)}) + div\_m(S, S_i, x^{(f)})$;
        $x^{(best)} \longleftarrow \arg\max_{x^{(f)}} f_{score}(x^{(f)})$;
        $S_i \longleftarrow S_i \cup x^{(best)}$;
until every subspace consists of $n$ features;
return $S$
```

In order to measure the individual quality of features, we propose to use mutual information between the feature and the target $qual\_m_{mi}(x^{(f)})$:

$$qual\_m_{mi}(x^{(f)}) = \sum_{y=1}^{M} \sum_{j=1}^{n} p(x_j^{(f)}, y) \log \left(\frac{p(x_j^{(f)}, y)}{p(x_j^{(f)})\, p(y)} \right), \tag{8.9}$$

where we denote marginal probabilities of $\boldsymbol{x}^{(f)}$ and $\mathcal{M}$ as $p(x_j^{(f)})$ and $p(y)$, respectively, and their joint probability as $p(x_j^{(f)}, y)$.

Please note that this is a supervised measure, requiring access to class labels. While the proposed EDFS approach is an unsupervised drift detector, we follow the assumptions of IKS that after the drift is detected, we query for class labels for the recent window. Therefore, this allows us to recalculate mutual information every time drift is being detected.

Feature subspaces diversity. Let S denote the set of the existing subspaces and S_j stand for the j-th subspace. Let $\mathcal{X}$ be a set of the available features $\mathcal{X} = \{x^{(1)}, x^{(2)}, ..., x^{(d)}\}$. Consider inserting additional feature $x^{(f)}$ into the currently considered subspace S_j. We define a diversity metric $div\_m(S, S_j, x^{(f)})$ as an average of two components: the proportion of existing subspaces already containing the considered feature $div\_m_x(S, x^{(f)})$ and the distance to the most similar subspace $div\_m_s(S, S_j)$:

$$div\_m(S, S_j, x^{(f)}) = \frac{div\_m_x(S, x^{(f)}) + div\_m_s(S, S_j)}{2}, \tag{8.10}$$

where:

$$div\_m_x(S, x^{(f)}) = 1 - \frac{|\left\{S_j : x^{(f)} \in S_j\right\}|}{|S|}, \tag{8.11}$$

and:

$$div\_m_s(S, S_j) = 1 - \max_{j \neq l} \frac{|S_j \cap S_l|}{|S_j|}. \tag{8.12}$$

By minimizing the proposed metric we ensure that the features are spread evenly among the subspaces, which should contribute to the creation of a diverse set of learners. We make an underlying assumption that large groups of features are not highly correlated, in which case the proposed dissimilarity would be too simplistic. In practice, the situations that would lead to a complete failure of the proposed metric are very rare.

Drift detection with feature subspaces. The EDFS approach combines univariate IKS with meaningfully constructed feature subspaces. Therefore, a proper strategy when a change can be considered significant enough to raise an alarm is needed. We will discuss here how to analyze changes in subspaces, in the entire stream, as well as how EDFS reacts to the appearance of concept drift.

- **Drift detection in subspaces.** One of the major drawbacks of IKS lies in its univariate nature and inability to detect drift that spans a combination of multiple features. A simple application of IKS in each subspace would not differ from applying IKS over the entire feature space, as each feature would be treated independently. We propose that for each subspace, IKS is performed over all features assigned to this subspace. Then a majority voting over IKS univariate outputs is performed and its outcome decided whether this subspace is considered as affected by concept drift or not. This voting strategy offers increased robustness to noise or small fluctuations as compared with IKS, leading to a reduced risk of false alarms. These local decisions are then used to decide the final output of the EDFS detector.

- **Drift detection in the stream.** A good drift detector aims to strike a balance between sensitivity to changes and robustness to false alarms. Therefore, we need a proper strategy to decide if the entire data stream is indeed affected by a concept drift, based on local outputs of feature subspace-based decisions. While using majority voting on a local scale may benefit the robustness of EDFS, it may damper the sensitivity too much when it comes to making a final decision. The majority voting would force EDFS to detect drift only when over 50% of feature subspaces were considered as drifting ones. Therefore, on the global level, we propose to use the winner-take-all approach. This means that if at least one subspace is labeled as a drifting one, then EDFS detects drift over the entire stream.

- **Reacting to change.** After EDFS detects a drift, three steps are being taken: (i) class label query; (ii) classifier rebuilding; and (iii) subspace reconstruction. Class labels are requested for the entire new window that has been labeled as coming from a new distribution, as we need to adapt to the change. These new instances are used to retrain a classifier, in order to forget old concepts and make the classifier suitable for the current state of the stream. Additionally, previously established feature subspaces may no longer be meaningful and EDFS needs to build new feature subspaces by applying Algorithm 7 on newly acquired instances.

8.4 Experimental study

This experimental study was designed to empirically evaluate the effectiveness of EDFS for unsupervised drift detection, as well as to answer the three following research questions.

- **RQ1:** Is using drift detection over feature subspaces more effective than using a univariate IKS detector?
- **RQ2:** Does creating feature subspaces in a meaningful manner lead to a better drift detection over random subspaces?
- **RQ3:** Does EDFS offer an improvement over state-of-the-art unsupervised drift detectors?

8.4.1 Data stream benchmarks

For the purpose of evaluating our proposed algorithm, we generated 10 diverse and large-scale data stream benchmarks using the MOA environment [214]. By using data stream generators, we were able to fully control the nature and occurrence of concept drifts, which in turn led to a more explainable experimental study. By analyzing how the proposed method behaves in a controlled environment we may gain more in-depth insights into its strong and weak points. Details of the used data streams are given in Table 23.

8.4.2 Setup

Here, we will present the details of the experimental study design and its environment.

Table 23.: Properties of used data stream benchmarks.

Stream	Gen	Inst	Feat	Cls	Type	Drifts
HYP_{IF}	HYPER	1 mln	10	2	inc-fast	20
HYP_{IS}	HYPER	1 mln	10	2	inc-slow	10
LED_{M}	LED	1 mln	24	10	mixed	5
LED_{S}	LED	1 mln	24	10	sudden	50
RBF_{B}	RBF	1 mln	100	5	blips	100
RBF_{G}	RBF	1 mln	40	20	grad	20
RBF_{GR}	RBF	6 mln	20	10	grad-rec	5
SEA_{G}	SEA	3 mln	3	4	grad	10
SEA_{S}	SEA	3 mln	3	4	sudden	20
TRE_{S}	TREE	2 mln	10	6	sudden	50

Reference drift detectors. We compared the proposed **EDFS** with a **univariate IKS** [238] (which allows us to answer RQ1) and **RFS** – an identical architecture that utilizes random feature subspaces (which allows us to answer RQ2). Additionally, in the second part of the study, we utilized two state-of-the-art unsupervised ensemble drift detection methods – a committee of statistical detectors (**LDCNet**) [236] and an ensemble of heterogeneous concept drift detectors (**EHCDD**) [135].

Base classifiers. All experiments were conducted with the usage of Adaptive Hoeffding Tree [109] as an underlying learning and classification algorithm.

Parameters. EDFS and RFS used $k = 10$ subspaces and $n = 0.1$ for constructing its local drift detectors. These values were established during our experimental investigations as the most stable and effective ones. However, for each specific data stream, one may use a dedicated parameter selection method [231] to tune EDFS. We used windows of size 1000 for drift detection and batch processing. All reference methods used parameters suggested by their authors.

Statistical analysis. We analyzed the significance of obtained results using the Bayesian sign-rank test [234].

8.4.3 Experiment 1: Role of feature subspaces

In this experiment we wanted to answer **RQ1** and **RQ2**, to establish if ensemble drift detection in feature subspaces is beneficial to univariate drift detection, as well as to evaluate the effect of creating features subspaces in a guided manner. The results of the experiments according to the prequential accuracy are given in Tab. 24, while the outcomes of the Bayesian sign-rank test of statistical significance are visualized in Fig. 52.

Table 24.: Comparison of EDFS with randomly created subspaces and univariate detector according to the prequential accuracy [%].

Algorithm	HYP_{IF}	HYP_{IS}	LED_{M}	LED_{S}	RBF_{B}	RBF_{G}	RBF_{GR}	SEA_{G}	SEA_{S}	TRE_{S}
EDFS	**87.21**	**84.09**	**63.54**	**62.18**	**87.36**	**92.84**	**94.06**	83.79	81.02	**81.52**
RFS	82.42	82.23	60.12	60.02	80.74	90.11	89.38	81.25	78.01	77.42
IKS	84.95	82.87	56.43	58.95	85.43	92.02	92.14	**85.04**	**81.97**	75.20

Benefits of ensemble strategy. From the obtained results we can see that the ensemble strategy proposed by EDFS offers superior results to standard IKS, both in terms of performance over individual benchmarks, as well as in terms of statistical significance. While the observation that an ensemble detector is better than a single model does not seem novel and confirms recent trends [22], there is more to this comparison. One must note that IKS is de facto an ensemble strategy that combines univariate IKS detectors created over the entire feature space. Therefore, IKS is an implicit ensemble approach that assumes no correlation between the drift occurrence and the combination of features. EDFS takes IKS as a base model but combines it

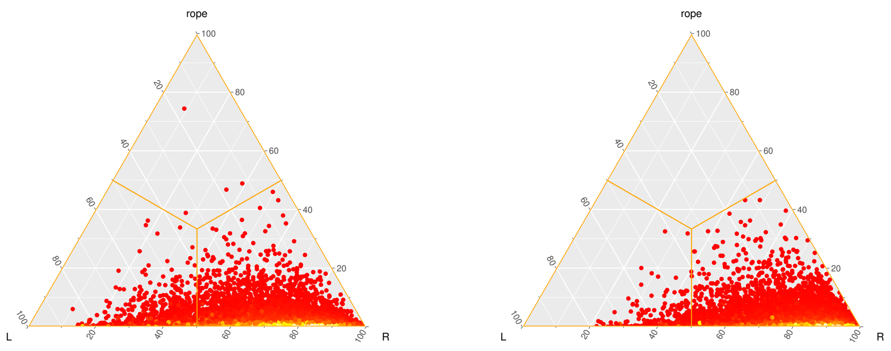


Fig. 52.: Posteriors for EDFS (R) vs. (left) RFS / (right) univariate IKS from the bayesian sign-rank test. Higher concentration of points on one of the sides of the triangle shows that a given method has a higher probability of being statistically significantly better.

with feature subspaces and two-level decision making (local and global). Obtained experiments prove that this is a much better strategy and allows us to more efficiently detect drifts, especially in scenarios where we deal with more complex types of drifts (e.g., RBF_{GR} or TRE_{S}).

Benefits of meaningful feature subspaces. EDFS assumes that drift detection in feature subspaces is more beneficial than univariate drift detection as long as the feature subspaces are of high quality and display some level of diversity. In order to test this hypothesis, we compared our approach with RFS using identical architecture and combination strategies but creating subspaces in a random manner. The obtained results confirm our assumptions, showing that in all of the tested cases EDFS returned statistically significantly better results. Furthermore, RFS in many cases is outperformed by univariate IKS. This shows that creating feature subspaces without any guidance may actually harm the drift detection procedure, leading to an increased number of incorrect alarms.

8.4.4 Experiment 2: Comparison with other detectors

In this experiment we wanted to answer **RQ3**, to establish if the proposed EDFS offers improvements over state-of-the-art unsupervised drift detectors. We selected two unsupervised ensemble strategies to offer a fair comparison. The results of experiments according to the prequential accuracy are given in Table 25, and the outcomes of the Bayesian sign-rank test of statistical significance are visualized in Figure 53.

Table 25.: Comparison of EDFS with randomly created subspaces and the univariate detector according to the prequential accuracy [%].

Algorithm	HYP_{IF}	HYP_{IS}	LED_{M}	LED_{S}	RBF_{B}	RBF_{G}	RBF_{GR}	SEA_{G}	SEA_{S}	TRE_{S}
EDFS	**87.21**	**84.09**	**63.54**	**62.18**	**87.36**	**92.84**	94.06	83.79	81.02	**81.52**
LDCNet	84.81	83.04	57.41	55.36	84.99	91.86	**95.27**	80.18	75.44	79.90
EHCDD	85.82	83.51	58.02	57.11	85.82	90.59	91.88	**84.92**	**82.91**	78.84

Benefits of EDFS. From the obtained results, we can see that EDFS performs favorably when compared to other unsupervised ensemble methods. LDCNet is an ensemble of statistical detectors that aims at anticipating the potential directions of drift. However, it is known to be very sensitive to noise and small changes in streams. EDFS does not have such problems, as using feature subspaces created in a guided manner allows for a more robust analysis of the magnitude of change. EHCDD is an effective and flexible framework that combines heterogeneous detectors with a weighted combination of their outputs. However, its main drawback lies in its sensitivity to weak ensemble members. A poorly tuned or over-sensitive detector will significantly impair the performance of EHCDD. EDFS uses a homogeneous pool of detectors (IKS) and a two-level combination strategy, leading to statistically better

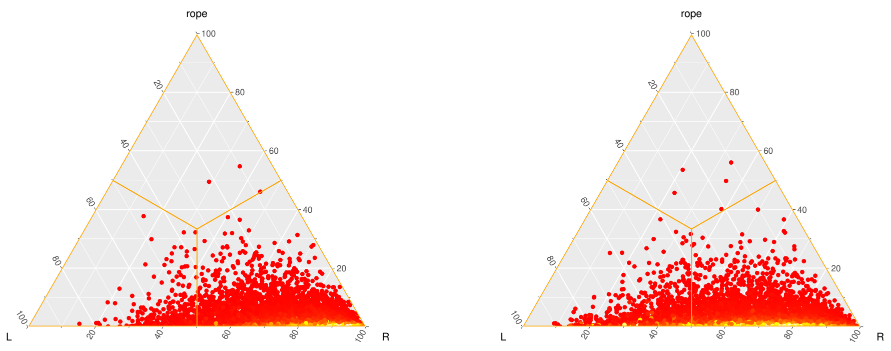


Fig. 53.: Posteriors for EDFS (R) vs. (left) LDCNet / (right) EHCDD from the bayesian sign-rank test. Higher concentration of points on one of the sides of the triangle shows that a given method has a higher probability of being statistically significantly better.

performance.

8.5 Summary

In this chapter, we proposed EDFS – an unsupervised ensemble drift detector for data stream mining that creates an ensemble of local drift detectors, each trained on a different feature subspace. We employed a guided strategy for constructing feature subspaces that utilized a combination of feature quality and diversity measures. This allowed us to alleviate the drawbacks of popular univariate drift detectors (e.g., discussed IKS) and offer a more robust change detection for complex drifts. EDFS reconstructs its subspaces after every drift, allowing for capturing new properties of the feature space. We used a two-level combination strategy, where univariate IKS detectors within each feature subspace were combined using majority voting, while drift was detected on a global scale using a winner-take-all strategy over outputs from all subspaces. A carefully crafted experimental study showed that such an ensemble architecture offers significant benefits as compared with the univariate approach, as well as displayed the importance of creating feature subspaces in a guided manner. Finally, we showed that EDFS compares favorably with state-of-the-art unsupervised ensemble drift detectors.

Our future works will focus on evaluating different ways of constructing subspaces, as well as applications to semi-supervised learning.

CHAPTER 9

DYNAMIC ENSEMBLE DIVERSITY AND ADAPTATION TO CONCEPT DRIFT

Ensemble learning for data stream mining has been gaining increasing attention [22], taking advantages of the mentioned solutions and using multiple classifiers at the same time [239]. Ensembles have been shown not only to improve predictive accuracy, but also to efficiently handle concept drift by replacing outdated classifiers in the pool [240]. One of the key aspects of ensemble learning is diversity among base classifiers [241]. In general, it is desirable to create mutually different, yet complementary, base learners to expand the overall competency of the whole committee. One group of methods used to diversify ensembles are region-based algorithms that train dedicated classifiers for different subspaces of data. For static datasets, Lee and Kim [242] proposed the eSVM ensemble, dividing the dataset into heterogeneous parts used by base learners. It is also possible to generate clusters for each class or even split it internally to create an ensemble of one-class models [243].

The idea of adapting diversity to streaming data environments is not a recent one, however, the coverage of it is still surprisingly limited. The first and most interesting work considering diversity and evolving data streams was presented by Minku [120]. It posed the important question – how diversity may influence learning from drifting streams. The main conclusion of the work is that high diversity improves recovery from concept drifts, but on the other hand, it usually impedes the learning process during stable periods. Additionally, the work points out that the more severe is concept drift, the higher diversity is needed. To control the level of diversity, authors dynamically manipulated the Poisson distribution used by the online bagging algorithm [223]. Lower values of λ provided higher diversity, while higher values of the parameter were suitable for stable periods. Minku and Yao [244] developed the presented idea and created a more sophisticated algorithm (DDD), focusing on maintaining a high-diversity ensemble for concept drifts and a low-diversity one for stable substreams.

While ensemble diversification seems like a valuable direction in data stream mining, it must be done in a controlled fashion. It is remarkable that although there are some works presenting ensembles promoting diversification, almost none of them investigate the diversity measures explicitly [245]. Brzezinski [246] presented a work that emphasized the lack of diversity analysis in the data streams domain. He investigated six well-known diversity measures, using different ensembles and drifting data streams, showing that they can be easily used in such settings and analyzing the impact of changes on the measures. It has to be mentioned here, however, that substantial doubts about the usefulness of different diversity metrics were raised in some publications [241].

In this chapter, we propose a new ensemble learning scheme that allows for effective, empirically proven dynamical diversification of base classifiers. It uses an online clustering approach to create locally specialized classifiers trained on chunks of spatially related instances. Diversity management is controlled by a drift detector that allows to dynamically change the diversification level according to the current state of the stream. An extensive experimental study on synthetic and real data stream benchmarks shows the efficacy of the proposed ensemble and offers an insight into the role of diversity in data streams.

9.1 Proposed algorithm

We present the first attempt to build a clustering-driven ensemble that promotes dynamic diversity for drifting data streams. Based on the published observations concerning diversity and concept drifts, we dynamically adjust it accordingly to the current state of a stream. Algorithm 8 presents the general framework implementing the clustering-based idea, while in the next paragraphs we discuss each of its components in detail.

Algorithm 8: Online clustering-driven ensemble.

```
Data: ensemble size $e$, ClusteringStrategy ($parameters$),
      DriftIndicator ($parameters$), ControlStrategy ($parameters$)
Result: ensemble $\boldsymbol{L}_i$ at every iteration $i$
Initialization: $i \leftarrow 1$, $\boldsymbol{L}_0 \leftarrow []$, $\boldsymbol{C}_0 \leftarrow []$
repeat
    receive incoming instance $\boldsymbol{x}_i$;
    request the true label $y_i$ of instance $\boldsymbol{x}_i$;
    update clusters ClusteringStrategy($\boldsymbol{C}_{i-1}, \boldsymbol{x}_i, e$);
    if $|\boldsymbol{C}_i| > |\boldsymbol{C}_{i-1}|$ then
        add new classifier $l$ to $\boldsymbol{L}_{i-1}$;
    $d \leftarrow$ DriftIndicator($\boldsymbol{x}_i, y_i$);
    $r \leftarrow$ ControlStrategy($d$);
    $\boldsymbol{I}_c \leftarrow$ indices of $r|\boldsymbol{C}_i|$ centroids closest to $\boldsymbol{x}_i$;
    for $j \leftarrow 1$ to $len(\boldsymbol{I}_c)$ do
        update classifier $\boldsymbol{L}_i[\boldsymbol{I}_c[j]]$ with $(\boldsymbol{x}_i, y_i)$;
    $i \leftarrow i + 1$;
until stream ends;
```

Ensemble. We use instances $\boldsymbol{x}_i$ to update the core modules of our algorithm – a set of clusters $\boldsymbol{C}_i$ and an ensemble $\boldsymbol{L}_i$ (we maintain one of each for a whole stream). To create a diverse ensemble we utilize one classifier per cluster. Whenever a new cluster is formed, so the number of clusters increases $|\boldsymbol{C}_i| > |\boldsymbol{C}_{i-1}|$, a new base learner is added to the ensemble. Each classifier learns only from data belonging to its cluster. This space partitioning creates classifiers specialized in recognizing different parts of data, so in general, they should be diverse, yet mutually complementary. We combine decisions of the base learners using accuracy-weighted majority voting. One must also remember that the base learners have to be able to handle concept drifts since the framework improves adaptivity, but it does not provide it itself.

Clustering strategy. The clusters are actualized in an online manner. The incoming instance $\boldsymbol{x}_i$ is assigned to a cluster based on `ClusteringStrategy` used. Currently, there are few such algorithms available, even fewer try to tackle data streams problems, like evolving concepts or anomaly detection. In our framework we use an online k-means algorithm [247], which has been shown as a competitor to the offline k-means++ algorithm. However, it tends to find slightly more clusters than a given parameter e. The method generates clusters in an online manner, but it does not use any direct mechanism to handle changes in data, so the approximation of the optimal solution may be impeded. It is not a crucial aspect in our case, since we focus mainly on the changes in the conditional probability, rather than on data distribution, therefore, we do not need very precise clusters. Nevertheless, the algorithm will modify its centroids incrementally starting from near-optimal seeds. To boost this process we use a windowing mechanism by applying the moving average to the centroid's features. Our experiments show that such an approach is sufficient in practice.

Drift indicator. After updating the clusters, we collect the measure of concept drift d calculated by `DriftIndicator`. There are several online change detectors already published, and many of them are based on statistical tests and error measures. They are usually dedicated to different types of concept drift. In our framework, we use the windowed error ϵ_ω of size ω as a drift indicator, calculated using the moving average method. It is assumed that the error increases when concept changes since classifiers need time to adapt to the drifting data. One must remember that the indicator is very sensitive to the window size and access to labeled instances.

Control strategy. The drift indicator d is used to control the value of the range variable $r \in \langle 0, 1 \rangle$, by applying `ControlStrategy`. The strategy defines a relation (trade-off) between d and r. The range variable determines how many classifiers use a newly arriving instance. First, Euclidean distances between the object and all centroids are calculated. Then, indices of the $r|\boldsymbol{C}_i|$ closest centroids are selected and classifiers paired with them are updated. The varying range of classifiers controls the level of diversity among base learners. The idea is as follows – when concept drift appears the error increases and r decreases. As a result, instances are used only by the closest classifiers and they start differentiating from each other. When a concept becomes stable, the error decreases and r increases. This leads to the situation in which base learners more frequently learn from the same instances, therefore, diversity should decrease in favor of variance reduction.

Work presented in [120] indicates that dosing different amounts of diversity is beneficial, however, it does not determine how high or low the diversity should be. The linear relation is probably the most straight-forward one. We define three different control strategies using a sigmoid function $\sigma(x) = x(\beta - 1)/(2\beta x - \beta - 1)$, where $x = 1 - \epsilon$ and $\beta \in \langle 0, 1 \rangle$ controls the shape of the curve representing relation between error (ϵ) and range (diversity).

- LINEAR ($\beta = 0$) – it maintains the linear relation between error and diversity, so any change in the former will result in directly proportional change in the latter.
- STABLE ($\beta < 0$) – diversity increases slower than error, so this approach promotes low-diversity ensembles.
- DIVERSE ($\beta > 0$) – it is opposite to the previous strategy, so it promotes

higher-diversity committees.

Intensity. The algorithm presented above is a basic version. Since we want our framework to be able to adapt to the varying speed of changes, we may need to somehow intensify the diversification process. One should be aware of the fact that when the drift appears abruptly and we have access to but few instances from the new concept, the diversification method, based on a single instance usage, may be insufficient for changing diversity effectively. We propose a simple improvement that aims to boost the process by using a single instance several times. We consider two approaches.

- FX – it uses a fixed number of duplicates regardless a state of a stream. The idea is to equally intensify both increasing and decreasing diversification.
- ER – the strategy increases the number of duplications if an error is higher, so it promotes intensification when concept drift occurs.

9.2 Experimental study

In our evaluation section, we aim to investigate three major properties of our method. The first one is **general performance** during drifts (in tables: Drifts) and stable periods (Stable). We report the average values of the prequentially calculated accuracy and Cohen's kappa. For stable substreams, we also present standard deviations (StdS) that indicate whether performance during them is indeed stable. Secondly, we investigate the ability of our method to generate **diversity** during concept drifts and to reduce it when stable periods appear. We explicitly measure 2 different diversity measures: disagreement (D) and double fault (DF) to show that our method of diversification effectively manages it as intended. We present time series of the metrics along with classification performance, which is, in fact, our drift indicator. Finally, we evaluate our framework **on real data streams** to find if it is able to compete with other well-known ensembles. Since currently we cannot define concept drifts for most of the real streams, we investigate the algorithms in general, using the prequential accuracy and kappa. Even if the streams are not as predictable as the synthetic ones, they are surely more reliable, since we still do not know how authentic data generators are.

9.2.1 Data

Using MOA [214], we generated 14 artificial streams based on 5 different synthetic concepts and consisting of different types of concept drift. They are summarized in Tab. 26.

Table 26.: Summary of the used synthetic data streams.

Name	Inst	Attr	Cls	First	Dist	Width	Drifts	Noise
SEA1	600k	3	2	150k	300k	100	3	0.05
SEA2	600k	3	2	150k	300k	10k	3	0.05
STAG1	600k	3	2	150k	300k	100	3	-
STAG2	600k	3	2	150k	300k	10k	3	-
RBF1	1m	15	5	250k	250k	100	3	0.05
RBF2	1m	15	5	250k	250k	10k	3	0.05
RBF3	1.2m	15	5	400k	400k	50k	2	0.15
RBF4	1.2m	15	5	400k	400k	100k	2	0.15
TREE1	1m	15	5	250k	250k	100	3	0.05
TREE2	1m	15	5	250k	250k	10k	3	0.05
TREE3	1.2m	15	5	400k	400k	50k	2	0.15
TREE4	1.2m	15	5	400k	400k	100k	2	0.15
HYPER1	500k	15	5	-	-	-	-	0.01
HYPER2	500k	15	5	-	-	-	-	0.01

Table 27.: Summary of the used real data streams.

Name	Inst	Attr	Cls	Name	Inst	Attr	Cls
Activity	10 853	43	8	Hepatitis	1 000 000	20	2
ActivityRaw	1 048 570	3	6	Lymph	1 000 000	19	4
Airlines	539 383	7	2	Poker	829 201	10	10
Connect4	67 557	42	3	Sensor	2 219 804	5	54
Covertype	581 012	54	7	Spam	9 324	499	2
DJ30	138 166	8	30	Weather	18 159	8	2
Electricity	45 312	8	2	Wine	1 000 000	658	2
Gas	13 910	128	6				

The HYPER1 stream was generated using a magnitude of change $t = 0.001$ and HYPER2 using $t = 0.01$. For the final evaluation, we used 15 real streams in total. Most of them (12) are pure real streams, while 3 (Hepatitis, Lymph and Wine) were generated on the basis of real datasets coming from UCI. They are presented in Tab. 27.

9.2.2 Algorithms

We evaluate our framework using different strategies for controlling diversity – linear (**CL**), diversity-oriented (**CLD**) or stability-oriented (**CLS**), as well as, without dynamic diversification (static **SCL**). We check how intensification methods using a fixed (**CL+FX**) and error-driven (**CL+ER**) number of duplications influence performance. Finally, we combine the strategies and evaluate them on synthetic streams. Due to limited space, on real streams we evaluate only two configurations (**CLD+ER** and **CLS+ER**).

For the online k-means strategy we selected $e = 10$ as a number of target clusters. It determines the desired size of an ensemble, however, it may usually vary between 10 and 15 base learners [247]. Since we want to adapt to the various speed of changes, including sudden ones, for windowed clusters and error indicators we use size $\omega = 100$ to make it reactive. For the `FX` intensification strategy we set $n = 20$ as a number of duplications per instance. As base learners we selected Adaptive Hoeffding Trees (AHT).

During the detailed analysis of our framework on synthetic streams we compare it with another algorithm that promotes dynamic diversity – online bagging based on [120]. We control λ using the same control strategies as in our solution (**BAG**, **BAG-D** and **BAG-S**). On real streams we compare our two most promising configurations with the bagging algorithms and 6 well-known and widely cited online ensembles: Learn++.NSE (**LNSE**) [156], Dynamic Weighted Majority (**DWM**) [121], Accuracy Updated Ensemble (**AUE**) [123], Adaptable Diversity-based Online Boosting (**ADOB**) [248], Online Smooth Boosting (**OSB**) [249] and OzaBag-ASHT (**OB-AHT**) [223], which is a basic online bagging algorithm using AHT and maintaining static diversity among them. For all ensembles we selected $e = 10$ for their size. The rest of parameters are set to default values and base learners are AHT.

9.2.3 Evaluation

All presented series results (accuracy and kappa) were collected using the prequential evaluation and window whose size was $\omega = 1000$. We distinguish results for concept drifts and stable periods by calculating averages separately. For each change j that has its peak at p_j and width w_j we treat results within $i \in \langle p_j - w_j/2, p_j + w'_j \rangle$ as measurements for drifts. We set $w'_j = w_j/2$ for $w \geq 20000$ and $w' = 10000$ for $w < 20000$, since we treat concept drift as something relative to the model performance and drops in effectiveness can be longer than an actual period of concept change, especially a short one. To make the evaluation setting more realistic, we simulated limited access to labeled data. We provided a warm start on each data stream using 10% of it, as a fully labeled initialization batch, and then only 10% of randomly selected instances from the remaining part.

9.2.4 Results

Obtained results clearly show that our diversification heuristic is effective and that the whole framework, if only properly configured, can effectively adapt to changes. Different strategies and combinations evince different behaviors, but only some of them can be established as the proper ones. In general, differences in results for kappa are greater than for accuracy. Below we discuss all the results, showing why certain approaches are better than the others.

Static vs. dynamic diversity. First of all, we can see in Fig. 54 that when static diversification is used (SCL), the diversity level changes not adequately to the

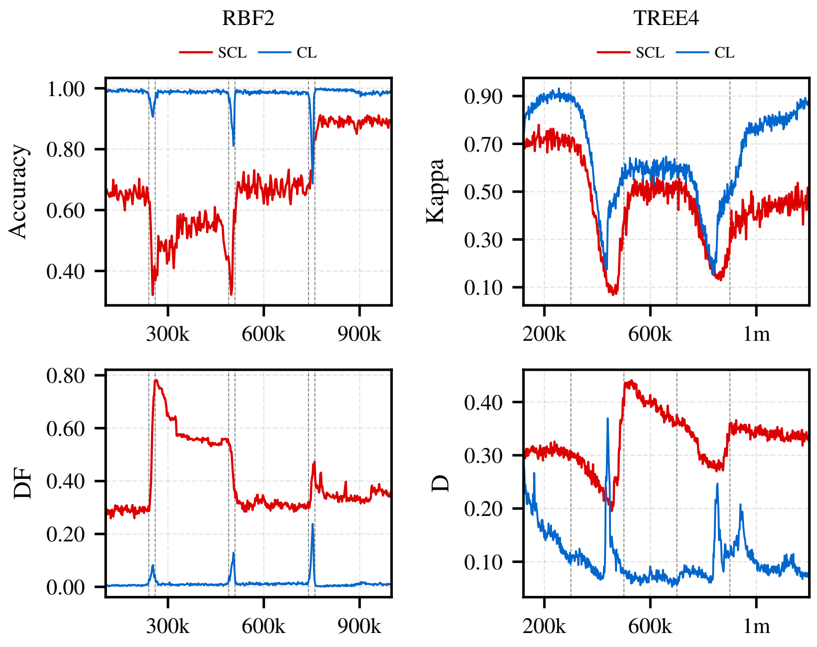


Fig. 54.: Performance and diversity series for RBF2 (left) and TREE4 (right), using different strategies.

error (concept drift, between dotted lines) and the adaptation of the framework may be very unstable. At the same time, once we add the linear control mechanism (CL), the reaction to changes is significantly better adjusted. The dynamic range effectively increases diversity just after a drift occurs and, as a result, leads to a faster reduction of an error. One may look a bit closer to see that the error stops increasing once diversity is high enough and that recovery begins subsequently. At the same time, when the error is getting low, the diversity level is being reduced, leading to stabilization. The same observation is valid for all results we describe in the next paragraphs.

In the general summary (Tab. 28), one can see that CL outperforms the static SCL during drifts and stable periods. These observations allow us to conclude that our method is an effective mechanism for generating dynamic diversification. We

Table 28.: Summary of synthetic streams results for accuracy and kappa.

	Accuracy				**Kappa**			
Algorithm	All	Drifts	Stable	StdS	All	Drifts	Stable	StdS
SCL	0.6753	0.5331	0.6519	0.0808	0.4833	0.2459	0.3477	0.1251
CL	0.8838	0.7221	0.9197	0.0420	0.8143	0.5615	0.8594	0.0734
CLD	0.8453	0.6569	0.8726	0.1010	0.7599	0.4625	0.7941	0.1580
CLS	0.8866	0.7564	0.9228	0.0297	0.8224	0.6052	0.8694	0.0573
CL+FX	0.8227	0.7856	0.8539	**0.0246**	0.7168	0.6268	0.7360	**0.0492**
CL+ER	**0.9028**	**0.8258**	**0.9411**	0.0272	0.8434	**0.7042**	0.8957	0.0526
CLD+FX	0.8241	0.7628	0.8550	0.0267	0.7164	0.5907	0.7353	0.0557
CLD+ER	0.8925	0.7855	0.9297	0.0352	0.8288	0.6365	0.8764	0.0674
CLS+FX	0.8042	0.7622	0.8369	0.0314	0.6848	0.5891	0.7069	0.0573
CLS+ER	0.8981	0.8247	0.9379	0.0283	**0.8444**	0.7026	**0.8983**	**0.0492**
BAG	0.8410	0.6989	0.9097	0.0481	0.7508	0.5291	0.8473	0.0796
BAG-D	0.7645	0.6468	0.8055	0.1062	0.6223	0.4354	0.6876	0.1606
BAG-S	0.8507	0.7499	0.9199	0.0298	0.7680	0.6164	0.8649	0.0548

made the comparison since for some ensembles it is possible that they increase their diversity itself when drift occurs [246]. It does not happen in our case.

Change-diversity trade-off. The results for different relations between the drift indicator and diversity show that, in general, stability-oriented or at least linear approaches improve adaptation, while strategies focusing on increased diversity tend to impede it (Tab. 28). For both accuracy/kappa when our framework and the bagging algorithm were promoting higher diversity (CLD and BAG-D) during the whole learning process, they worked worse than the configurations using the linear (CL and BAG) or stability-oriented control (CLS, BAG-S). The latter achieved better results not only for stable periods (about 0.92/0.87 for CLS), than the former (0.87/0.79 for CLD), but also during drifts (about 0.75/0.61 for CLS, 0.66/0.46 for CLD). Differences for the bagging approaches were even bigger than for our framework. We

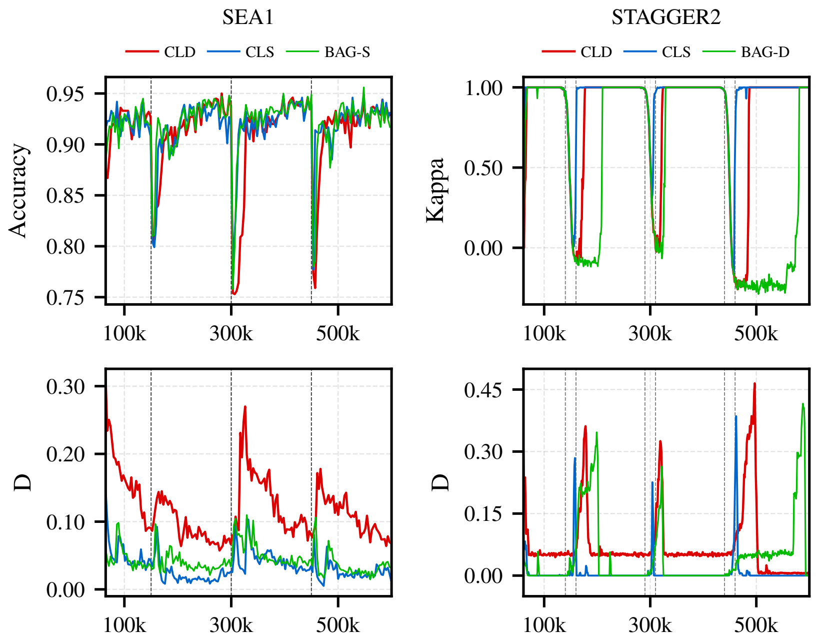


Fig. 55.: Performance and diversity series for SEA1 (left) and STAGGER2 (right), using different strategies.

can also notice that the high-diversity strategies were much more unstable during pure concepts (about 0.10 for accuracy standard deviation and 0.16 for kappa) than stability-based ones (about 0.3 and 0.6 respectively). The latter were also slightly more stable than the linear approaches.

We can see in Fig. 55 and 56 that all high-diversity ensembles were, indeed, maintaining higher values of disagreement. It could be beneficial during drifts, however, CLD and BAG-D algorithms struggled with reducing the high level of diversification after drifts. It usually resulted in prolonged adaptation, even if drops in performance were sometimes on the same level (SEA1, STAGGER2). The linear and stability-based methods maintained lower diversity even during drifts, but at the same time, they were stabilizing learning much more quickly. CLS was able to trigger very precise diversification just around drift peaks on STAGGER2 and TREE3, as well as, to significantly reduce the maximal loss on TREE3 and RBF4.

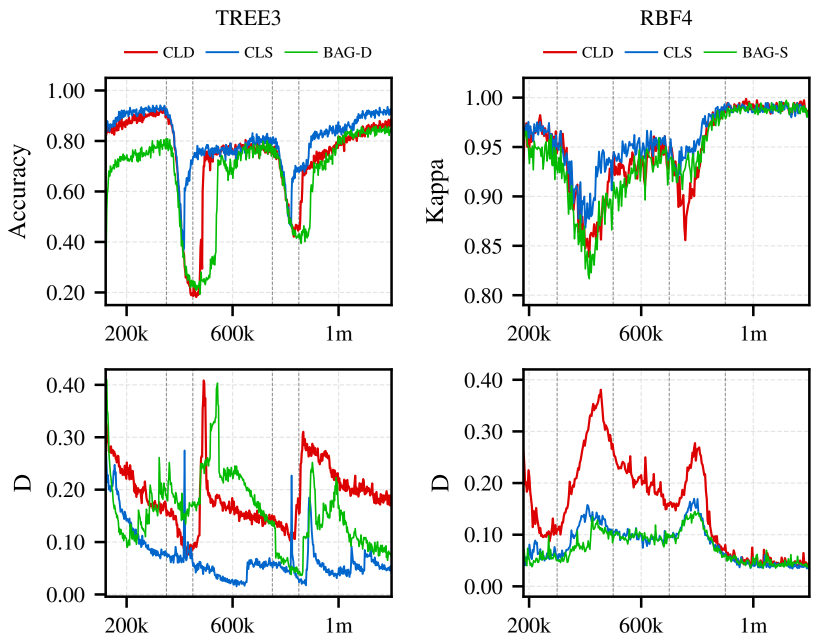


Fig. 56.: Performance and diversity series for TREE3 (left) and RBF4 (right), using different strategies.

Applying intensification. While considering intensification improvements, there is a clear indication that the strategy using the error-driven heuristic (CL+ER) is more reasonable in practice than the approach which tries to intensify both diversification and stabilization (CL+FX). We can see in Tab. 28 that although CL+FX slightly improves recovery from drifts (0.79/0.62) over basic CL (0.72/0.56) it impedes the performance of our framework during stable periods (0.85/0.74 and 0.92/0.86, respectively). Since the latter are longer than the former, the overall performance is also worse (0.82/0.72 for CL+FX and 0.88/0.81 for CL). In Fig. 57 and 58 we can see that CL+FX disturbed the diversity control, making it almost unresponsive to drifts. On the other hand, CL+ER was maintaining relatively low diversity during stable periods, similarly to basic CL, and at the same time, it was able to intensify recovery from drifts by increasing diversity even more precisely. One can notice that

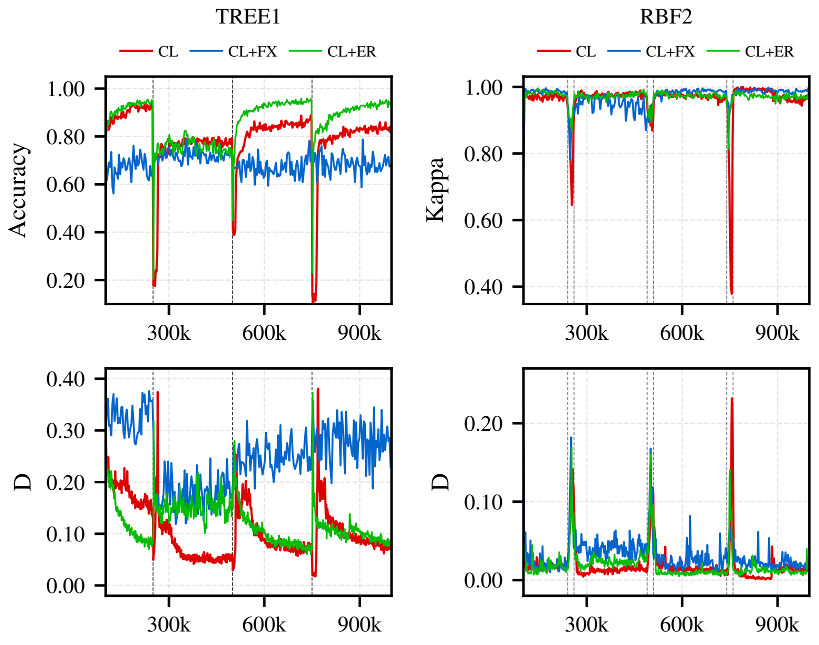


Fig. 57.: Performance and diversity series for TREE1 (left) and RBF2 (right), using different strategies.

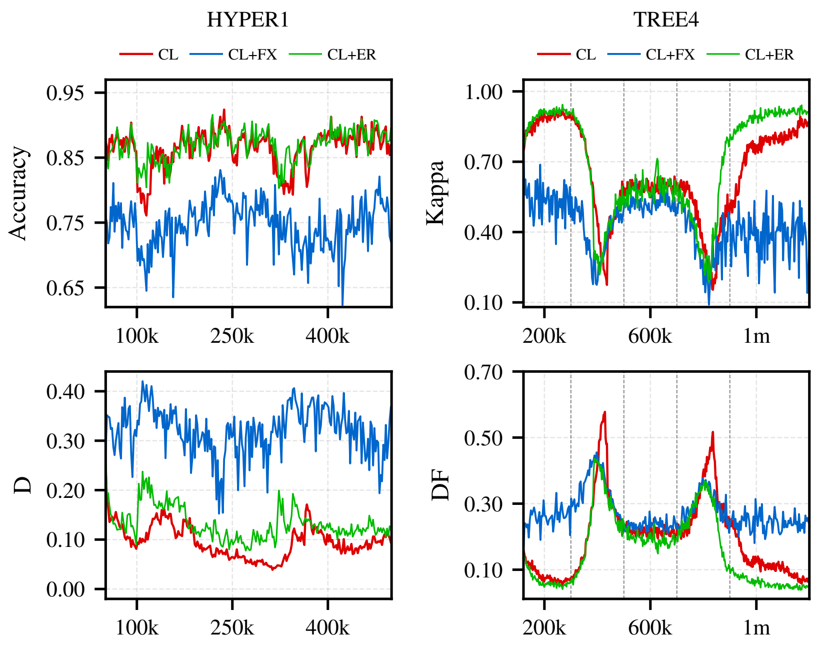


Fig. 58.: Performance and diversity series: for HYPER1 (left) and TREE4 (right), using different strategies.

diversification peaks were concurring with the lowest performance drops on TREE1, RBF2 and TREE4.

CL+ER provides significant improvements over basic CL mainly during drifts (0.83/0.70). Improvements are less significant for stable periods. The FX strategy obtained the lowest standard deviation since better approaches were constantly improving their performance, leading to bigger differences between the lowest and highest values, even during stable periods.

Combinations. We can see that results for combinations of the two mentioned mechanisms reflect the relations between these methods and that the intensification strategies are the main factor behind obtained improvements. Firstly, for both CLD and CLS heuristics differences between FX and ER were significant (about 0.07-0.09/0.11-0.16) as for CL+FX and CL+ER. Secondly, differences between FX and ER strategies combined with low or high diversity approaches were minor, similar to CLS

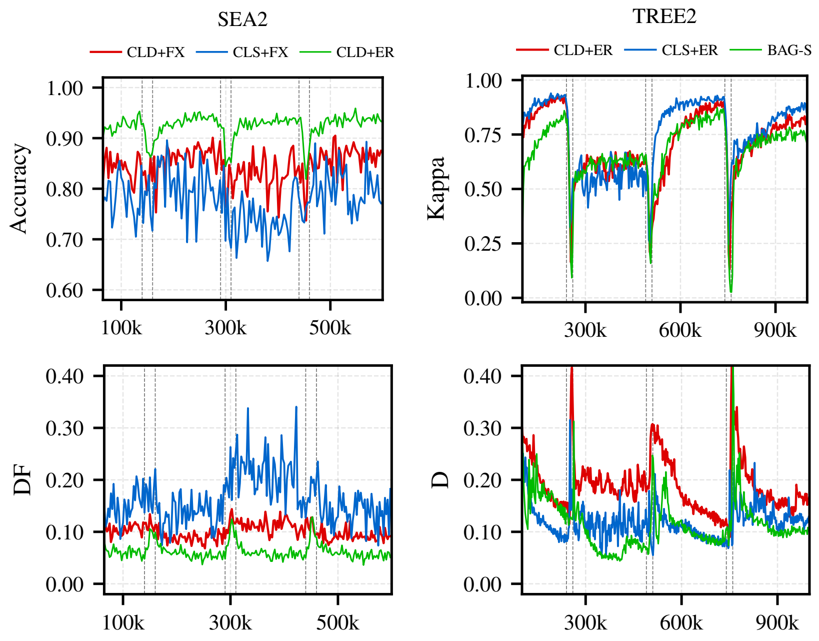


Fig. 59.: Performance and diversity series for SEA2 (left) and TREE2 (right), using different strategies.

and CLD. Thirdly, one can notice that adding ER slightly reduced the gap between CLD and CLS, as well as, that FX even reversed the relation between the control strategies. Finally, out of the two best combinations, CLS+ER was better than CLD+ER regarding performance during drifts (0.82/0.70 and 0.79/0.64, respectively in Tab. 28). Nonetheless, they were still on a similar level as simpler CL+ER. In Fig. 59 we can see that the FX heuristic was disturbing the adaptation process regardless of a control strategy used (SEA2) and that, depending on the strategy, ER worked better or worse than BAG-S (TREE2).

Real data streams. In Tab. 29 we can see results for the final evaluation conducted on real data streams. It presents average accuracies and kappa values for two different configurations of our framework, all bagging heuristics and six popular ensembles. We can see that while for bagging the dynamic λ approach was insufficient to make it competitive with most of the ensembles, our best clustering-driven configuration

Table 29.: Results on real data streams with the respect to accuracy and kappa.

Stream	**CLD+ER**	**CLS+ER**	BAG	BAG-D	BAG-S	OB-AH	LNSE	DWM	AUE	ADOB	OSB
Activity	**0.7150**	0.6961	0.6224	0.5821	0.6180	0.6819	0.4890	0.6877	0.5200	0.6494	0.6697
ActivityRaw	0.5837	0.8612	0.4558	0.4057	0.5683	0.6928	0.5139	**0.8756**	0.4993	0.4053	0.6270
Airlines	0.6033	0.6003	0.6220	0.6150	0.6182	**0.6536**	0.6181	0.6560	0.6323	0.4726	0.6365
Connect4	0.7038	0.7087	0.6818	0.6581	0.6876	0.7023	0.6300	**0.7150**	0.6603	0.6080	0.7127
Covertype	0.8046	**0.8119**	0.6713	0.6508	0.6860	0.7396	0.7564	0.7960	0.7440	0.3704	0.7359
DJ30	0.9475	0.9518	0.7956	0.6521	0.8451	0.9814	0.8230	0.9813	**0.9912**	0.9819	0.9858
Electricity	0.7997	**0.8101**	0.7231	0.7109	0.7368	0.7786	0.7185	0.7858	0.7308	0.7058	0.7724
Gas	0.6495	0.6652	0.5551	0.5459	0.5503	0.5526	0.4398	**0.7162**	0.4861	0.3848	0.5489
Hepatitis	0.9083	0.9093	0.8397	0.8347	0.8410	0.9026	0.8695	0.8886	0.9156	0.9111	**0.9178**
Lymph	0.8893	0.8726	0.7944	0.7868	0.7998	0.8947	0.8448	0.8623	0.8898	0.8806	**0.8978**
Poker	**0.7315**	0.7039	0.5855	0.5585	0.5982	0.7203	0.5465	0.6640	0.6047	0.5543	0.7155
Sensor	0.4518	0.6192	0.0616	0.0538	0.2611	0.4741	0.3344	**0.6796**	0.4421	0.0303	0.3678
Spam	**0.9171**	0.9152	0.8857	0.8544	0.8867	0.7508	0.6712	0.8044	0.4853	0.8607	0.7589
Weather	0.7329	**0.7351**	0.6879	0.6760	0.6819	0.7018	0.6791	0.6912	0.7234	0.7236	0.7063
Wine	0.9270	0.9239	0.8707	0.8686	0.8709	0.9323	0.9131	0.9208	0.9320	0.9283	**0.9336**
Average	0.7577	**0.7856**	0.6568	0.6302	0.6833	0.7440	0.6565	0.7816	0.6838	0.6311	0.7324

Stream	**CLD+ER**	**CLS+ER**	BAG	BAG-D	BAG-S	OB-AH	LNSE	DWM	AUE	ADOB	OSB
Activity	**0.6264**	0.6048	0.4978	0.3991	0.5051	0.5933	0.3069	0.5971	0.3086	0.5592	0.5819
ActivityRaw	0.4723	0.8107	0.1699	0.0896	0.3561	0.5504	0.2839	**0.8280**	0.2731	0.0329	0.4494
Airlines	0.1914	0.1838	0.2114	0.1708	0.2079	0.2684	0.2132	**0.2836**	0.2371	-0.0823	0.2327
Connect4	0.3110	**0.3601**	0.1869	0.0004	0.2367	0.2661	0.1511	0.3293	0.1969	0.2202	0.3148
Covertype	0.6829	**0.6981**	0.4708	0.4362	0.4967	0.5784	0.6082	0.6739	0.5846	0.0126	0.5676
DJ30	0.9456	0.9502	0.7884	0.6399	0.8397	0.9807	0.8168	0.9807	**0.9909**	0.9813	0.9853
Electricity	0.5848	**0.6075**	0.4333	0.4024	0.4597	0.5315	0.4103	0.5513	0.4433	0.3547	0.5174
Gas	0.5716	0.5973	0.4661	0.4509	0.4614	0.4690	0.3257	**0.6595**	0.3855	0.2504	0.4656
Hepatitis	0.7061	0.7111	0.5567	0.5411	0.5602	0.6968	0.6159	0.6594	**0.7310**	0.7057	0.7398
Lymph	0.7934	0.7632	0.6462	0.6336	0.6551	0.8032	0.7132	0.7454	0.7939	0.7762	**0.8092**
Poker	**0.5130**	0.4640	0.2359	0.1771	0.2574	0.4807	0.2035	0.3908	0.2808	0.1188	0.4785
Sensor	0.4402	0.6110	0.0430	0.0349	0.2461	0.4631	0.3203	**0.6728**	0.4304	0.0111	0.3546
Spam	**0.7860**	0.7701	0.6932	0.6397	0.7011	0.4887	0.3516	0.5117	0.1688	0.6378	0.4936
Weather	0.3319	**0.3512**	0.3023	0.2123	0.0752	0.1428	0.2118	0.1154	0.2455	0.2605	0.1743
Wine	0.8893	0.8844	0.8043	0.8011	0.8045	0.8972	0.8681	0.8799	0.8968	0.8911	**0.8992**
Average	0.5897	**0.6245**	0.4337	0.3753	0.4575	0.5474	0.4267	0.5919	0.4645	0.3820	0.5376

(CLS+ER, 0.79/0.62) was able to outperform 8 out of 9 other algorithms on average and to be at least competitive to the second best ensemble – DWM (0.78/0.59).

In fact, CLD+ER and CLS+ER were on a very similar level of performance on almost all data streams, however, the latter one turned out to be much more efficient on two very long and constantly unstable steams – ActivityRaw (0.58/0.47 for CLD+ER and 0.86/0.81 for CLS+ER) and Sensor (0.45/0.44 and 0.62/0.61, respectively). It is possible that these streams are characterized by several short drifts or that the ensembles cannot saturate the concepts sufficiently, so they suffer from constant fluctuations and as a result, CLD+ER had multiple problems while reducing diversity.

The CLS+ER ensemble was the most often the best or second best solution (0.53 of cases for accuracy and 0.60 for kappa, Fig. 60), outperforming all other ensembles for both metrics on Covertype, Electricity and Weather (bold green), and being better than not our ensembles on Activity and Spam (green). Only for Poker the CLD+ER ensemble was able to be more effective than CLS+ER and, at the same time, than the other algorithms. Regarding accuracy, DWM was slightly more often the best algorithm (0.33), however, it never was the second best (usually being worse than both our algorithms in such cases) and more often 6th or 7th (0.33) than CLS+ER

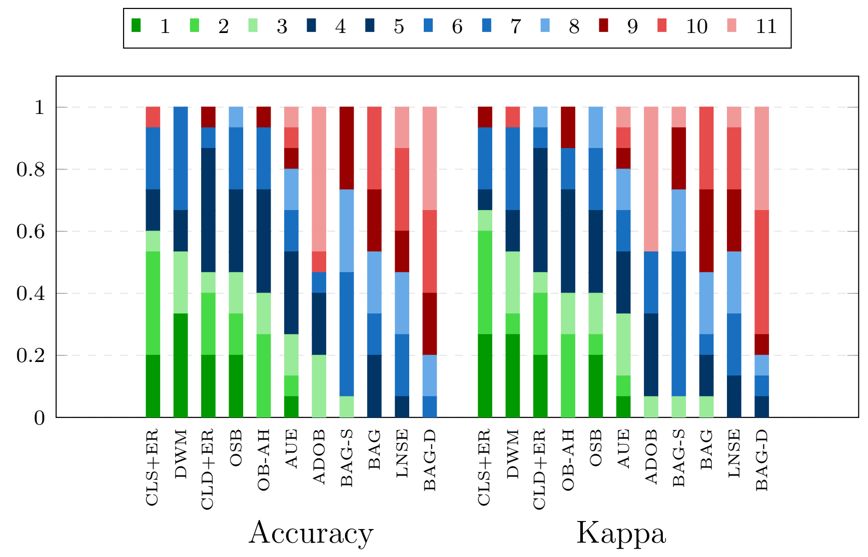


Fig. 60.: Ranks of ensembles for all real data streams as a fraction of all cases (15).

(0.2). For kappa both ensembles were equally frequently the best classifiers (0.27) and the rest relations remained very similar. Between the best and the worst ensembles, OB-AH and OSB were competitive with CLD+ER, however, still significantly worse than CLS+ER, especially for kappa values (the gap was about 0.08). It is also worth noting that even if CLS+ER was sometimes on further ranks (3-7), it was usually on a similar level as the best ones (for example Connect4 or Poker) or on a lower, but still not dramatically worse one (for example Gas or Sensor).

We can see that dynamic diversification for the online bagging performed much worse than for the clustering-driven solutions. In general, BAG-S tuned out to be the best option (0.68/0.46) and BAG-D (0.63/0.38) to be the worst one. Therefore, considering both our framework and the bagging algorithms, the relations between different configurations were very similar to those observed on synthetic streams. However, the dynamically diversified bagging algorithms were worse than most of the ensembles. They were not able to improve upon static OB-AH (0.74/0.55). BAG-S was better on average only than very poorly performing LNSE (0.66/0.43) and ADOB (0.63/0.38), which was very often the worst algorithm (0.47 of cases). BAG-D almost always occupied the last positions (0.80 of cases for both metrics).

9.3 Summary

In this chapter, we proposed the online clustering-driven ensemble framework exploiting dynamic diversification. Using fully-controlled synthetic streams, we investigated different approaches to diversity control and intensification, as well as, we combined both methods and evaluated all configurations. As a summary based on the presented results, we can conclude the following.

- The clustering-driven diversification based on the dynamic range parameter is able to provide reactive and efficient diversification. The presented results also confirm observations made in [120], that high diversity during changes improves adaptation and that smooth reduction of diversification after drifts helps with restoring stability.

- The stability-oriented or at least linear control is preferable in our framework over the high-diversity one. The former can be still sensitive to changes and recover sufficiently fast, while the latter does not provide any significant improvement during drifts and, at the same time, it impedes adaptation by overdosing diversity when stable periods emerge.

- The intensification methods, based on learning from duplications of single instances, may significantly improve recovery from drifts. The error-driven method, which intensifies diversification and leaves stabilization on a regular level, provides such enhancements over the basic algorithm without intensification, while still being competitive during stable periods. Applying this mechanism to the whole stream results in disturbed diversification control and impaired adaptation.

- The best proposed configuration of our framework is able to outperform most of the considered well-known and cited ensembles or, at least, to be competitive. It proves that the abilities of our framework displayed and analyzed on synthetic streams are also present in realistic scenarios.

In our future works, we plan to apply different drift-aware clustering methods and more sophisticated intensification approaches, as well as analyze dynamic diversification in the context of imbalanced streams.

CHAPTER 10

REACTIVE SUBSPACE BUFFERS FOR CONTINUAL LEARNING FROM NON-STATIONARY DATA

Standard continual learning methods for stationary data are built around the assumption that once learned knowledge should be remembered and stored in the model as long as possible. They assume that previously gathered information stays permanently valid (stationary), which is usually reflected in the experiment design, e.g., through scenarios considering subsequently incoming classes or tasks that do not appear ever again [25, 10]. Unfortunately, the given assumption does not always hold true since modern dynamic data sources may be affected by concept drift, as depicted by the example of a binary recommendation system presented in the introduction of this dissertation. Continual learning models should not only retain useful knowledge but also be ready to update outdated concepts.

The adaptation to concept drifts is the main goal and domain of data stream mining algorithms. As shown in the previous chapters, such models are usually focused on tackling the most recent learning problems (from multiple perspectives) since in the standard streaming setting their evaluation is based solely on the most recent instances – we simply aim at maximizing streaming metrics and do not care about instances that do not come to the model [22, 144]. While this assumption can be valid in many cases, it makes such algorithms vulnerable to catastrophic forgetting in scenarios where labeling of different concepts is not more or less uniformly distributed in time, as shown for streaming decision trees in Chapter 4.

In general, the weaknesses and lack of universality of continual learning algorithms calls for developing methods and benchmarks that can bridge the gap between continual learning from stationary data (knowledge retention) and algorithms focused on reacting to non-stationary distributions (concept drift adaptation) [8].

In this chapter, we propose a holistic approach to class-incremental continual learning, based on experience replay. The novelty of our work is that our algorithm allows for both avoiding catastrophic forgetting and updating previously learned classes if they are affected by concept drift. We distinguish three vital aspects of the proposed framework: (i) capability for class-instrumental continual learning; (ii) capability for retaining useful knowledge to mitigate catastrophic forgetting; and (iii) capability for adaptation to changes by forgetting outdated knowledge and updating the model. Our approach combines centroid-driven memory for storing class-based prototypes with a reactive subspace buffer that can detect and react to concept drift affecting one or more classes. It traces the dominant class in each of the clusters, allowing for switching labels among clusters and splitting them whenever local changes are being detected. We simultaneously ensure the diversity of information stored within class buffers with their reactivity to concept drifts. In our evaluation, we consider a realistic and illustrative learning scenario – continual preference learning and recommendation. To the best of our knowledge, it is the first benchmark explicitly designed to evaluate holistic continual learning methods.

10.1 Class-incremental experience replay under concept drift

The prevalent majority of the class-incremental methods based on experience replay focus on storing the most representative instances or prototypes for stationary data [189]. They rely on the assumption that classes of the observed and selected instances cannot change, therefore there is no need to control them. As a result, the instances picked for a given class will remain in its buffer for a very long time and the only criterion which may trigger their removal or replacement will be the representativeness or diversity of the memory [250]. However, in many real-world applications the mentioned assumption does not hold true. In the presented example of a binary recommendation system, the preferences may change, invalidating some of the experiences stored in the buffer. In such a case, we have to address the concept drift problem and update our memory adequately.

In the following sections, we introduce two commonly used basic algorithms – class buffers and centroid-driven memory – in the context of the given problem. We also propose an adaptive experience replay approach capable of adapting to concept changes.

10.1.1 Class buffers

Standard experience replay methods tackle the catastrophic forgetting problem by storing a separate buffer per class. They assume that a label of an incoming instance is known, so they can perfectly balance the storage and reasonably diversify their available memory. Due to practical concerns, the class buffers have limited capacity, therefore, there is the necessity of selecting which (or if) previously captured instances should be replaced with the currently incoming ones.

The most simplistic approaches use basic algorithms like FIFO (queue) effectively acting as sliding windows [251]. The problem with such methods is that they may very quickly erase the memory of earlier examples, that may be representative for a given problem, leading to catastrophic forgetting [252]. Assuming that incoming instances may be somehow correlated in time, one possible modification mitigating this issue is to enforce a wider spread of the stored instances across time. To achieve that, we can sample instances stored in the buffer, using a simple formula:

$$r \sim \mathcal{U}(0, 1) < \tau, \tag{10.1}$$

where r is a random variable sampled from the uniform distribution and τ is a threshold specified by a user. By increasing the threshold we can enforce quicker replacements, while, on the other hand, by decreasing it we can make the buffer more conservative. Too low τ will lead to impaired learning from new data, while too high τ will inevitably lead to catastrophic forgetting. Although balanced thresholds should be preferable for classic stationary scenarios, such approaches will fail when concept drift occurs, imposing unnecessary avoidance of forgetting and impeding adaptation to changes.

10.1.2 Centroid-driven memory

Usually, the simple class buffer methods are too simplistic, since they do not utilize any significant characteristics of the incoming data. This is especially important when we have to deal with complex or high-level abstract classes (e.g., in recommendation) since it is very likely that there are several different subspaces that should be covered by the maintained buffers. This is where the clustering methods can be found very useful [253]. They are usually utilized to diversify the replay buffer by grouping instances into differing groups, which should result in better coverage of the decision space [254]. The centroids can be used as instances themselves (prototypes) [255], [256], or solely as representations of buffers to forward new examples to their similar memory cells [257]. In this work, we focus on the latter approach.

Although the centroid-driven approaches are one step further than the simple class buffers, they are still susceptible to concept drifts. The reason for that is the fact that they very often do not check whether previously created clusters are still valid for a given class. If a given subconcept cluster changes its label (e.g., from liking to disliking), the new incoming instances will start (slowly) updating class centroids for another class. However, they will not affect the old cluster for the previous class, since the new instances will not be identified as those belonging to it, leaving it obsolete and impeding the learning process when one samples from it. This will, once again, lead to the opposite of catastrophic forgetting, resulting in much slower or non-existent adaptation to the current concepts.

10.1.3 Reactive subspace buffer

To address the presented problem, we propose a modification of the clustering-driven replay buffers, called **Reactive Subspace Buffer (RSB)**, capable not only of efficient knowledge aggregation but also of adequate forgetting when it is needed. The outline of the algorithm is given in Algorithm 9. More details can be found in our public repository: github.com/lkorycki/rsb.

In the given algorithm, for each new instance $\boldsymbol{x}$ with a label y, we first ensure that there are at least c_{min} centroids for the class. Then, we find the nearest cluster $\boldsymbol{C}_x$ for the given instance $\boldsymbol{x}$. If the given cluster belongs to the class y of the instance, we simply update it, its buffer $\boldsymbol{B}_x$ of maximum size b_{max} and sliding window $\boldsymbol{W}_x$ of maximum size ω_{max}, where the last component is responsible for tracking the most current concepts for the given centroid. Otherwise, there is a risk that a concept drift appeared and instances of a different class have started appearing around the centroid. Therefore, if the instance $\boldsymbol{x}$ is sufficiently close (we use simple standard deviation rules), we update the sliding window of the centroid $\boldsymbol{C}_x$, but not the cluster itself. Now, if we detect that there is a significant number of instances with labels different from the current label of the centroid, we switch it to the new majority class. By doing so, we allow the buffer to quickly react to a potential drift. Otherwise, we

Algorithm 9: Reactive Subspace Buffer (RSB).

```
Data: min centroids $c_{max}$, max centroids $c_{max}$, buffer size $b_{max}$, window size
      $\omega_{max}$
Result: replay buffers $\boldsymbol{B}$ at every iteration
repeat
    receive incoming instance $\boldsymbol{x}$ and its label $y$;
    if $c_y < c_{min}$ then
        add new centroid $\boldsymbol{C}_{new}$, buffer $\boldsymbol{B}_{new}$ and window $\boldsymbol{W}_{new}$ for $y$;
        continue;
    find the closest centroid $\boldsymbol{C}_x$ for $\boldsymbol{x}$;
    if $y_{\boldsymbol{C}_x} == y$ then
        update centroid $\boldsymbol{C}_x$, buffer $\boldsymbol{B}_x$ and its window $\boldsymbol{W}_x$ with $(\boldsymbol{x}, y)$;
    else if $\boldsymbol{x}$ is within $\boldsymbol{C}_x$ then
        update window $\boldsymbol{W}_x$ with $(\boldsymbol{x}, y)$;
        if should switch $\boldsymbol{C}_x$ then
            move $\boldsymbol{C}_x$ to centroids of $y$ and update it using $\boldsymbol{W}_x$;
    else
        find the closest centroid $\boldsymbol{C}_{y,x}$ for $(\boldsymbol{x}, y)$;
        if $\boldsymbol{x}$ is within $\boldsymbol{C}_{y,x}$ or $c_y \geq c_{max}$ then
            update centroid $\boldsymbol{C}_{y,x}$, buffer $\boldsymbol{B}_{y,x}$ and its window $\boldsymbol{W}_{y,x}$ with
              $(\boldsymbol{x}, y)$;
        else
            add new centroid $\boldsymbol{C}_{new}$, buffer $\boldsymbol{B}_{new}$ and window $\boldsymbol{W}_{new}$ for $y$;
    check for splits and removals
until stream ends;
```

find the closest centroid $\boldsymbol{C}_{y,x}$ belonging to the same class y as $\boldsymbol{x}$ and we either update it, if $\boldsymbol{x}$ is sufficiently close to the cluster and the maximum number of clusters c_{max} has not been reached, or create a new centroid for the given class y.

Finally, for each centroid $\boldsymbol{C}$, after every n_s-th update of its sliding window $\boldsymbol{W}$, we check whether it did not switch labels but is impure enough to be split into two separate classes. We apply a simple formula checking if $c_1/c_2 - 1.0 < \tau_s$, where c_1 and c_2 are the first and second most numerous classes in the cluster and τ_s is a threshold determined by a user. During this step, we also get rid of minuscule clusters for which less than $\tau_r = \alpha_r \omega_{max}$ instances were registered, where α_r is set by a user.

The whole algorithm is then used as a part of the experience replay method, in which we attempt to sample one instance for each centroid $\boldsymbol{C}$ from its buffer $\boldsymbol{B}$, based on the purity criterion:

$$\gamma_{\boldsymbol{C}} = \tanh(\beta \frac{c_1 - c_2}{c_1 + c_2}) > r \sim \mathcal{U}(0,1), \tag{10.2}$$

where c_1 and c_2 are, once again, the most numerous classes in the cluster, and $\beta = 4$. By doing so, we provide an additional mechanism preventing us from enhancing outdated or at least uncertain concepts. Finally, since by using probabilistic sampling we make the total number of sampled instances non-deterministic, we apply oversampling to balance the selected batch.

To summarize – by enabling: (i) tracking the current dominant classes in a given cluster, (ii) switching labels between clusters, (iii) splitting them, and (iv) sampling from the replay buffer based on clusters purity, we make the centroid-driven algorithm sensitive to concept changes. At the same time, by maintaining stable replay buffers for subspaces that do not change, we can still avoid catastrophic forgetting. As a result, we are able to obtain a method capable of both remembering what is valid and forgetting what is outdated. In addition, since our method is based on local buffers, it should be able to efficiently diversify more complex concepts without explicit knowledge of its subconcepts.

10.2 Experimental study

In the experimental study, we attempt to prove that our algorithm is capable of class-incremental learning from stationary and non-stationary data. We aim to show that it can both (i) avoid catastrophic forgetting by maintaining diversified subspace-oriented replay buffers, and (ii) adapt to concept drifts by forgetting outdated information. All of the presented experiments can be conducted using scripts provided in the mentioned repository.

10.2.1 Data

To evaluate the proposed algorithm we decided to simulate a binary recommendation system by assigning superclasses (0/1) to the classes from the original datasets. By doing so we could simulate the situation in which a user likes or dislikes certain types of available images (subconcepts). We constructed two types of class-incremental datasets: **stationary** and **drifting**.

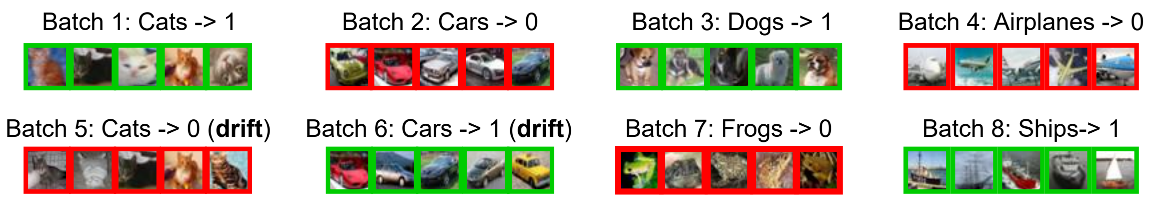


Fig. 61.: General idea of the design for the drifting benchmark sequences.

For the former, we simply used five image benchmarks: MNIST, FASHION, SVHN, CIFAR10 and IMAGENET10, which is a subset of the 64x64 ImageNet set. During the evaluation we were feeding our models class after class, interleaving 0/1 assignments (for example, the first class from CIFAR10 was 1, the second one was 0, and so on). For the latter scenario, we were changing the 0/1 labels for two consecutive classes after three or four stationary ones. As a result, we obtained 30 batches of classes for each dataset, representing both stationary and concept drifting periods (for more details please refer to the repository). An example of our approach is depicted in Fig. 61.

10.2.2 Algorithms

We evaluated our algorithm as a part of the experience replay framework. The module consisted of a classifier (a neural net) and the replay buffer, which was used to sample additional instances for a given input batch. In order to compare our method (**RSB**) with other mentioned approaches, we run experiments using four additional classifiers: (i) offline neural network retraining after each batch (**OFFLINE**), (ii) a naively fine-tuning neural network (**NN**), which learned from the batches without any additional mechanisms for handling catastrophic forgetting, (iii) a simple class buffer (**CB**), which stored separate buffers for both recommendation classes, and a centroid-based method that utilized an on-line k-means algorithm to create representations of the original classes treated as subconcepts (or subspaces) of the recommendation space (**SB**).

While configuring our method, we used the following values of its parameters: $c_{min} = 0.5c_{max}$, where $c_{max} = 10$ for all of the datasets except for FASHION for which we set $c_{max} = 20$ based on preliminary experiments. Each buffer of the method could store at most $b_{max} = 100$ instances and equal was the size of each sliding window $\omega_{max} = 100$. We also empirically set $n_s = 1000$ and $\tau_s = 0.5$ for splitting, and $\alpha_r = 0.4$ for removals. These settings worked very well with all of the considered datasets. We used the same values of c_{max} and b_{max} for the SB algorithm. When it comes to the CB method, we set $b_{max} = 2000$ per class to provide similar memory resources compared with RSB and SB. Furthermore, we distinguished CB with $\tau = 0.0$ ($\mathbf{CB}_{old}$) and $\tau = 1.0$ ($\mathbf{CB}_{new}$) to check their performance in stationary and non-stationary scenarios. We can say that CB_{old} represents standard continual learning methods for stationary data (it indefinitely stores once observed instances), while CB_{new} represents streaming approaches (constantly replaces examples on the fly).

All of the mentioned algorithms used pre-trained convolutional feature extractors, from which we used representations returned by a middle layer of the classifier (we needed a high-level representation due to the nature of our task). For MNIST and FASHION we used a simple CNN with two convolutional layers consisting of

32 (5x5) and 64 (3x3) filters, interleaved with ReLU, batch normalization and max pooling (2x2). For SVHN, CIFAR10 and IMAGENET10 we utilized ResNet18. As a trainable classifier we chose a 3-layer fully connected net with 512, 256, 128 neurons in the hidden layers interleaved with ReLU, batch normalization and dropout ($p = 0.5$). During training, we used the Adam optimizer. After each batch, the classifier learned for either 5 epochs (IMAGENET10) or 10 (the rest). Additionally, we initialized each algorithm with 10% of the first and the second class.

10.2.3 Evaluation

We evaluated the presented methods in a class-incremental setting, where each original class is treated as a subconcept of the binary recommendation space and comes as a whole in a form of a batch (Fig. 61). In our scenario, we assume that old subconcepts may become outdated and batches may change their labels. We measured the accuracy of a given algorithm after each batch (a new or updated class), utilizing holdout testing sets, and then, based on [12], used it to calculate the normalized average accuracy over the whole sequence:

$$\Omega_{all} = \frac{1}{T} \sum_{t=1}^{T} \frac{\alpha_t}{\alpha_{\text{offline},t}}, \tag{10.3}$$

where α_t is the model performance after t classes and $\alpha_{\text{offline},t}$ is the optimal performance obtained by the offline learner.

To make our scenario more challenging, we assumed that we did not know the classes of the original set, only the recommendation labels. This allowed us to create a complex decision space without explicit knowledge of its subspaces and with a lot of potential for local concept drifts.

10.2.4 Results

Performance on stationary continual learning. Firstly, we evaluated the performance of RSB against the reference approaches in class-incremental continual learning with stationary properties. That means there was no concept drift present in the data and the main challenge lay in aggregating learned knowledge and avoiding catastrophic forgetting. We used this scenario first as an ablation study, to show that RSB is capable of learning newly arriving classes, without forgetting the previously seen ones. Tab. 30 shows the normalized average accuracy results over the five used benchmarks, while Fig. 62 depicts the changes in accuracy over time, calculated after each class (subconcept) batch. We omit the MNIST plot as it has identical characteristics as the FASHION plot.

In the presented results, we can see that all of the considered experience replay approaches were able to obtain satisfactory performance on the stationary sequences, slightly below the offline upper bound. They significantly improved upon the naive fine-tuning (NN), which severely suffered from catastrophic forgetting. The simple class buffers performed similarly on average. Holding instances of the earliest classes

Table 30.: Normalized average accuracy (absolute values for the offline baseline) for stationary sequences.

Model	MNIST	FASHION	SVHN	CIFAR10	IMG10
OFFLINE	1.0	0.9865	1.0	1.0	1.0
NN	0.5529	0.5603	0.5529	0.5596	0.4886
ER-CB$_{old}$	0.9537	0.9554	0.9414	0.9106	0.8828
ER-CB$_{new}$	0.8754	0.8990	0.9235	0.9298	0.9349
ER-SB	0.9897	0.9739	0.9750	0.9675	**0.9513**
ER-RSB	**0.9967**	**0.9926**	**0.9816**	**0.9740**	0.9405

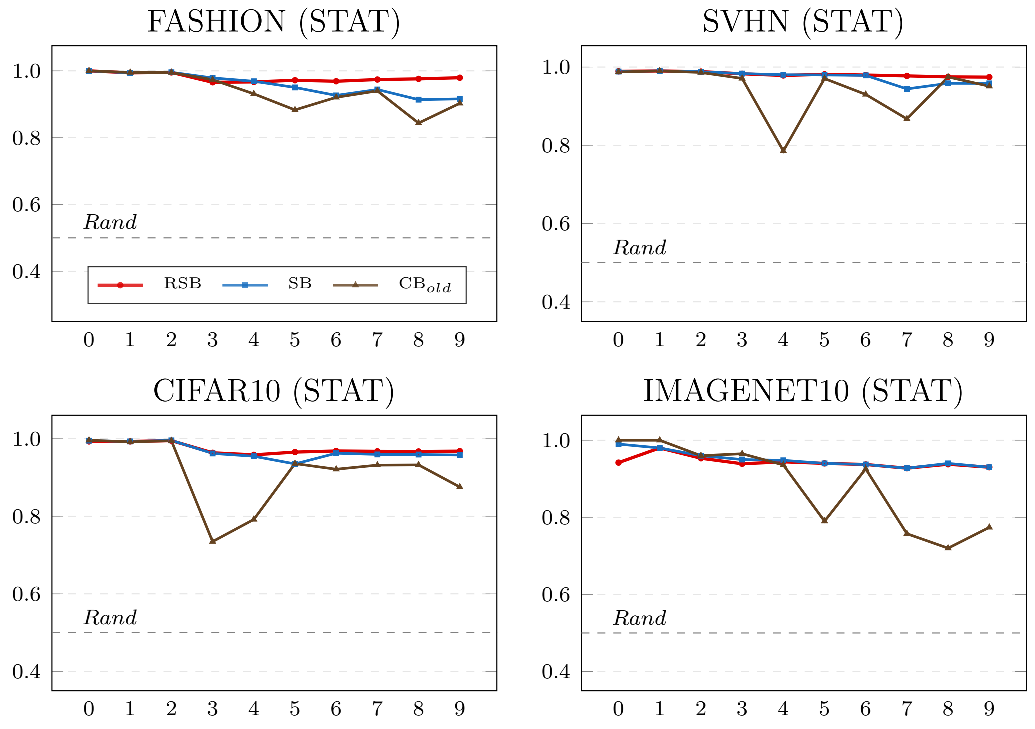


Fig. 62.: Average accuracy over all classes for stationary class-incremental sequences.

(CB$_{old}$) turned out to be a bit better approach on simpler benchmarks, while giving a higher priority to the newer instances (CB$_{new}$) resulted in higher accuracy on CIFAR10 and IMAGENET10. The more sophisticated centroid-driven experience replay (SB, RSB) provided even higher quality on all sequences by maintaining more diversified memory buffers per recommendation class. Finally, the results indicate that our method is often capable of improving upon the simpler centroid-based method (SB), most likely by correcting partially inaccurate clusters.

In Fig. 62 we can clearly see that RSB displayed stable incremental learning capabilities and was not affected by catastrophic forgetting. This is especially visible on FASHION, SVHN and CIFAR10 datasets, where with the increasing number of classes reference methods displayed drops of performance, while RSB achieved stable results for all arriving classes. For SVHN, we can see that CB$_{old}$ returned to similar performance as RSB after the 8-th class – but the intermediate learning process between classes no. 4 and 8 was significantly impaired. SB was much more resilient to forgetting, yet it performed slightly worse than RSB on 3 out of 4 sequences and on average. This allows us to conclude that RSB is robust to both catastrophic forgetting and false concept drift detection on stationary data.

Performance on continual learning under concept drift. After establishing that RSB displays robustness to catastrophic forgetting, we needed to evaluate its capability of simultaneous incremental learning and adaptation to drift. We used the same five benchmarks that now were injected with concept drift as discussed in Sec. 5.1. This way we should be able to see if RSB is able to detect changes in previously learned classes and correctly modify the underlying classifier to update its stored knowledge. Tab. 31 shows the normalized average accuracy results over five used benchmarks, while Fig. 63 depicts the changes in accuracy over time. Again, we omit MNIST plot as it has identical characteristics as the FASHION plot.

For continual learning under concept drift we can see significant differences among the examined algorithms. Neither CB$_{old}$, CB$_{new}$ or SB was capable of keeping

Table 31.: Normalized average accuracy (absolute values for the offline baseline) for drifting sequences.

Model	MNIST	FASHION	SVHN	CIFAR10	IMG10
OFFLINE	1.0	0.9744	1.0	1.0	1.0
NN	0.5894	0.6043	0.5872	0.5884	0.4546
ER-CB$_{old}$	0.5977	0.6473	0.5494	0.5635	0.6084
ER-CB$_{new}$	0.7422	0.7931	0.7743	0.7918	0.8540
ER-SB	0.7268	0.7341	0.7267	0.7004	0.6696
ER-RSB	**0.9938**	**0.9745**	**0.9722**	**0.9545**	**0.9187**

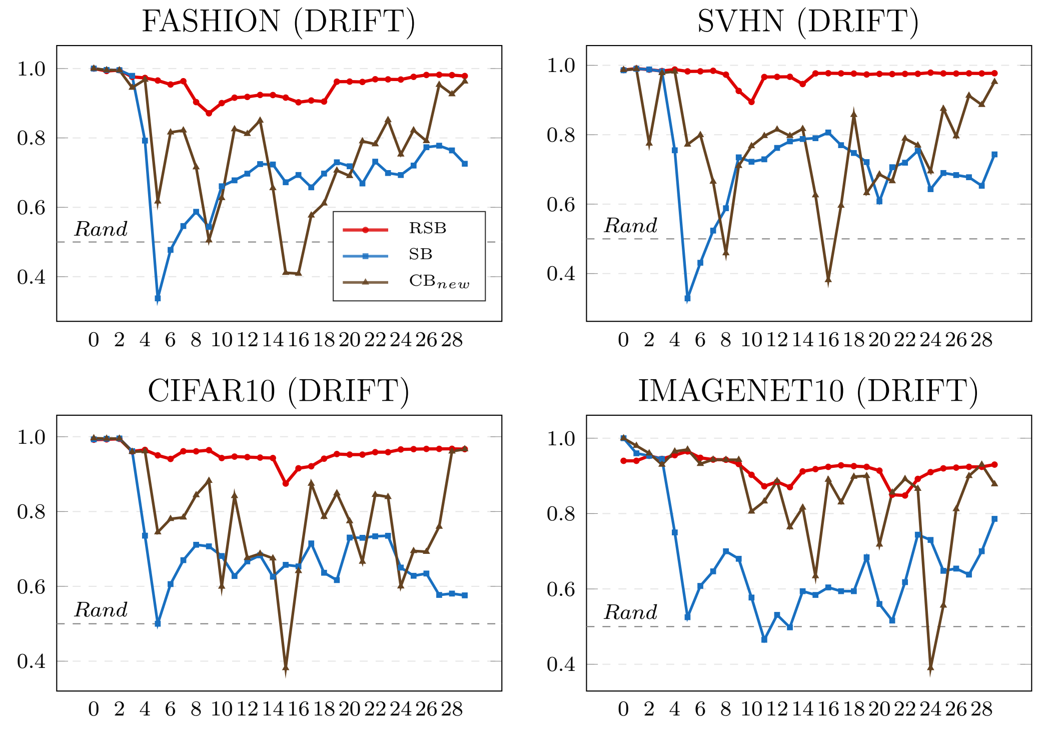


Fig. 63.: Average accuracy over all classes for drifting class-incremental sequences. Drifts occur in batches 4, 5, 9, 10, 14, 15, 19, 20, 24 and 25.

up with the presence of concept drift in the data. The main reason for that was the fact that CB$_{old}$ and SB kept outdated instances in their buffers, impeding the adaptation process by forcing the model to retain obsolete concepts. On the other hand, CB$_{new}$ adapted to newer concepts much better than CB$_{old}$, but it was not able to store instances for older classes, which inevitably led to catastrophic forgetting. For all five datasets the proposed RSB displayed the most stable performance, which is especially striking in the case of FASHION, SVHN and CIFAR10 sequences. By analyzing the plots we can see how the reference methods were significantly impacted by the first occurrence of concept drift, often dropping to similar or lower performance levels as the random approach. Sometimes they were slowly recovering their performance over time, but this was happening at an unacceptable rate.

To gain further insights into the performance of the experience replay under concept drift let us look at Fig. 64 that depicts the accuracy over selected drifting classes. We can see that both CB$_{new}$ and SB were highly sensitive to any drift in data. Even if sometimes they could spontaneously recover their performance (which usually was rather a coincidence), the next occurrence of concept drift could easily bring their performance back to the level of random decision (or even below). In the case of the MNIST class 0 we can see that the SB method could not recover at any point of time after the first drift. The extremely low accuracy was caused by obsolete centroids, which did not update their label and kept generating invalid instances for the recommendation class. These results clearly indicate that standard experience replay approaches cannot handle concept drifts, and that some of the occurring errors may even never be corrected. On the contrary, the proposed RSB is characterized by excellent robustness to concept drift, stable performance, and on-the-fly adaptation to changes in previously learned classes without any delay or loss in predictive power.

Finally, we should be aware that concept drift may affect not only the performance of models on previously seen classes, but also their incremental learning capabilities. As the underlying neural network model tries to handle the catastrophic

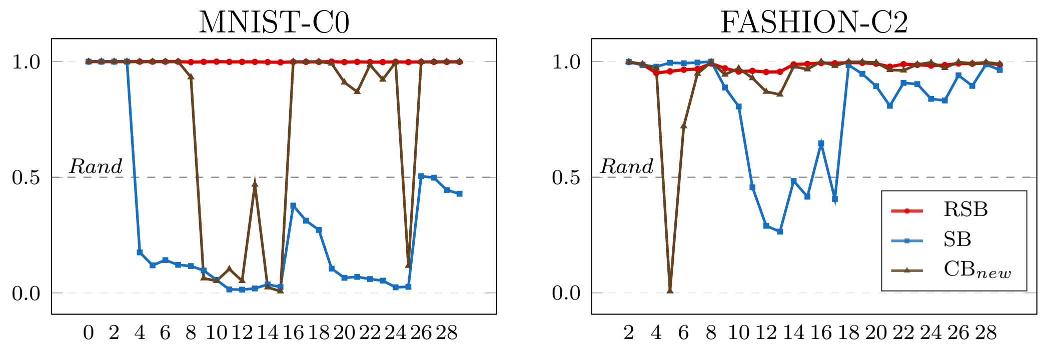


Fig. 64.: Accuracy for the selected classes under concept drift. C0 drifts in batches 4 and 5, and C2 drifts in 9 and 10.

forgetting by using instances from the buffer for experience replay, it utilizes instances coming from outdated concepts that may be contradicting the most current ones. Therefore, this may impact its ability to incorporate and retain new knowledge, resulting in a significant decrease in the model's predictive power. This allows us to conclude that continual learning under concept drift requires a strong interplay between avoiding catastrophic forgetting and adaptation to concept drift, as weaker performance on one will negatively affect the other. The proposed RSB offers an excellent balance between these two tasks, leading to a well-rounded and stable continual learning solution.

10.3 Summary

In this chapter, we have discussed a unified approach to continual learning that bridges the gap between avoiding catastrophic forgetting and data stream mining under concept drift. By pointing to the fact that these fields are two faces of the same coin, we showed that there is a need for developing holistic systems that are capable of incremental incorporation of new information while offering adaptation capabilities by selective forgetting. This was illustrated by a practical example of continual learning of user's preferences that expand and evolve over time.

To address this challenging scenario, we have proposed an experience replay approach based on a reactive subspace buffer. It combines clustering-driven memory, storing diverse instances per class, with adaptation components that allow for dynamic monitoring, relabeling, and splitting of existing clusters. As a result, our method provides both the capability of accommodating new classes without catastrophic forgetting and the ability to react to concept drift affecting the previously learned classes. In our experimental study, we exhibited the effectiveness of our algorithm and proved that it is an effective and complete approach to continual learning that is not limited by either inability to accommodate new information, or by the inability to adapt to changes.

We have shown that while existing standard experience replay approaches are able to handle the problem of avoiding catastrophic forgetting, they do not possess mechanisms allowing for adaptation in previously learned classes affected by concept drift. We suppose that similar issues can be identified in other continual learning algorithms. Therefore, our future works will focus on improving different approaches. This may involve, for example, introducing adaptive masking, reactive regularization and dynamic neural network structures capable of reacting to drifts. These will be important steps towards creating a holistic view of continual learning systems that can handle diverse challenges present in various real-life problems.

CHAPTER 11

FINAL SUMMARY

11.1 Conclusions

Continual learning is a challenging and vast research domain mainly because we have to deal with an unlimited continuum of events, having strictly limited resources and knowledge. Also, by enforcing incremental capabilities we open a new dimension of problems that add more complexity to many of the issues known from the standard offline settings.

The presented dissertation addressed these problems in an exploratory way. Its main goal was to look at the continual learning paradigm as a whole, analyze its various branches and address identified issues covering different aspects of learning from sequentially incoming data. By doing so, this work not only filled several gaps in the current continual learning research but also emphasized the complexity and diversity of problems in this domain, encouraging (hopefully) the readers to look at this concept from multiple perspectives.

The prevalent part of this work focused on addressing problems related to either catastrophic forgetting or concept drift adaptation. For the former, we showed that deep continual learning can be successfully extended to alternative machine learning models (mixture models and decision trees), potentially stimulating new research directions. For the latter, we presented works tackling compound problems that can be easily encountered when learning from infinite non-stationary data streams – effective adaptation to concept drift under strictly limited supervision, dealing with non-stationary imbalanced streaming environments, detecting changes in dynamic skewed distributions and in unsupervised settings, or exploiting adjustable diversity for the purpose of ensemble adaptation. Last but not least, we introduced a holistic method for continual learning, highlighting how important it is to design algorithms and experiments that explicitly address both catastrophic forgetting and change adaptation in order to provide reliable and robust continual learning solutions. Extensive evaluations conducted for all of the presented contributions have substantiated their effectiveness and supported the validity of the introduced suppositions.

We hope that this dissertation will help others with achieving the main continual learning goals, including convergence to offline baselines in stationary scenarios and full extension of machine learning paradigms to more universal non-stationary environments.

11.2 Open challenges and future directions

While the presented dissertation covered multiple different aspects and issues of continual learning, the enormous challenge, that the problem poses as a whole, leaves no doubt that it is just the tip of the iceberg that should be revealed more in the coming years of research. In addition to more specific future works suggested for each of the methods presented in the previous chapters, this final section extends them and introduces other open challenges research directions identified by us.

11.2.1 Improving continual learning from stationary data

Current issues that can be addressed in the context of advancing methods providing stability for static concepts, flexibility when incorporating new stationary data and scalability with the number of new concepts include the following directions.

Streaming. The prevalent number of works on continual learning assume that new data come in the form of finite batches either per class (class-incremental) or task

(task-incremental) [9, 10, 61]. This is a very strong assumption since in many practical applications it is not possible to define a discrete and finite set representing a class or task once and for all. It is more likely that a user will be aggregating the data across time, without complete information about classes and tasks, and without a specific order (data-incremental). This is why to significantly increase universality of continual learning systems, we have to enable fully streaming processing, in which instances come one by one, without any additional assumptions.

Task and class imbalance. The problem of handling imbalanced data in continual learning has not been properly addressed, making such systems susceptible to imbalanced/biased distributions. This issue cannot be simply boiled down to applying the same solutions as those known for standard deep learning, since besides balancing the learning process, we also have to be careful not to forget older information. The latter problem has been already addressed in the context of imbalance and is known as a bias between new and old classes [62, 63]. Combined with the standard imbalance (class ratios) it constitutes a non-trivial problem requiring dedicated solutions in a form of more sophisticated loss functions (e.g. combining knowledge distillation with ratio-based weighting) and oversampling techniques (e.g. GAN-based). This is a critical problem for many real-world applications struggling with the issue of robustness of models.

Continual ensembles. Committees seem like a natural candidate for continual learning since each member of the committee can be treated as a memory cell that aggregates different portions of the incoming knowledge. However, the utilization of ensemble techniques in continual learning remains almost completely unexplored for neural networks. While it is clear that using several deep neural nets may be unrealistic, especially when confronted with a potentially infinite surge of data, the idea of building intrinsic committees, in which internal subnets act as base learners, or creating ensembles of lightweight output layers looks like a feasible and powerful concept. There are but few works that introduced simple foundations for such research by, for example, extracting members of the ensemble by masking, or utilizing an autoencoder to select the most competent base learner for a given task. These basic frameworks suffer from critical limitations that should be addressed. Good results of ensembles dedicated to data stream mining [22, 116] should provide further encouragement towards exploring this area in the context of incremental learning from stationary data.

Continual transfer learning. One of the fundamental reasons why we may want to use continual learning systems is their desired ability to enhance the creation of new models with previously collected knowledge (forward transfer) and, at the same time, to enrich utilized old models with new data whenever it is available (backward transfer) [12]. Unfortunately, most of the proposed works offer the former to some limited extent (e.g. only from a common backbone net in the mentioned ensembles), and the latter is practically non-existent (e.g. in the masking techniques). Furthermore, continual learning seems to lack mechanisms for avoiding forced transfer from incompatible concepts. It is important, as in the considered scenarios we can never know what type of data we will encounter and if we should try to transfer the knowledge to the maintained models.

More hybridization. The main problems with aggregating knowledge in deep learning models are strongly connected with the way neural networks learn. In this work, we showed that it is possible to successfully hybridize deep neural networks with other machine learning models that may be inherently resilient to catastrophic forgetting, e.g. mixture models as classifiers. Therefore, a natural compelling idea should be to consider using different machine learning models in continual learning scenarios. One should also remember that it does not have to be limited only to classification layers – these methods could be potentially mixed into the whole deep learning architectures, including feature extractors [258].

11.2.2 Improving continual learning from non-stationary data

Important issues related to improving the quality and flexibility of methods designed for handling adaptation to concept drifts include, but are not limited to, the topics given below.

Learning on a budget. One of the primary issues that can be identified, is insufficient attention given to the problem of supervision availability in streaming learning. While we addressed this problem at the level of supervised and some unsupervised approaches, there is also a very important domain of semi-supervised methods that theoretically seem to be designed for such cases as learning from data streams (having a vast amount of unlabeled instances and only few annotated). The utilization of such methods could potentially with improving recovery rates after drifts, without additional labeling costs. However, as pointed out by us and shown in one of our additional works, semi-supervised techniques suffer from the non-stationary properties of data [203], therefore, it is an open question how to adapt them to such scenarios. Furthermore, few-shot or zero-shot learning approaches, known mainly from the offline deep learning domain, could also find a lot of applicability here [83].

Concept drift detection. As mentioned in our works dedicated to the problem of drift detection, it may be extremely useful to have access to detectors capable of detecting changes in an unsupervised way and sensitive to local drifts [124]. Taking that into consideration, designing detectors capable of recognizing multivariate changes in feature subspaces without supervision seems like an important issue that is definitely worth further exploration. Furthermore, we can say the same about designing explainable change detection allowing for a better understanding of dynamics present in data (all of the published detectors are black boxes). Finally, because the proposed drift detectors were not designed specifically for deep neural networks, it is possible that there is a great potential for finding more precise, localized and accurate detection algorithms for deep architectures by utilizing information coming from individual layers or groups of neurons.

Proactive approaches. The majority of algorithms designed for non-stationary data exhibits solely reactive adaptation properties. This means that they will adapt to observed changes only post-factum, which will usually result in significant adaptation delays and prolonged periods of a model's incompetence. An alternative approach assumes the presence of proactive techniques capable of predicting that a change may occur and preparing a model for the drift before it actually appears [177]. Such methods focus not only on modeling the concepts themselves but also on creating models for the changes (dynamics) observed in the data across time – we can call it *change mining*. It is almost a completely unexplored area with a great potential for novelty and substantial advancements in contemporary machine learning models.

Addressing the lack of holistic approaches Combining methods for tackling catastrophic forgetting and concept drift within a single learning algorithm can be seen as one of the ultimate goals of continual learning. In practice, the final models should be capable of effectively distinguishing what to remember and what to forget without negatively affecting either of these abilities. The gap in the research related to the given problem, introduced in this work, can be further addressed by preparing unifying studies comprehensively analyzing both considered scenarios from the perspective of a shared goal [8], proposing new evaluation approaches, or designing dedicated algorithms capable of handling the complex issue. The last part may consist of: adjusting methods designed for the stabilization of stationary concepts to make them suitable for non-stationary data, or adjusting data stream mining models to tackle the catastrophic forgetting issue. The outcome of these attempts should be a group of generic unified continual learning algorithms capable of incorporating new concepts, remembering those that do not change (tackling catastrophic forgetting) and updating those that become outdated (tackling concept drift). The algorithms should provide equilibrium between stability and complete flexibility, offering a holistic approach to next-level continual learning.

Benchmarks and applications. Last but not least, it can be easily noticed that there is a lack of real-world applications of continual machine learning algorithms. We think that it is important for the popularization of the proposed methods to exhibit their abilities to handle realistic problems at each stage of the research. Some of the potential ideas include: medical applications (learning new cases, patients, evolving diseases etc.), recommendation systems (e.g. user's taste is subject to more or less frequent changes) or models for social media (unstable concepts, data streams). Furthermore, one of the main problems that may be slowing down the development of continual models is the limited scope of data benchmarks that are taken into consideration. For example, if we use only data from such classification tasks as MNIST, CIFAR or ImageNet, then we will never recognize the need of developing continual learning models (and metrics) that can handle concept drifts, since nines will always be nines and cats will always be cats. Therefore, inspired by collections like [259], it would be a great idea to prepare real-world benchmarks consisting of concept drifts and make them public.

References

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

[2] Scott Coyne, Praveen Madiraju, and Joseph Coelho. "Forecasting Stock Prices Using Social Media Analysis". In: *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress*. 2017, pp. 1031–1038.

[3] Sorin Mihai Grigorescu et al. "A Survey of Deep Learning Techniques for Autonomous Driving". In: *arXiv* abs/1910.07738 (2020).

[4] Yang Yan, Bin Wang, and Jun Zou. *Blockchain*. WORLD SCIENTIFIC, 2021. DOI: 10.1142/12264.

[5] Tim Sweeney. "Foundational Principles Technologies for the Metaverse". In: *ACM SIGGRAPH 2019 Talks*. SIGGRAPH '19. Los Angeles, California: Association for Computing Machinery, 2019. ISBN: 9781450363174.

[6] German I. Parisi et al. "Continual lifelong learning with neural networks: A review". In: *Neural Networks* 113 (2019), pp. 54–71. ISSN: 0893-6080.

[7] Yen-Chang Hsu, Yen-Cheng Liu, and Zsolt Kira. "Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines". In: *arXiv* abs/1810.12488 (2018).

[8] Rudolf Szadkowski, Jan Drchal, and Jan Faigl. "Continually trained life-long classification". In: *Neural Computing and Applications* (2021).

[9] Matthias De Lange et al. “Continual learning: A comparative study on how to defy forgetting in classification tasks”. In: *CoRR* abs/1909.08383 (2019).

[10] Marc Masana et al. “Class-incremental learning: survey and performance evaluation”. In: *CoRR* abs/2010.15277 (2020).

[11] Robert M. French. “Catastrophic forgetting in connectionist networks”. In: *Trends in Cognitive Sciences* 3.4 (1999), pp. 128–135.

[12] Ronald Kemker et al. “Measuring Catastrophic Forgetting in Neural Networks”. In: *AAAI*. 2018.

[13] Ameya Prabhu, Philip H. S. Torr, and Puneet Kumar Dokania. “GDumb: A Simple Approach that Questions Our Progress in Continual Learning”. In: *ECCV*. 2020.

[14] Jie Lu et al. “Learning under Concept Drift: A Review”. In: *IEEE Transactions on Knowledge and Data Engineering* 31.12 (2019), pp. 2346–2363.

[15] Ammar Shaker and Eyke Hüllermeier. “Recovery analysis for adaptive learning from non-stationary data streams: Experimental design and case study”. In: *Neurocomputing* 150 (2015), pp. 250–264.

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

[17] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. “Task-Free Continual Learning”. In: *CoRR* abs/1812.03596 (2018).

[18] Massimo Caccia et al. “Online Fast Adaptation and Knowledge Accumulation (OSAKA): a New Approach to Continual Learning”. In: *NeurIPS*. 2020.

[19] Sergio Ramírez-Gallego et al. "A survey on data preprocessing for data stream mining: Current status and future directions". In: *Neurocomputing* 239 (2017), pp. 39–57.

[20] Jean Paul Barddal et al. "A survey on feature drift adaptation: Definition, benchmark, challenges and future directions". In: *Journal of Systems and Software* 127 (2017), pp. 278–294.

[21] Benedikt Pfülb and Alexander Rainer Tassilo Gepperth. "A comprehensive, application-oriented study of catastrophic forgetting in DNNs". In: *arXiv* abs/1905.08101 (2019).

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

[23] Furqan Rustam et al. "COVID-19 Future Forecasting Using Supervised Machine Learning Models". In: *IEEE Access* 8 (2020), pp. 101489–101499.

[24] Martin Müller and Marcel Salathé. *Addressing machine learning concept drift reveals declining vaccine sentiment during the COVID-19 pandemic.* 2020.

[25] Matthias De Lange and Tinne Tuytelaars. "Continual Prototype Evolution: Learning Online from Non-Stationary Data Streams". In: *arXiv* abs/2009.00919 (2020).

[26] Xiaoyu Tao et al. "Bi-Objective Continual Learning: Learning 'New' While Consolidating 'Known'". In: *AAAI.* 2020.

[27] Łukasz Korycki and Bartosz Krawczyk. "Streaming Decision Trees for Lifelong Learning". In: *Machine Learning and Knowledge Discovery in Databases. Research Track.* Ed. by Nuria Oliver et al. Cham: Springer International Publishing, 2021, pp. 502–518.

[28] Eli Verwimp, Matthias De Lange, and Tinne Tuytelaars. "Rehearsal revealed: The limits and merits of revisiting samples in continual learning". In: *arXiv* abs/2104.07446 (2021).

[29] Sylvestre-Alvise Rebuffi et al. "iCaRL: Incremental Classifier and Representation Learning". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 5533–5542.

[30] Jihwan Bang et al. "Rainbow Memory: Continual Learning with a Memory of Diverse Samples". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, 2021, pp. 8214–8223.

[31] Zalán Borsos, Mojm'ir Mutn'y, and Andreas Krause. "Coresets via Bilevel Optimization for Continual Learning and Streaming". In: *arXiv* abs/2006.03875 (2020).

[32] David Lopez-Paz and Marc'Aurelio Ranzato. "Gradient Episodic Memory for Continual Learning". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Long Beach, California, USA, 2017, 6470–6479.

[33] Arslan Chaudhry et al. "Efficient Lifelong Learning with A-GEM". In: *arXiv* abs/1812.00420 (2019).

[34] Rahaf Aljundi et al. "Gradient based sample selection for online continual learning". In: *NeurIPS*. 2019.

[35] Tyler L. Hayes et al. "REMIND Your Neural Network to Prevent Catastrophic Forgetting". In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. 2020, pp. 466–483.

[36] Sayna Ebrahimi et al. “Remembering for the Right Reasons: Explanations Reduce Catastrophic Forgetting”. In: *CoRR* abs/2010.01528 (2020).

[37] Yaoyao Liu et al. “Mnemonics Training: Multi-Class Incremental Learning Without Forgetting”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 12242–12251.

[38] James Smith et al. “Always Be Dreaming: A New Approach for Data-Free Class-Incremental Learning”. In: *arXiv* abs/2106.09701 (2021).

[39] Timothée Lesort et al. “Generative Models from the perspective of Continual Learning”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. 2019, pp. 1–8.

[40] Amanda Rios and Laurent Itti. “Closed-Loop Memory GAN for Continual Learning”. In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI’19. Macao, China, 2019, 3332–3338.

[41] Mengyao Zhai et al. “Lifelong GAN: Continual Learning for Conditional Image Generation”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 2759–2768.

[42] Xin Su et al. “Generative Memory for Lifelong Learning”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.6 (2020), pp. 1884–1898.

[43] Mohammad Rostami et al. “Generative Continual Concept Learning”. In: *AAAI*. 2020.

[44] Tyler L. Hayes et al. “Replay in Deep Learning: Current Approaches and Missing Biological Elements”. In: *Neural Computation* 33.11 (Oct. 2021), pp. 2908–2950.

[45] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. “Distilling the Knowledge in a Neural Network”. In: *arXiv* abs/1503.02531 (2015).

[46] Zhizhong Li and Derek Hoiem. “Learning without Forgetting”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.12 (2018), pp. 2935–2947.

[47] Saihui Hou et al. “Lifelong Learning via Progressive Distillation and Retrospection”. In: *ECCV*. 2018.

[48] Prithviraj Dhar et al. “Learning Without Memorizing”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 5133–5141.

[49] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526.

[50] Friedemann Zenke, Ben Poole, and Surya Ganguli. “Continual Learning through Synaptic Intelligence”. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML’17. Sydney, NSW, Australia, 2017, 3987–3995.

[51] Rahaf Aljundi et al. “Memory Aware Synapses: Learning What (not) to Forget”. In: *Computer Vision – ECCV 2018*. Ed. by Vittorio Ferrari et al. 2018, pp. 144–161.

[52] Pingbo Pan et al. “Continual Deep Learning by Functional Regularisation of Memorable Past”. In: *arXiv* abs/2004.14070 (2020).

[53] Rahaf Aljundi, Marcus Rohrbach, and Tinne Tuytelaars. “Selfless Sequential Learning”. In: *arXiv* abs/1806.05421 (2019).

[54] Jhair Gallardo, Tyler L. Hayes, and Christopher Kanan. “Self-Supervised Training Enhances Online Continual Learning”. In: *arXiv* abs/2103.14010 (2021).

[55] Alaa El Khatib and Fakhri Karray. “Preempting Catastrophic Forgetting in Continual Learning Models by Anticipatory Regularization”. In: *2019 International Joint Conference on Neural Networks (IJCNN)* (2019), pp. 1–7.

[56] Sayna Ebrahimi et al. “Adversarial Continual Learning”. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. 2020, pp. 386–402.

[57] Arthur Douillard et al. “PODNet: Pooled Outputs Distillation for Small-Tasks Incremental Learning”. In: *ECCV*. 2020.

[58] Jonathan Schwarz et al. “Progress & Compress: A scalable framework for continual learning”. In: *arXiv* abs/1805.06370 (2018).

[59] Amanda Rios and Laurent Itti. “Lifelong Learning Without a Task Oracle”. In: *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)* (2020), pp. 255–263.

[60] Arthur Douillard et al. *DyTox: Transformers for Continual Learning with DYnamic TOken eXpansion*. 2021.

[61] Davide Maltoni and Vincenzo Lomonaco. “Continuous learning in single-incremental-task scenarios”. In: *Neural Networks* 116 (2019), pp. 56–73.

[62] Yue Wu et al. “Large Scale Incremental Learning”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 374–382.

[63] Bowen Zhao et al. “Maintaining Discrimination and Fairness in Class Incremental Learning”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 13205–13214.

[64] Saihui Hou et al. “Learning a Unified Classifier Incrementally via Rebalancing”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[65] Guile Wu, Shaogang Gong, and Pan Li. “Striking a Balance Between Stability and Plasticity for Class-Incremental Learning”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 1124–1133.

[66] Arun Mallya and Svetlana Lazebnik. “Piggyback: Adding Multiple Tasks to a Single, Fixed Network by Learning to Mask”. In: *arXiv* abs/1801.06519 (2018).

[67] Arun Mallya and Svetlana Lazebnik. “PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 7765–7773.

[68] Chrisantha Fernando et al. “PathNet: Evolution Channels Gradient Descent in Super Neural Networks”. In: *CoRR* abs/1701.08734 (2017).

[69] Shivangi Srivastava et al. “Adaptive Compression-based Lifelong Learning”. In: *arXiv* abs/1907.09695 (2019).

[70] Steven C. Y. Hung et al. “Compacting, Picking and Growing for Unforgetting Continual Learning”. In: *arXiv* abs/1910.06562 (2019).

[71] Zixuan Ke, Bing Liu, and Xingchang Huang. “Continual Learning of a Mixed Sequence of Similar and Dissimilar Tasks”. In: *NeurIPS*. 2020.

[72] Andrei A. Rusu et al. “Progressive Neural Networks”. In: abs/1606.04671 (2016).

[73] Jaehong Yoon et al. “Lifelong Learning with Dynamically Expandable Networks”. In: *arXiv* abs/1708.01547 (2018).

[74] Shipeng Yan, Jiangwei Xie, and Xuming He. “DER: Dynamically Expandable Representation for Class Incremental Learning”. In: *arXiv* abs/2103.16788 (2021).

[75] Tom Veniat, Ludovic Denoyer, and Marc’Aurelio Ranzato. *Efficient Continual Learning with Modular Networks and Task-Driven Priors*. 2021.

[76] Xi lai Li et al. “Learn to Grow: A Continual Structure Learning Framework for Overcoming Catastrophic Forgetting”. In: *ICML*. 2019.

[77] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. “Expert Gate: Lifelong Learning with a Network of Experts”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 7120–7129.

[78] Yeming Wen, Dustin Tran, and Jimmy Ba. “BatchEnsemble: An Alternative Approach to Efficient Ensemble and Lifelong Learning”. In: *arXiv* abs/2002.06715 (2020).

[79] Pravendra Singh et al. “Rectification-based Knowledge Retention for Continual Learning”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 15277–15286.

[80] Felix Wiewel, Andreas Brendle, and Bin Yang. “Continual Learning Through One-Class Classification Using VAE”. In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 3307–3311.

[81] Wenpeng Hu et al. “Continual Learning by Using Information of Each Class Holistically”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.9 (2021), pp. 7797–7805.

[82] Ya jun Liu et al. “More Classifiers, Less Forgetting: A Generic Multi-classifier Paradigm for Incremental Learning”. In: *ECCV*. 2020.

[83] Chi Zhang et al. “Few-Shot Incremental Learning with Continually Evolved Classifiers”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 12450–12459.

[84] Xiaoyu Tao et al. “Few-Shot Class-Incremental Learning”. In: *CoRR* abs/2004.10956 (2020).

[85] Kai Zhu et al. “Self-Promoted Prototype Refinement for Few-Shot Class-Incremental Learning”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 6797–6806.

[86] Sung Whan Yoon et al. “XtarNet: Learning to Extract Task-Adaptive Representation for Incremental Few-Shot Learning”. In: *ICML*. 2020.

[87] Ali Cheraghian et al. “Semantic-Aware Knowledge Distillation for Few-Shot Class-Incremental Learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 2534–2543.

[88] Khurram Javed and Martha White. “Meta-Learning Representations for Continual Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019.

[89] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and

Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. 2017, pp. 1126–1135.

[90] Johannes von Oswald et al. “Continual learning with hypernetworks”. In: *CoRR* abs/1906.00695 (2019).

[91] Xu He et al. “Task Agnostic Continual Learning via Meta Learning”. In: *arXiv* abs/1906.05201 (2019).

[92] Wenpeng Hu et al. “Overcoming Catastrophic Forgetting for Continual Learning via Model Adaptation”. In: *ICLR*. 2019.

[93] Shawn Beaulieu et al. “Learning to Continually Learn”. In: *CoRR* abs/2002.09571 (2020).

[94] Tyler L. Hayes and Christopher Kanan. “Lifelong Machine Learning with Deep Streaming Linear Discriminant Analysis”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2020), pp. 887–896.

[95] Ziyang Wu et al. “Incremental Learning via Rate Reduction”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 1125–1133.

[96] Yaoyao Liu, Bernt Schiele, and Qianru Sun. “Adaptive Aggregation Networks for Class-Incremental Learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 2544–2553.

[97] Lu Yu et al. “Semantic Drift Compensation for Class-Incremental Learning”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 6980–6989.

[98] Zheda Mai et al. “Supervised Contrastive Replay: Revisiting the Nearest Class Mean Classifier in Online Class-Incremental Continual Learning”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2021, pp. 3584–3594.

[99] Nikhil Mehta et al. “Continual Learning using a Bayesian Nonparametric Dictionary of Weight Factors”. In: *AISTATS*. 2021.

[100] Dushyant Rao et al. “Continual Unsupervised Representation Learning”. In: *CoRR* abs/1910.14481 (2019).

[101] Andrés R. Masegosa et al. “Analyzing concept drift: A case study in the financial sector”. In: *Intell. Data Anal.* 24.3 (2020), pp. 665–688.

[102] Gustavo H. F. M. Oliveira, Leandro L. Minku, and Adriano L. I. Oliveira. “GMM-VRD: A Gaussian Mixture Model for Dealing With Virtual and Real Concept Drifts”. In: *International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary, July 14-19, 2019*. IEEE, 2019, pp. 1–8.

[103] João Gama and Gladys Castillo. “Learning with Local Drift Detection”. In: *Advanced Data Mining and Applications, Second International Conference, ADMA 2006, Xi'an, China, August 14-16, 2006, Proceedings*. Vol. 4093. Lecture Notes in Computer Science. Springer, 2006, pp. 42–55.

[104] Paulo Mauricio Gonçalves Jr and Roberto Souto Maior de Barros. “RCD: A recurring concept drift framework”. In: *Pattern Recognition Letters* 34.9 (2013), pp. 1018–1025. ISSN: 0167-8655.

[105] Bartosz Krawczyk and Alberto Cano. “Online ensemble learning with abstaining classifiers for drifting and noisy data streams”. In: *Applied Soft Computing* 68 (2018), pp. 677–692.

[106] Tegjyot Singh Sethi and Mehmed M. Kantardzic. “Handling adversarial concept drift in streaming data”. In: *Expert Systems with Applications* 97 (2018), pp. 18–40.

[107] Sergio Ramírez-Gallego et al. “Nearest Neighbor Classification for High-Speed Big Data Streams Using Spark”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.10 (2017), pp. 2727–2739.

[108] Albert Bifet and Ricard Gavaldà. “Learning from Time-Changing Data with Adaptive Windowing”. In: *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*. SIAM, 2007, pp. 443–448.

[109] Albert Bifet and Ricard Gavaldà. “Adaptive Learning from Evolving Data Streams”. In: *Advances in Intelligent Data Analysis VIII, 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31 - September 2, 2009. Proceedings*. Vol. 5772. Lecture Notes in Computer Science. Springer, 2009, pp. 249–260.

[110] Albert Bifet et al. “Extremely Fast Decision Tree Mining for Evolving Data Streams”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 2017, pp. 1733–1742.

[111] Roberto Souto Maior de Barros and Silas Garrido Teixeira de Carvalho Santos. “A large-scale comparison of concept drift detectors”. In: *Information Sciences* 451-452 (2018), pp. 348–370.

[112] Jogendra Nath Kundu et al. “Class-Incremental Domain Adaptation”. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. 2020, pp. 53–69.

[113] Doyen Sahoo et al. “Online Deep Learning: Learning Deep Neural Networks on the Fly”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 2660–2666.

[114] Jiangpeng He et al. “Incremental Learning in Online Scenario”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[115] Michal Woźniak, Manuel Graña, and Emilio Corchado. “A survey of multiple classifier systems as hybrid systems”. In: *Inf. Fusion* 16 (2014), pp. 3–17.

[116] Alberto Cano and Bartosz Krawczyk. “Kappa Updated Ensemble for drifting data stream mining”. In: *Machine Learning* 109.1 (2020), pp. 175–218.

[117] Albert Bifet, Geoffrey Holmes, and Bernhard Pfahringer. “Leveraging Bagging for Evolving Data Streams”. In: *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part I*. Vol. 6321. Lecture Notes in Computer Science. Springer, 2010, pp. 135–150.

[118] Nikunj C. Oza and Stuart J. Russell. “Online Bagging and Boosting”. In: *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics, AISTATS 2001, Key West, Florida, USA, January 4-7, 2001*. Ed. by Thomas S. Richardson and Tommi S. Jaakkola. Society for Artificial Intelligence and Statistics, 2001.

[119] Heitor Murilo Gomes et al. “Adaptive random forests for evolving data stream classification”. In: *Machine Learning* 106.9-10 (2017), pp. 1469–1495.

[120] Leandro L. Minku, Allan P. White, and Xin Yao. “The Impact of Diversity on Online Ensemble Learning in the Presence of Concept Drift”. In: *IEEE Transactions on Knowledge and Data Engineering* 22.5 (2010), pp. 730–742.

[121] J. Zico Kolter and Marcus A. Maloof. “Dynamic Weighted Majority: An Ensemble Method for Drifting Concepts”. In: *Journal of Machine Learning Research* 8 (2007), pp. 2755–2790.

[122] Haixun Wang et al. “Mining concept-drifting data streams using ensemble classifiers”. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*. ACM, 2003, pp. 226–235.

[123] Dariusz Brzezinski and Jerzy Stefanowski. “Reacting to Different Types of Concept Drift: The Accuracy Updated Ensemble Algorithm”. In: *IEEE Transactions on Neural Networks and Learning Systems* 25.1 (2014), pp. 81–94.

[124] Anjin Liu et al. “Accumulating regional density dissimilarity for concept drift detection in data streams”. In: *Pattern Recognition* 76 (2018), pp. 256 –272.

[125] Tegjyot Singh Sethi and Mehmed Kantardzic. “On the reliable detection of concept drift from streaming unlabeled data”. In: *Expert Systems with Applications* 82 (2017), pp. 77 –99. ISSN: 0957-4174.

[126] João Gama et al. “Learning with Drift Detection”. In: *Advances in Artificial Intelligence, Proceedings of SBIA 2004*. Vol. 3171. LNCS. Springer Verlag, 2004, pp. 286–295.

[127] Manuel Baena-García et al. “Early drift detection method”. In: *In Fourth International Workshop on Knowledge Discovery from Data Streams*. 2006.

[128] Roberto S.M. Barros et al. "RDDM: Reactive drift detection method". In: *Expert Systems with Applications* 90 (2017), pp. 344 –355.

[129] Danilo Rafael de Lima Cabral and Roberto Souto Maior de Barros. "Concept drift detection based on Fisher's Exact test". In: *Information Sciences* 442-443 (2018), pp. 220 –234.

[130] Roberto Souto Maior de Barros, Juan Isidro González Hidalgo, and Danilo Rafael de Lima Cabral. "Wilcoxon Rank Sum Test Drift Detector". In: *Neurocomputing* 275 (2018), pp. 1954 –1963.

[131] Kyosuke Nishida and Koichiro Yamauchi. "Detecting Concept Drift Using Statistical Testing". In: *Discovery Science*. Ed. by Vincent Corruble, Masayuki Takeda, and Einoshin Suzuki. 2007, pp. 264–269.

[132] David Tse Jung Huang et al. "Detecting Volatility Shift in Data Streams". In: *2014 IEEE International Conference on Data Mining*. 2014, pp. 863–868.

[133] I. Frías-Blanco et al. "Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds". In: *IEEE Transactions on Knowledge and Data Engineering* 27.3 (2015), pp. 810–823.

[134] Ali Pesaranghader and Herna L. Viktor. "Fast Hoeffding Drift Detection Method for Evolving Data Streams". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Paolo Frasconi et al. 2016, pp. 96–111.

[135] Andrzej Lapinski et al. "An Empirical Insight Into Concept Drift Detectors Ensemble Strategies". In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. 2018, pp. 1–8.

[136] Ali Pesaranghader, Herna Viktor, and Eric Paquet. "Reservoir of diverse adaptive learners and stacking fast Hoeffding drift detection methods for evolving data streams". In: *Machine Learning* 107.11 (2018), pp. 1711–1743.

[137] Edwin David Lughofer et al. "Drift detection in data stream classification without fully labelled instances". In: *2015 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS)*. 2015, pp. 1–8.

[138] Bartosz Krawczyk, Bernhard Pfahringer, and Michal Woźniak. "Combining active learning with concept drift detection for data stream mining". In: *2018 IEEE International Conference on Big Data (Big Data)*. 2018, pp. 2239–2244.

[139] Piotr Sobolewski and Michał Woźniak. "Comparable Study of Statistical Tests for Virtual Concept Drift Detection". In: *Proceedings of the 8th International Conference on Computer Recognition Systems CORES 2013*. Ed. by Robert Burduk et al. 2013, pp. 329–337.

[140] Tamraparni Dasu et al. "An information-theoretic approach to detecting changes in multi-dimensional data streams". In: *In Proc. Symp. on the Interface of Statistics, Computing Science, and Applications*. 2006.

[141] Anjin Liu et al. "Regional Concept Drift Detection and Density Synchronized Drift Adaptation". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 2280–2286.

[142] Łukasz Korycki and Bartosz Krawczyk. *Adversarial Concept Drift Detection under Poisoning Attacks for Robust Data Stream Mining*. 2020.

[143] Shuo Wang, Leandro L. Minku, and Xin Yao. "A Systematic Study of Online Class Imbalance Learning With Concept Drift". In: *IEEE Transactions on Neural Networks and Learning Systems* 29.10 (2018), pp. 4802–4821.

[144] Gabriel Aguiar, Bartosz Krawczyk, and Alberto Cano. *A survey on learning from imbalanced data streams: taxonomy, challenges, empirical study, and reproducible experimental framework*. 2022.

[145] T. Ryan Hoens, Robi Polikar, and Nitesh V. Chawla. "Learning from streaming data with concept drift and imbalance: an overview". In: *Progress in Artificial Intelligence* 1.1 (2012), pp. 89–101.

[146] Bartosz Krawczyk. "Learning from imbalanced data: open challenges and future directions". In: *Progress in AI* 5.4 (2016), pp. 221–232.

[147] Adel Ghazikhani, Reza Monsefi, and Hadi Sadoghi Yazdi. "Recursive least square perceptron model for non-stationary and imbalanced data stream classification". In: *Evolving Systems* 4.2 (2013), pp. 119–131.

[148] Peilin Zhao and Steven C.H. Hoi. "Cost-sensitive Online Active Learning with Application to Malicious URL Detection". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. Chicago, Illinois, USA, 2013, pp. 919–927.

[149] Yang Lu, Yiu ming Cheung, and Yuan Yan Tang. "Dynamic Weighted Majority for Incremental Learning of Imbalanced Data Streams with Concept Drift". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 2393–2399.

[150] Bilal Mirza, Zhiping Lin, and Nan Liu. "Ensemble of subset online sequential extreme learning machine for class imbalance and concept drift". In: *Neurocomputing* 149 (2015). Advances in neural networks Advances in Extreme Learning Machines, pp. 316 –329.

[151] Hualong Yu et al. “Active Learning From Imbalanced Data: A Solution of Online Weighted Extreme Learning Machine”. In: *IEEE Transactions on Neural Networks and Learning Systems* 30.4 (2019), pp. 1088–1103.

[152] Łukasz Korycki and Bartosz Krawczyk. “Low-Dimensional Representation Learning from Imbalanced Data Streams”. In: *Advances in Knowledge Discovery and Data Mining*. 2021, pp. 629–641.

[153] Hang Zhang et al. “Resample-Based Ensemble Framework for Drifting Imbalanced Data Streams”. In: *IEEE Access* 7 (2019), pp. 65103–65115.

[154] Siqi Ren et al. “Selection-based resampling ensemble algorithm for nonstationary imbalanced stream data learning”. In: *Knowledge-Based Systems* 163 (2019), pp. 705 –722.

[155] Gregory Ditzler and Robi Polikar. “Incremental Learning of Concept Drift from Streaming Imbalanced Data”. In: *IEEE Transactions on Knowledge and Data Engineering* 25.10 (2013), pp. 2283–2301.

[156] Ryan Elwell and Robi Polikar. “Incremental Learning of Concept Drift in Nonstationary Environments”. In: *IEEE Transactions on Neural Networks* 22.10 (2011), pp. 1517–1531.

[157] Wing W. Y. Ng et al. “Cost-Sensitive Weighting and Imbalance-Reversed Bagging for Streaming Imbalanced and Concept Drifting in Electricity Pricing Classification”. In: *IEEE Transactions on Industrial Informatics* 15.3 (2019), pp. 1588–1597.

[158] Shuo Wang, Leandro L. Minku, and Xin Yao. “Resampling-Based Ensemble Methods for Online Class Imbalance Learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 27.5 (2015), pp. 1356–1368.

[159] Shuo Wang, Leandro L. Minku, and Xin Yao. “Dealing with Multiple Classes in Online Class Imbalance Learning”. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence.* IJCAI’16. New York, New York, USA, 2016, pp. 2118–2124.

[160] Alberto Cano and Bartosz Krawczyk. “ROSE: robust online self-adjusting ensemble for continual learning on imbalanced drifting data streams”. In: *Machine Learning* (2022).

[161] Łukasz Korycki and Bartosz Krawczyk. “Concept Drift Detection from Multi-Class Imbalanced Data Streams”. In: *2021 IEEE 37th International Conference on Data Engineering (ICDE).* 2021, pp. 1068–1079.

[162] Edwin Lughofer. “On-line active learning: A new paradigm to improve practical useability of data stream modeling methods”. In: *Inf. Sci.* 415 (2017), pp. 356–376.

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

[164] Pawel Ksieniewicz et al. “Data stream classification using active learned neural networks”. In: *Neurocomputing* 353 (2019), pp. 74–82.

[165] H. S. Seung, M. Opper, and H. Sompolinsky. “Query by Committee”. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory.* COLT ’92. Pittsburgh, Pennsylvania, USA, 1992, pp. 287–294.

[166] Bartosz Krawczyk and Michal Woźniak. “Online query by committee for active learning from drifting data streams”. In: *2017 International Joint Conference on Neural Networks (IJCNN).* 2017, pp. 2120–2127.

[167] Bartosz Krawczyk and Alberto Cano. "Adaptive Ensemble Active Learning for Drifting Data Stream Mining". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 2763–2771.

[168] Cheong Hee Park and Youngsoon Kang. "An active learning method for data streams with concept drift". In: *2016 IEEE International Conference on Big Data (Big Data)*. 2016, pp. 746–752.

[169] Kaito Fujii and Hisashi Kashima. "Budgeted stream-based active learning via adaptive submodular maximization". In: *Advances in Neural Information Processing Systems 29*. 2016, pp. 514–522.

[170] Saad Mohamad, Moamar Sayed-Mouchaweh, and Abdelhamid Bouchachia. "Active learning for classifying data streams with unknown number of classes". In: *Neural Networks* 98 (2018), pp. 1 –15.

[171] G. Ditzler and R. Polikar. "Semi-supervised learning in nonstationary environments". In: *The 2011 International Joint Conference on Neural Networks*. 2011, pp. 2741–2748.

[172] Giovanna Castellano and Anna Maria Fanelli. "Classification of Data Streams by Incremental Semi-supervised Fuzzy Clustering". In: *Fuzzy Logic and Soft Computing Applications*. Cham, 2017, pp. 185–194.

[173] Łukasz Korycki and Bartosz Krawczyk. "Combining Active Learning and Self-Labeling for Data Stream Mining". In: *Proceedings of the 10th International Conference on Computer Recognition Systems CORES 2017*. 2018, pp. 481–490.

[174] Tal Wagner et al. “Semi-Supervised Learning on Data Streams via Temporal Label Propagation”. In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. 2018, pp. 5095–5104.

[175] Ricardo Sousa and João Gama. “Co-training Semi-supervised Learning for Single-Target Regression in Data Streams Using AMRules”. In: *Foundations of Intelligent Systems*. 2017, pp. 499–508.

[176] Karl B. Dyer, Robert Capo, and Robi Polikar. “COMPOSE: A Semisupervised Learning Framework for Initially Labeled Nonstationary Streaming Data”. In: *IEEE Transactions on Neural Networks and Learning Systems* 25.1 (2014), pp. 12–26.

[177] Atsutoshi Kumagai and Tomoharu Iwata. “Learning Dynamics of Decision Boundaries without Additional Labeled Data”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom, 2018, 1627–1636.

[178] Łukasz Korycki and Bartosz Krawczyk. “Class-Incremental Experience Replay for Continual Learning under Concept Drift”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2021), pp. 3644–3653.

[179] Gido M. van de Ven and Andreas Savas Tolias. “Three scenarios for continual learning”. In: *arXiv* abs/1904.07734 (2019).

[180] Pietro Buzzega et al. “Dark Experience for General Continual Learning: a Strong, Simple Baseline”. In: *arXiv* abs/2004.07211 (2020).

[181] Volodymyr Melnykov and Ranjan Maitra. “Finite mixture models and model-based clustering”. In: *Statistics Surveys* 4 (2010), pp. 80 –116.

[182] Paulo Martins Engel and Milton Roberto Heinen. “Incremental Learning of Multivariate Gaussian Mixture Models”. In: *Advances in Artificial Intelligence – SBIA 2010*. Ed. by Antônio Carlos da Rocha Costa, Rosa Maria Vicari, and Flavio Tonidandel. 2010, pp. 82–91.

[183] Alexander Rainer Tassilo Gepperth and Benedikt Pfülb. “Gradient-Based Training of Gaussian Mixture Models for High-Dimensional Streaming Data”. In: *Neural Process. Lett.* 53 (2021), pp. 4331–4348.

[184] Ehsan Variani, Erik McDermott, and Georg Heigold. “A Gaussian Mixture Model layer jointly optimized with discriminative features within a Deep Neural Network architecture”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 4270–4274.

[185] Benedikt Pfülb and Alexander Rainer Tassilo Gepperth. “Overcoming Catastrophic Forgetting with Gaussian Mixture Replay”. In: *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), pp. 1–9.

[186] Ting Su and Jennifer G. Dy. “In search of deterministic methods for initializing K-means and Gaussian mixture clustering”. In: *Intell. Data Anal.* 11 (2007), pp. 319–338.

[187] Reshad Hosseini and Suvrit Sra. “Matrix Manifold Optimization for Gaussian Mixtures”. In: *Advances in Neural Information Processing Systems*. Vol. 28. 2015.

[188] Nicholas J. Higham. “Cholesky factorization”. English. In: *Wiley Interdisciplinary Reviews: Computational Statistics* 1.2 (Sept. 2009), pp. 251–254.

[189] David Rolnick et al. “Experience Replay for Continual Learning”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on*

Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. 2019, pp. 348–358.

[190] Massimiliano Patacchiola and Amos Storkey. *Self-Supervised Relational Reasoning for Representation Learning*. 2020.

[191] Saining Xie et al. "Aggregated Residual Transformations for Deep Neural Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 5987–5995.

[192] Minghao Zhou et al. *Diagnosing Batch Normalization in Class Incremental Learning*. 2022.

[193] Quang Hong Pham, Chenghao Liu, and Steven C. H. Hoi. "Continual Normalization: Rethinking Batch Normalization for Online Continual Learning". In: *arXiv* abs/2203.16102 (2022).

[194] 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*. KDD '00. Boston, Massachusetts, USA, 2000, 71–80.

[195] Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. "Handling Numeric Attributes in Hoeffding Trees". In: *Advances in Knowledge Discovery and Data Mining, 12th Pacific-Asia Conference, PAKDD 2008, Osaka, Japan, May 20-23, 2008 Proceedings*. Vol. 5012. Lecture Notes in Computer Science. Springer, 2008, pp. 296–307.

[196] Heitor Murilo Gomes, Jesse Read, and Albert Bifet. "Streaming Random Patches for Evolving Data Stream Classification". In: *2019 IEEE Interna-*

tional Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019. IEEE, 2019, pp. 240–249.

[197] Sattar Hashemi et al. “Adapted One-versus-All Decision Trees for Data Stream Classification”. In: *IEEE Trans. Knowl. Data Eng.* 21.5 (2009), pp. 624–637.

[198] Sebastian Raschka. “Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning”. In: *CoRR* abs/1811.12808 (2018).

[199] David A. Cieslak et al. “Hellinger distance decision trees are robust and skew-insensitive”. In: *Data Mining and Knowledge Discovery* 24 (2011), pp. 136–158.

[200] R. J. Lyon et al. “Hellinger Distance Trees for Imbalanced Streams”. In: *2014 22nd International Conference on Pattern Recognition* (2014), pp. 1969–1974.

[201] Bartosz Krawczyk and Przemyslaw Skryjomski. “Cost-Sensitive Perceptron Decision Trees for Imbalanced Drifting Data Streams”. In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings, Part II*. Vol. 10535. Lecture Notes in Computer Science. Springer, 2017, pp. 512–527.

[202] Joshua Plasse and Niall M. Adams. “Handling delayed labels in temporally evolving data streams”. In: *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*. IEEE Computer Society, 2016, pp. 2416–2424.

[203] Łukasz Korycki and Bartosz Krawczyk. *Mining Drifting Data Streams on a Budget: Combining Active Learning with Self-Labeling*. 2021.

[204] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. 1st. The MIT Press, 2010.

[205] Saad Mohamad, Abdelhamid Bouchachia, and Moamar Sayed Mouchaweh. "A Bi-Criteria Active Learning Algorithm for Dynamic Data Streams". In: *IEEE Trans. Neural Netw. Learning Syst.* 29.1 (2018), pp. 74–86.

[206] Jing Zhang, Xindong Wu, and Victor S. Sheng. "Learning from crowdsourced labeled data: a survey". In: *Artificial Intelligence Review* 46 (2016), pp. 543–576.

[207] Łukasz Korycki and Bartosz Krawczyk. "Active Learning with Abstaining Classifiers for Imbalanced Drifting Data Streams". In: *2019 IEEE International Conference on Big Data (Big Data)*. 2019, pp. 2334–2343.

[208] Ziqing Lu et al. "LocalDrop: A Hybrid Regularization for Deep Neural Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), 1–1. ISSN: 1939-3539.

[209] Yue Zhu, Kai Ming Ting, and Zhi-Hua Zhou. "New Class Adaptation Via Instance Generation in One-Pass Class Incremental Learning". In: *2017 IEEE International Conference on Data Mining, ICDM 2017, New Orleans, LA, USA, November 18-21, 2017*. 2017, pp. 1207–1212.

[210] Cyprien de Masson d'Autume et al. "Episodic Memory in Lifelong Language Learning". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. 2019, pp. 13143–13152.

[211] Bennett Eisenberg. "On the expectation of the maximum of IID geometric random variables". In: *Statistics & Probability Letters* 78.2 (2008), pp. 135 –143.

[212] Wassily Hoeffding. “Probability Inequalities for Sums of Bounded Random Variables”. In: *Journal of the American Statistical Association* 58.301 (1963), pp. 13–30.

[213] Bernard Lewis Welch. “The Generalization of ‘Student’s’ Problem when Several Different Population Variances are Involved”. In: *Biometrika* 34.1/2 (1947), pp. 28–35.

[214] Albert Bifet et al. “MOA: Massive Online Analysis”. In: *Journal of Machine Learning Research* 11 (2010), pp. 1601–1604.

[215] Silas Garrido Teixeira de Carvalho Santos et al. “Speeding Up Recovery from Concept Drifts”. In: *Machine Learning and Knowledge Discovery in Databases*. 2014, pp. 179–194.

[216] Albert Bifet et al. “Efficient Online Evaluation of Big Data Stream Classifiers”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’15. Sydney, NSW, Australia, 2015, 59–68.

[217] Bogdan Gulowaty and Pawel Ksieniewicz. “SMOTE Algorithm Variations in Balancing Data Streams”. In: *Intelligent Data Engineering and Automated Learning - IDEAL 2019 - 20th International Conference, Manchester, UK, November 14-16, 2019, Proceedings, Part II*. Springer, 2019, pp. 305–312.

[218] Yu Sun et al. “Online Ensemble Learning of Data Streams with Gradually Evolved Classes”. In: *IEEE Trans. Knowl. Data Eng.* 28.6 (2016), pp. 1532–1545.

[219] Shuo Wang, Leandro L. Minku, and Xin Yao. “A multi-objective ensemble method for online class imbalance learning”. In: *2014 International Joint Con-*

ference on Neural Networks, IJCNN 2014, Beijing, China, July 6-11, 2014. 2014, pp. 3311–3318.

[220] Shuya Ding et al. “Kernel based online learning for imbalance multiclass classification”. In: *Neurocomputing* 277 (2018), pp. 139–148.

[221] Nitesh V. Chawla et al. “SMOTE: Synthetic Minority Over-sampling Technique”. In: *J. Artif. Intell. Res.* 16 (2002), pp. 321–357.

[222] Paula Branco, Luís Torgo, and Rita P. Ribeiro. “Relevance-Based Evaluation Metrics for Multi-class Imbalanced Domains”. In: *Advances in Knowledge Discovery and Data Mining - 21st Pacific-Asia Conference, PAKDD 2017, Jeju, South Korea, May 23-26, 2017, Proceedings, Part I.* 2017, pp. 698–710.

[223] Nikunj C. Oza. “Online bagging and boosting”. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Waikoloa, Hawaii, USA, October 10-12, 2005.* IEEE, 2005, pp. 2340–2345.

[224] Amal Saadallah et al. “BRIGHT - Drift-Aware Demand Predictions for Taxi Networks (Extended Abstract)”. In: *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019.* IEEE, 2019, pp. 2145–2146.

[225] Savitha Ramasamy, Arulmurugan Ambikapathi, and Kanagasabai Rajaraman. “Online RBM: Growing Restricted Boltzmann Machine on the fly for unsupervised representation”. In: *Appl. Soft Comput.* 92 (2020), p. 106278.

[226] Yin Cui et al. “Class-Balanced Loss Based on Effective Number of Samples”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019.* Computer Vision Foundation / IEEE, 2019, pp. 9268–9277.

[227] Xiaohai Sun. “Assessing Nonlinear Granger Causality from Multivariate Time Series”. In: *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part II.* Vol. 5212. Lecture Notes in Computer Science. Springer, 2008, pp. 440–455.

[228] Chahira Mahjoub et al. “On the performance of temporal Granger causality measurements on time series: a comparative study”. In: *Signal Image Video Process.* 14.5 (2020), pp. 955–963.

[229] Daniel K. Antwi, Herna L. Viktor, and Nathalie Japkowicz. “The PerfSim Algorithm for Concept Drift Detection in Imbalanced Data”. In: *12th IEEE International Conference on Data Mining Workshops, ICDM Workshops, Brussels, Belgium, December 10, 2012.* IEEE Computer Society, 2012, pp. 619–628.

[230] Shuo Wang and Leandro L. Minku. “AUC Estimation and Concept Drift Detection for Imbalanced Data Streams with Multiple Classes”. In: *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020.* IEEE, 2020, pp. 1–8.

[231] Bruno Veloso, João Gama, and Benedita Malheiro. “Self Hyper-Parameter Tuning for Data Streams”. In: *Discovery Science - 21st International Conference, DS 2018, Limassol, Cyprus, October 29-31, 2018, Proceedings.* 2018, pp. 241–255.

[232] Łukasz Korycki and Bartosz Krawczyk. “Online Oversampling for Sparsely Labeled Imbalanced and Non-Stationary Data Streams”. In: *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020.* IEEE, 2020, pp. 1–8.

[233] Łukasz Korycki and Bartosz Krawczyk. *Instance exploitation for learning temporary concepts from sparsely labeled drifting data streams.* 2020.

[234] Alessio Benavoli et al. "Time for a Change: a Tutorial for Comparing Multiple Classifiers Through Bayesian Analysis". In: *Journal of Machine Learning Research* 18 (2017), 77:1–77:36.

[235] Anjin Liu et al. "Accumulating regional density dissimilarity for concept drift detection in data streams". In: *Pattern Recognition* 76 (2018), pp. 256 –272.

[236] Piotr Sobolewski and Michal Woźniak. "Concept Drift Detection and Model Selection with Simulated Recurrence and Ensembles of Statistical Detectors". In: *J. UCS* 19.4 (2013), pp. 462–483.

[237] J. David Destephen Lavaire et al. "Dimensional scalability of supervised and unsupervised concept drift detection: An empirical study". In: *2015 IEEE International Conference on Big Data (Big Data).* 2015, pp. 2212–2218.

[238] Denis Moreira dos Reis et al. "Fast Unsupervised Online Drift Detection Using Incremental Kolmogorov-Smirnov Test". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016.* 2016, pp. 1545–1554.

[239] Diego Marron et al. "Low-latency multi-threaded ensemble learning for dynamic big data streams". In: *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017.* 2017, pp. 223–232.

[240] Michal Woźniak. "Application of Combined Classifiers to Data Stream Classification". In: *Computer Information Systems and Industrial Management*

- *12th IFIP TC8 International Conference, CISIM 2013, Krakow, Poland, September 25-27, 2013. Proceedings*. 2013, pp. 13–23.

[241] Ludmila I. Kuncheva and Christopher J. Whitaker. "Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy". In: *Machine Learning* 51.2 (2003), pp. 181–207.

[242] Yu-Ri Lee and Hyoung-Nam Kim. "A data partitioning method for increasing ensemble diversity of an eSVM-based P300 speller". In: *Biomedical Signal Processing and Control* 39 (2018), pp. 53 –63.

[243] Bartosz Krawczyk, Michal Woźniak, and Bogus Cyganek. "Clustering-based Ensembles for One-class Classification". In: *Inf. Sci.* 264 (Apr. 2014), pp. 182–195. ISSN: 0020-0255.

[244] Leandro L. Minku and Xin Yao. "DDD: A New Ensemble Approach for Dealing with Concept Drift". In: *IEEE Transactions on Knowledge and Data Engineering* 24.4 (2012), pp. 619–633.

[245] Silas Garrido Teixeira de Carvalho Santos et al. "Speeding Up Recovery from Concept Drifts". In: *Proceedings of the 2014th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part III*. ECMLPKDD'14. Nancy, France, 2014, pp. 179–194.

[246] Dariusz Brzezinski and Jerzy Stefanowski. "Ensemble Diversity in Evolving Data Streams". In: *Discovery Science*. 2016, pp. 229–244.

[247] Edo Liberty, Ram Sriharsha, and Maxim Sviridenko. "An Algorithm for Online K-Means Clustering". In: *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. 2016, pp. 81–89.

[248] Silas Santos et al. “Speeding Up Recovery From Concept Drifts”. In: Sept. 2014, pp. 179–194.

[249] Shang-Tse Chen, Hsuan-Tien Lin, and Chi-Jen Lu. “An Online Boosting Algorithm with Theoretical Justifications”. In: *Proceedings of the 29th International Coference on International Conference on Machine Learning.* ICML’12. Edinburgh, Scotland, 2012, 1873–1880.

[250] Jakob N. Foerster et al. “Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017.* Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1146–1155.

[251] Tom Schaul et al. “Prioritized Experience Replay”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings.* Ed. by Yoshua Bengio and Yann LeCun. 2016.

[252] Mohammad Rostami, Soheil Kolouri, and Praveen K. Pilly. “Complementary Learning for Overcoming Catastrophic Forgetting Using Experience Replay”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019.* ijcai.org, 2019, pp. 3339–3345.

[253] Jérémie Sublime, Basarab Matei, and Pierre-Alexandre Murena. “Analysis of the influence of diversity in collaborative and multi-view clustering”. In: *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017.* IEEE, 2017, pp. 4126–4133.

[254] Stefan Lee et al. "Stochastic Multiple Choice Learning for Training Diverse Deep Ensembles". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016, pp. 2119–2127.

[255] Tyler L. Hayes, Nathan D. Cahill, and Christopher Kanan. "Memory Efficient Experience Replay for Streaming Learning". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 9769–9776.

[256] Haobin Shi et al. "A Sample Aggregation Approach to Experiences Replay of Dyna-Q Learning". In: *IEEE Access* 6 (2018), pp. 37173–37184.

[257] Mengmi Zhang et al. "Prototype Reminding for Continual Learning". In: *CoRR* abs/1905.09447 (2019).

[258] Peter Kontschieder et al. "Deep Neural Decision Forests". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1467–1475.

[259] Vincenzo Lomonaco and Davide Maltoni. "CORe50: a New Dataset and Benchmark for Continuous Object Recognition". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 17–26.