DISSERTATION

MEMORY MODELS FOR INCREMENTAL LEARNING ARCHITECTURES

VIKTOR LOSING

Bielefeld University,
Machine Learning Research Group

SUPERVISED BY
PROF. DR. BARBARA HAMMER,
PROF. DR. HEIKO WERSING

JULY 16, 2019

The difference between the almost right word and the right word is really a large matter. 'tis the difference between the lightning bug and the lightning.

— MARK TWAIN

ABSTRACT

Technological advancement leads constantly to an exponential growth of generated data in basically every domain, drastically increasing the burden of data storage and maintenance. Most of the data is instantaneously extracted and available in form of endless streams that contain the most current information. Machine learning methods constitute one fundamental way of processing such data in an automatic way, as they generate models that capture the processes behind the data. They are omnipresent in our everyday life as their applications include personalized advertising, recommendations, fraud detection, surveillance, credit ratings, high-speed trading and smart-home devices. Thereby, batch learning, denoting the offline construction of a static model based on large datasets, is the predominant scheme. However, it is increasingly unfit to deal with the accumulating masses of data in given time and in particularly its static nature cannot handle changing patterns. In contrast, incremental learning constitutes one attractive alternative that is a very natural fit for the current demands. Its dynamic adaptation allows continuous processing of data streams, without the necessity to store all data from the past, and results in always up-to-date models, even able to perform in non-stationary environments. In this thesis, we will tackle crucial research questions in the domain of incremental learning by contributing new algorithms or significantly extending existing ones. Thereby, we consider stationary and non-stationary environments and present multiple real-world applications that showcase merits of the methods as well as their versatility. The main contributions are the following:

- One novel approach that addresses the question of how to extend a model for prototype-based algorithms based on cost minimization.
- We propose local split-time prediction for incremental decision trees to mitigate the trade-off between adaptation speed versus model complexity and run time.
- An extensive survey of the strengths and weaknesses of state-of-the-art methods that provides guidance for choosing a suitable algorithm for a given task.
- One new approach to extract valuable information about the type of change in a dataset.
- We contribute a biologically inspired architecture, able to handle different types of drift using dedicated memories that are kept consistent.
- Application of the novel methods within three diverse real-world tasks, highlighting their robustness and versatility.
- Investigation of personalized online models in the context of two real-world applications.

ACKNOWLEDGEMENTS

I am thankful to my supervisors Barbara Hammer and Heiko Wersing who guided me throughout the time. Both showed a lot of understanding and patience even in some critical moments. The plenty discussions we had kept me motivated and largely shaped the outcomes of this thesis.

Most of the work has been conducted at the Honda Research Institute Europe (HRI-EU) in Offenbach and the Machine Learning group of the Bielefeld University. Thanks to all colleagues there for the pleasant working atmosphere and the memorable social events as the numerous dinners, beach volleyball games, guitar- and SingStar sessions, and boulder evenings. In particularly, I am thankful to my office buddies Benjamin Metka, Dennis Orth and Christian Limberg for the fun times and countless laughs, but also various discussions that influenced this work. They also did most of the proof-reading on a very short notice.

Many thanks to the Honda-Research Institute Europe for the generous funding of this PhD project that gave me the great opportunity to present at different international conferences around the world. Thanks also to Honda-Research Institute Japan, in particular to Taizo Yoshikawa for the fruitful collaboration during the the PHUME project and the nice dinners in Offenbach and Tokyo. Furthermore, I want to thank Albert Bifet and his group for the joint-research we were able to do in Paris.

Finally, I am grateful to my family for the unconditional love, support and encouragement I received all my life.

CONTENTS

1 INTRODUCTION

RAPID technological progress increased mankind's capacity to store and compute information (Hilbert & López, 2011) and enabled the nowadays established practice to collect and store all conceivable information in digital form. The preserved information accumulates to enormous amounts and its growth rate increases daily. Every day Google receives 3.5 billion search queries; YouTube currently has 300 hours of video being uploaded every minute, and it is projected to grow up to 1,700 hours per minute by 2025; nearly 2 billion active users of Facebook share 4.5 billion pieces of content; Amazon sells about 13 million items world wide. These pioneering companies demonstrated that information can be the central pillar of a multi-billion dollar business. Even small firms adopted this approach and now digitize every transaction they are involved in, to boost their turnovers.

The type of acquired information ranges from personal information such as age, gender, place of residence over raw transactional data such as the purchase history to very specific data such as individual clicking behavior or average browsing session time. Collected data is mostly used to provide personalized recommendations, advertising own products to directly increase the turnovers, as done by Amazon for example: An estimated 35% of its 107 billion dollar net sales are attributed to its recommendation engine; or to place personalized advertisement for external products within own services as done by Google: 96% of its 75 billion dollar revenues are based on advertisement.

Data collection is omnipresent in science as well: Astronomical observatories, earth sensing satellites and climate observation networks generate terabytes of data on a daily basis. The Australian Square Kilometer Array Pathfinder (ASKAP) project acquires 7.5 terabytes/second of sample image data and is expected to increase to 750 terabytes/second by 2025 (Spencer, 2013). The field of genomics is predicted to be the biggest challenge in Big Data in the future (Stephens et al., 2015). Meanwhile, the rate at which data arises rapidly increases further - 90% of all the data in the world has been generated over the last two years. These masses of data are processed to gain further insights in various fields of research such as physics, astronomy, and medicine, enabling future breaktroughs and shape the development of mankind.

Data is also gathered by mobile devices. Since years, smartphones, smartwatches and fitness bands have been continuously tracking their user on basis of call and message logs, GPS positions, heart rates and acceleration data. In addition, more and more everyday appliances started to collect and exchange data as well, which is mainly driven by the Internet of Things (IOT) technology (Al-Fuqaha, Guizani, Mohammadi, Aledhari, & Ayyash, 2015). Its idea is to provide all-time-connected systems in smart-home or smart-city environments (Zanella, Bui, Castellani, Vangelista, & Zorzi, 2014). For example, smart speakers

such as Amazon's Alexa, but also smart TVs, continuously analyze audio waves for voice commands and simultaneously enrich their user profiles. Smart fridges monitor contained products to generate shopping lists or to order them directly online. Food monitoring also allows general health recommendations as well as personalized assistance for people on specific diets.

Thereby, the wide-spread strategy seems to be the more data is collected the better can customers be targeted and the more money can be earned. Obviously, such a strategy is highly questionable in regard to privacy issues. However, it cannot be denied that it also offers some benefits. As a company knows its customer better and better, it is able to provide superior services that satisfy highly individual demands in an efficient and convenient way. Even the society as a whole benefits: Extensive data collection is the foundation of smart city concepts that leverage data to improve various public services such as transport and parking, lighting, surveillance/maintenance of public areas, and garbage collection (Zanella et al., 2014).

The majority of the data is available in the form of data streams, i.e. new data is continuously generated. Efficient real-time processing and analysis is becoming extremely important not only as a way to reduce the burden of data storage, but is also crucial for various tasks that require a quick reaction to current situations. One exemplary application are warning systems for natural disasters such as earthquakes or typhoons, where an immediate reaction to corresponding sensor signals may save many lives (Alexander, 2017).

Additionally, a rapid adaptation of the models to current trends or important events can drastically increase the effectiveness. For example, dynamic control of the traffic flow by traffic-management systems can drastically reduce congestion (Nellore & Hancke, 2016). Control can occur on basis of adjusting traffic-light cycles and speed limits or may even imply to dynamically release additional lanes for certain directions. However, an accurate assessment of the current traffic situation is mandatory for such measures and ideally the position of each traffic participant is known at any time. Tracking of participants can be based on different information cues such as GPS or video streams of surveillance cameras that provide up-to-date signals. Real-time processing of information obtained from these sources allow to mitigate effects of suddenly occurring events such as accidents in a reactive way. However, in cases of foreseeable disruptions such as construction sides or public events a quick adaptation of the underlying models enables even a preventive reaction, which can avoid negative impacts altogether.

Machine learning methods are employed to mine collected data for relevant information and / or to predict future developments by generated models. In this context, batch / offline learning is still the predominantly applied paradigm, where a model is tediously constructed from scratch on basis of very large datasets. However, such a scheme does not meet the requirements to handle the enormous masses in the given time, leading to more and more unprocessed data. Furthermore, it is not able to deal with trends and changes over time,

as the temporal order of the data is not considered.

There are also scenarios where the data is initially not available, but is collected little by little during the application. One example is personalized learning, i.e. an adaptation to individual users. Obviously, the collection can only be done after the user started to use the product/ service. Ideally, systems noticeably adapt to user's preferences and idiosyncrasies on basis of only few interactions on the fly. Even though batch learning can be "bend" to such a setting by iterative retraining it is burdensome and inefficient, since the fundamental principles of batch learning do not apply here.

Overcoming this state of affair requires a paradigm shift to incremental learning from streams of information, allowing to discard already processed information. It is the natural way of learning as done by humans and animals (Jarvis, 2012). For example the learning of a new skill by humans develops gradually with increasing experience and is often categorized in four different stages from *unconscious incompetence*, the individual does not understand how to do something and does not recognize the deficit, to *unconscious competence*, the individual has had so much practice that it has become "second nature" (Dreyfus & Dreyfus, 1980). In particular, we learn and develop the skills during the task as done by incremental models which continuously incorporate information into their model, enabling up-to-date models. They are also able to process infinite streams instance-by-instance, making an permanent data storage unnecessary. At the same time, a minimal time and space complexity is provided, which is particularly attractive for Big Data tasks (Chen, Mao, & Liu, 2014).

Incremental algorithms enable devices to adapt to individual habits, opinions and environments, since learning is not restricted to the production phase. Resulting, highly personalized models are appealing in various applications, e.g. smart-home products, politics or advertisement (Tseng & Piller, 2011; Yang & Newman, 2013; Carolis, Ferilli, & Redavid, 2015; Bennett, 2012; Aguirre, Mahr, Grewal, de Ruyter, & Wetzels, 2015). Flexible adaptation of technical systems to user habits and preferences is very important for their acceptance, as noted for the learning Nest thermostat in (Yang & Newman, 2013), for example. Also other fully autonomous devices such as robotic vacuum cleaners or lawn mowers largely benefit from learning their application environment and keeping it up-to-date (Forlizzi & DiSalvo, 2006). Here, the main challenge is not large-scale processing, but rather continuous and efficient learning from few data. Incremental learning enables online adaptation which can be directly performed on the device independent from any mobile connection and cloud services, which may not be available in certain environments. Additionally, learning on the device is an elegant way to ensure privacy, as customers may not be willing to share data of their daily life.

As already indicated, learning from data streams often implies learning within changing environments. The capability to deal with change is essential, considering the fact that the world is constantly evolving. Algorithms trying to capture developing processes clearly need to continuously adapt in order to track changes. One example

are financial markets, which are heavily influenced by economical or political news, and even general public mood (Bikhchandani & Sharma, 2000). The progress of time is for humans inseparably linked with aging and new experiences that alter the personality, habits and preferences. Most humans live through typical life stages, which are associated with certain behavioral patterns, amount of available financial resources, or free time that all greatly influence daily choices (Helsper, 2010; Higgins, Duxbury, & Lee, 1994). Furthermore, the process of aging itself has its effect, for example human taste perception continuously deteriorate, influencing the dietary preferences (Mojet, Christ-Hazelhof, & Heidema, 2001).

Changing environments imply that old knowledge may become obsolete and even wrong, contradicting current beliefs. One common phenomena are overexposure effects. For example in fashion cycles, it is quite common that a high popularity of temporary styles or colors is often followed by an aversion of them (Acerbi, Ghirlanda, & Enquist, 2012). The same applies also for many other things such as popular songs or movies. Another example are temporary consumer demands, where interest vanishes after demands have been satisfied, or which may shift with different seasons (Ma, Li, Ding, & Orlowska, 2007; Feng & Gallego, 1995). Batch methods are by design unfit as they do not have any concept of time and consequently no mechanism to resolve conflicting information. In particular, they lack the ability to forget, which is mandatory to remove former facts / believes which become outdated and are in conflict with the current state. In contrast, incremental learning algorithms are dynamically evolving over time and thus are able to track changing processes. In particular, they can adapt their memory structure, enabling an explicit incorporation or deletion of information.

In this thesis, new algorithms are contributed to overcome major challenges of incremental learning from data streams. In particular, the work focuses on supervised classification, one specific type of incremental learning. However, incremental learning can also be applied in different settings, e.g. scenarios where the supervised signal is continuous (regression), or not available (unsupervised) (Gu, Sheng, Tay, Romano, & Li, 2015; Charikar, Chekuri, Feder, & Motwani, 2004).

A great importance is placed on real-world applications, as the merit of the methods is presented in different scenarios. More precisely, the overarching goal of this thesis is to improve incremental learning algorithms in key aspects, making them more viable for real-world applications. Key challenges are tackled by completely new developed algorithms or by specific extensions of existing ones. Thereby, we want to contribute to the understanding of such methods in terms of their strengths and weaknesses and showcase different application scenarios in which they provide a clear benefit.

The central challenge of incremental learning algorithms is to handle the stability-plasticity dilemma in an optimal way. The dilemma describes two desired, but contradicting properties. On the one hand, a high stability is necessary to preserve information as long as possible,

whereas on the other hand, a high plasticity is crucial for quick learning of new patterns (Grossberg, 1988). The corresponding research question is how to **adapt to subsequent novel data**, whereby the adaptation concerns model parameters, model meta-parameters, as well as model complexity in case of methods which have the ability to grow. In this thesis, the challenge is tackled from two different perspectives.

First, we shed light from the perspective of stationary environments, assuming data samples are independently drawn from one static distribution. Here, we address the challenge of **incremental adaptation of model parameters and meta-parameters** in intuitive model architectures originally proposed for the batch scenario. In the second, we drop the assumption of data being identical and independently distributed (i.i.d.), since it is often unrealistic in case of learning from data streams. Instead, instances are often temporally correlated as they describe real-world processes. Furthermore, such processes and therefore also the underlying distribution evolve over time in different ways. Changes of the underlying distribution are named *concept drift*. Hence, incremental learning is drastically more challenging. Changes of the environments can be completely different, therefore, they are categorized in various types. Different types are handled by different mechanisms in an optimal way. Corresponding major challenges are how to **design an architecture which is able to deal with different types of drift**, flexible enough to adapt to diverse tasks on the fly and what are ways to decide **whether past knowledge is still valid**.

1.1 CONTRIBUTIONS AND MANUSCRIPT STRUCTURE

After having introduced the general topic as well as the overarching research questions, we will address the latter by proposing novel algorithms. In the following, the scientific contributions of this work are summarized. Concretely, we briefly line out the content of each chapter, specify single research questions and refer to corresponding contributions.

Incremental Learning in Stationary Environments The incremental learning problem is defined in Chapter 3. It elaborates on the applicability of incremental learning algorithms in real-world problems. In particular, one major obstacle is the acquisition of ground truth information in an online way. Three different possibilities of ground truth extraction are proposed and discussed in the context of potential applications. The focus within this chapter is incremental learning in stationary environments, where we contribute in three ways.

- Some incremental algorithms are able to match their model complexity according to the task at hand by expanding their model on the fly. However, one key question is how to **expand the model in an optimal way**, as measured by a cost function such as the classification error. Section 3.2 gives an answer with respect to prototype-based algorithms. Concretely, we propose the

Incremental Learning Vector Quantization (ILVQ), an architecture which dynamically inserts new prototypes based on cost optimization.

- Growing algorithms have to dynamically decide **when to add new parameters** and thereby consider the related trade-off between fast model adaptation versus a high model complexity. In Section 3.3, we reduce the burden for incremental decision tree induction by predicting the split-time of nodes based on local information.

- Numerous incremental learning algorithms have been published. Naturally, the question concerning the **strengths and weaknesses of state-of-the-art algorithms** arises. One related problem in practice is to **choose a suitable learning algorithm for a given task at hand**. In Section 3.4, insight is provided in form of a practical survey which identifies key characteristics of incremental learning problems and accordingly evaluates state-of-the-art methods in an empirical way.

Incremental Learning in Non-Stationary Environments Changing environments and their impact on incremental learning are treated in Chapter 4. Concretely, the common term *concept drift* is formally introduced as well as its major forms. In relation to incremental learning under concept drift the following contributions are made.

- Assessing the drift characteristics in a given stream of data provides insights in the task at hand and facilitates the choice of suitable methods. However, it is yet unclear **how to extract information about the drift specifics from data**. In Section 4.3, we fill this gap by a novel approach, able to determine the drift type and the corresponding degree within a chunk of data.

- Non-stationary environments are very challenging because the possibly occurring type of changes are fundamentally different. Most approaches are able to handle some types of drift, thereby having crucial weak spots for other. Hence, one important research question is how to **design an architecture able to deal with different drift types**. An answer is given in form of the proposed Self-Adjusting Memory (SAM) architecture, described in Section 4.4.

- Ensemble methods are known to have a higher performance and robustness in comparison to single classifiers. They are a popular technique in non-stationary environments, allowing a quick adaptation by the addition of learners as well as their deletion. In Section 4.5, SAM is further improved by integration within a bagging ensemble to **increase its diversity** even further.

Real-World Applications One major focus of this thesis is the real-world application of incremental methods, where three main objectives

are targeted. First, we present viable application scenarios where incremental learning provides a concrete gain. Second, novel methods are transferred into real-world tasks to demonstrate their applicability and competitiveness in comparison to state-of-the-art approaches. Furthermore, we specifically investigate benefits of personalized online learning in concrete settings. Concretely, we compare models which are exclusively adapting to a personalized signal in an incremental way against those trained for multiple users in batch mode. Concretely, the following applications are presented in Chapter 5.

- Incremental learning based on visual input is especially challenging in an outdoor environment because changing lighting conditions heavily affect the representation, naturally generating concept drift. In Section 5.1, the proposed ILVQ architecture is applied within an interactive learning scenario in the domain of outdoor object recognition on a mobile robot.
- Most traffic accidents occur at intersections (U.S. Department of Transportation, 2007). In this context, driver-behavior prediction can be used to issue warnings or prevent risky actions for the purpose of a higher overall safety. In this context, we analyze benefits of personalized online models in Section 5.2.
- Personalized online models are also investigated in Section 5.3. Here, the task is the real-time classification of human motions based, which can be used to control motion-supportive devices to assist humans with their intended motion. Recorded motions are based on multiple Inertial Measurement Units (IMU) that are covering the whole body.

1.2 PUBLICATIONS IN THE CONTEXT OF THIS THESIS

Most of the work has been conducted at the Honda Research Institute Europe (HRI-EU) in Offenbach and the Machine Learning group of the Bielefeld University. In addition, I had the opportunity to visit the research group Data, Intelligence and Graphs (DIG) of Albert Bifet at the University Telecom ParisTech in Paris for one month. The Personalized Human Estimation (PHUME) project enabled me to participate in an international collaboration between HRI-EU and the Honda Research Institute Japan (HRI-JP) for several months. Thereby, I particularly collaborated with Taizo Yoshikawa from HRI-JP and was able to visit him and his co-workers at HRI-JP in Tokyo.

During my PhD studies I had the opportunity to present large parts of the work to an international audience in the form of journal articles and conference publications as follows:

Journal Articles

- Losing, V., Hammer, B., & Wersing, H. (2018b). Incremental online learning: A review and comparison of state of the art algorithms. *Neurocomputing*, *275*.

- Losing, V., Hammer, B., & Wersing, H. (2018c). Tackling heterogeneous concept drift with the self-adjusting memory (sam). *Knowledge and Information Systems*, *54*(1), 171–201.

Conference Articles

- Losing, V., Hammer, B., & Wersing, H. (2015). Interactive online learning for obstacle classification on a mobile robot. In *2015 international joint conference on neural networks (ijcnn)* (pp. 1–8). IEEE.
- Losing, V., Hammer, B., & Wersing, H. (2016a). Choosing the best algorithm for an incremental on-line learning task. In *2016 24th european symposium on artificial neural networks (esann)*.
- Losing, V., Hammer, B., & Wersing, H. (2016b). Dedicated memory models for continual learning in the presence of concept drift. In *Advances in neural information processing systems (nips) 29, continual learning workshop*.
- Losing, V., Hammer, B., & Wersing, H. (2016c). Knn classifier with self adjusting memory for heterogeneous concept drift. In *2016 ieee 16th international conference on data mining (icdm)* (pp. 291–300).
- Losing, V., Hammer, B., & Wersing, H. (2017a). Personalized maneuver prediction at intersections. In *2017 ieee 20th international conference on intelligent transportation systems (itsc)* (pp. 1–6).
- Losing, V., Hammer, B., & Wersing, H. (2017b). Self-adjusting memory: How to deal with diverse drift types. In *Proceedings of the twenty-sixth international joint conference on artificial intelligence, IJCAI-17* (pp. 4899–4903).
- Losing, V., Hammer, B., & Wersing, H. (2018a). Enhancing very fast decision trees with local split-time predictions. In *2018 ieee 16th international conference on data mining (icdm)*.
- Losing, V., Hammer, B., & Wersing, H. (2019). Personalized online learning of whole body motions using multiple inertial measurement units. In *Ieee international conference on robotics and automation (icra) 2019*.
- Losing, V., Hammer, B., Wersing, H., & Bifet, A. (2019). Tackling concept drift with a diverse self-adjusting memory ensemble. In *Submitted to international conference on data engineering (icde) 2019*.

2 HISTORICAL BACKGROUND

This chapter provides a short history of machine learning from the perspective of incremental learning. It describes how incremental learning emerged as an own discipline and lines out the development that lead to its focus on data streams within non-stationary environments.

2.1 BIOLOGICALLY INSPIRED LEARNING

Initial algorithms in machine learning research were heavily influenced by insights from the fields of Neurobiology. Since humans constitute the most impressive incremental learners based on their sensory input and experiences, research was focused on mirroring these learning processes. In particularly, algorithms were by design incremental and had various biological analogies. One prominent example are artificial Neural Networks (ANN), which constitute a simplified model of biological neural networks as they consist of interconnected neurons, where the connections are weighted. Similarly, neurons in the brain are connected to nearby neurons via different synaptic strengths. The early stage of machine learning was dominated by different types of ANNs. Rosenblatt (1958) invented the Perceptron algorithm which was able to perform binary classification. All input dimensions are connected with one single neuron and the algorithm learns the connection weights that represent a linear discrimination function. In other words, each input is multiplied with the connection weights and the algorithm outputs a one (the neuron fires) if the sum is above a certain threshold, otherwise a zero. The Perceptron learns on basis of mistakes, which also play a crucial role in natural learning. For instance, acethylochine is released in the brain to enhance learning in case of task failures (Hasselmo, 2006) and specific neurons are firing when people catch themselves during a mistake prompting a direct reaction in terms of corrected behavior (Fu et al., 2019).

The Perceptron algorithm was implemented on custom-built hardware and performed image recognition based on 400 photocells. The success of the algorithm able to learn by itself on basis of examples, led quickly to enthusiastic expectations as was highlighted by a report of the New York Times (1958):
"*The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. [...] Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language.*"
Even Rosenblatt himself got carried away in a related interview:
"*Dr. Frank Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers*".

Such promises were quickly exposed by Minsky and Papert (1972) as they showed the limitation of the Perceptron for certain problems, e.g it could not represent the XOR function. In particular, they demonstrated that ANNs with multiple linear layers that are sequentially connected were necessary to solve such problems. However, the Perceptron learning rule was insufficient as it could only train the weights of the output layer. Minskys criticism was interpreted as general flaw of neural networks that are ultimately limited in their capability and initiated the so-called first Artificial Intelligence (AI) winter, a period of disillusionment characterized by a drop in research funding and publications.

Eventually, the Backpropagation algorithm provided a way of training a multi-layer network and revived machine learning research. Errors are propagated backwards from the output layer and distributed on neurons of the hidden layers. In particular, the learning of ANNS was formalized as a cost function minimization for which Backpropagation delivered the necessary gradients. The success of the algorithm commenced a movement away from biologically inspired heuristics towards a precisely framed mathematical foundation. Backpropagation was derived by multiple researchers but Paul Werbos was the first to propose it for ANNs (Werbos, 1974). However, his contribution remained largely unnoticed and it took until the paper of Rumelhart, Hinton, and Williams (1985) that Backpropagation became widely known. The authors presented the key idea in a a concise way and specifically addressed the problems mentioned by Minsky.

2.2 PSEUDO-INCREMENTAL LEARNING

Early ANNs were incremental algorithms as their weights were consecutively adjusted to presented examples. However, the main priority of machine learning was mostly centered of reaching or even surpassing human-level capability. Therefore, the question concerning the representational power of ANNs, which was basically framed by Minsky's criticism, was a central interest, whereas aspects like online learning or time- and space efficiency were rather secondary. As Hornik, Stinchcombe, and White (1989) proved that ANNs with only one hidden layers are universal approximators that could be trained with Backpropagation, cost-function minimization became the standard way of learning.

Numerical instabilities such as exploding or vanishing gradients made Backpropagation impractical for deep architectures and put an end to incremental learning as the process either converged slowly or not at all (Bengio, Simard, & Frasconi, 1994). Consequently, techniques as Conjugate Gradient or Newtons method (Reed & Marks, 1998) were used to mitigate this issues, leading to batch-wise training as they require a large number of samples to be accurate. Moreover, ANNs were trained on basis of multiple iterations over large datasets, contradicting another principle of incremental learning.

2.3 EARLY INCREMENTAL/ONLINE LEARNING

As most research centered around ANNs and batch-wise learning via Backpropagation, a small community focused on efficient incremental learning, where the number of stored examples was strictly limited (often only one example was allowed) and the dataset was processed only once. The convergence analysis of the Perceptron by Novikoff (1962) was the first step but it took 30 additional years until the field of online learning was finally established by Littlestone, Kivinen, Wardon, Cesa-Bianchi and others (Littlestone, 1988; Cesa-Bianchi et al., 1997; Kivinen & Warmuth, 1997). Littlestone (1988) presented the WINNOW algorithm, a linear classifier which explicitly is evaluated in the online learning setting. The online learning community set a high priority on learning theory and researchers formally deduced upper-bounds for the number of mistakes an algorithm makes within certain tasks. Consequently, rather simple and mostly linear algorithms were analyzed, which delivered inferior classification performance compared to their non-linear batch counterparts for real-world tasks. Thus, the level of gained attention was comparatively limited.

Schlimmer and Fisher (1986) introduced Iterative Dichotomiser 4 (ID4), one of the first incremental non-linear models. It is an incremental decision tree and was derived from the successful batch algorithm ID3 (Quinlan, 1986). However, it did not produce the same trees in various cases and moreover was not able to learn certain problem classes. Such issues were fixed by UTGOFF (1988) with ID5R, a lossless incremental version of ID3, i.e. it generates the same tree as the batch version independent from the data order. ID4 was the beginning of a popular scheme in the research of incremental learning. It aimed for rephrasing accurate batch algorithms into an incremental prescription to get the best of both worlds.

Non-stationary environments were early considered within the Adaptive Resonance Theory (ART) introduced by Grossberg (1976a, 1976b). It is biologically inspired as it analyzes the mechanism in the brain that enable a stable learning even in rapidly changing environments. ART extracts a mathematical model that explicitly tries to imitate the brains extraordinary capability to balance between stability and plasticity on basis of a short- and long-term memory. Carpenter and Grossberg (1987a) developed various neural network architectures that partly implemented ART (Carpenter & Grossberg, 1987a, 1987b; Carpenter, Grossberg, & Reynolds, 1991).

The term concept drift can be traced back to Schlimmer and Granger (1986) who contributed the classification method STAGGER, which explicitly handles concept drift. STAGGER learned a weighted set of Boolean functions consisting of attribute-value pairs to describe concepts. Thereby, the combination of the rules could be based on conjunction, disjunction or a negation. STAGGER is able to follow changes in the concepts by adjusting the weights as well as forming new Boolean functions.

Widmer and Kubat (1996) provided an interesting view of concept drift, as they framed it as a lack of information, a hidden context that is missing in the data, which guides the change. They introduced the algorithm FLORA which was one of the first methods to use a sliding window in the context of non-stationary environments. The window is of fixed size and stores the most recent examples in a first-in-first-out (FIFO) data structure. FLORA stored sets of correct and incorrect concept descriptions, also based on Boolean functions. FLORA2 was introduced to adapt the window size dynamically by using multiple thresholds to increase and decrease the size based on the recent performance (Widmer & Kubat, 1992). FLORA3 was one of the first methods particularly tackling reoccurring concepts (Widmer & Kubat, 1993).

Even though the early concept drift methods were rather simple and applied only on toy examples, important foundations were established as the major types of drift were introduced and separately addressed by different approaches. Furthermore, important techniques to handle drift were contributed, for instance dynamic sliding windows are widely used today.

2.4 SUPPORT VECTOR MACHINE AND CONVEX OPTIMIZATION

Problems such as the vanishing and exploding gradient, local minima, and the difficulty of choosing a proper network architecture made the application of ANNs very challenging, in particular for deep architectures with many layers. Cortes and Vapnik (1995) introduced the Support Vector Machine (SVM), a large margin classifier which empowered with the kernel trick and soft margins achieved highly accurate results. The SVM is not prone to numerical issues, since it solves a convex optimization problem. As it furthermore determines its own structure in respect to the number of support vectors, researchers largely perceived this mathematically well-founded method as superior and abandoned ANNs. Their focus shifted to kernel methods and convex optimization, where convergence of the methods was guaranteed. Even Yann LeCun, a prominent advocate of ANNs, stated:

"*The optimal margin classifier has excellent accuracy, which is most remarkable, because unlike the other high performance classifiers, it does not include a priori knowledge about the problem. In fact, this classifier would do just as well if the image pixels were permuted with a fixed mapping.*" - LeCun et al. (1995).

The SVM was another step away from incremental learning as the algorithm was a purely batch algorithm. At this time, the main ambition of incremental learning research became to transfer successful batch methods to make them viable for very large datasets, where the batch algorithm could not be applied. Thereby, the aspiration was also to shift away from heuristics and either to provide lossless algorithms, or at least to proof similarity in the limit. Cauwenberghs and Poggio (2001) introduced the first lossless incremental SVM, (Oza,

2005) transferred the techniques of Bagging and Boosting (Breiman, 1996; Freund & Schapire, 1997) and Liang, Huang, Saratchandran, and Sundararajan (2006) introduced the incremental Extreme Learning Machine. In the same spirit, Domingos and Hulten (2000) proposed the Very Fast Decision Tree (VFDT), an incremental decision tree that provably construct trees that are similar to those of batch algorithms. The VFDT placed a high value on efficiency and included various techniques to speed up the training and to bound the memory demand. One reason for its popularity was an efficient implementation that was made public by the authors. Another important algorithm of that time was Learn++ proposed by Polikar, Upda, Upda, and Honavar (2001). It was largely inspired by Adaptive Boosting (Freund, Schapire, & Abe, 1999). Learn++ constructs a sequential ensemble of weak classifiers which are differently weighted for the classification. The weak classifiers consist of batch models, initially ANNs but later also SVMs, and are trained on disjunct chunks of data.

2.5 THE RISE OF TREE ENSEMBLES

The Random Forest (RF) was popularized by Breiman (2001). It is a Bagging (Breiman, 1996) ensemble of decision trees that uses randomization techniques to avoid overfitting. They perform well out-of-the box and are efficient even for large and high-dimensional datasets. Whereas, the SVM requires the tuning of the kernel with, has a high training complexity and thus scales poorly to large datasets. As it became clear that RFs also deliver similar performances (Fernández-Delgado, Cernadas, Barro, & Amorim, 2014) people started to apply them in different areas, which ended the dominance of the SVM. Saffari, Leistner, Santner, Godec, and Bischof (2009) adapted them to the incremental learning scheme and was able to achieve competitive results.

The technological progress lead to more and more generated data in a myriad of domains. In particular, data became real-time available in forms of streams, generated from sensor networks, mobiles, smart devices asf. It initiated a diversification of incremental learning away from batch learning, in terms of a focus on highly efficient models that continuously learn from data streams. Researchers aimed for linear or even logarithmic time- and space complexities and approximations were accepted as long as the approximative error could be bounded. The VFDT was in line with this scheme and numerous contributions focused on its improvement. In fact, the VFDT can be seen as the starting point of the focus to learning from massive data streams (Domingos & Hulten, 2000).

As data streams are capturing real-world problems they are naturally non-stationary, the assumption of identical and independent distributed data was mostly abandoned which further increased the diversification from batch learning. Hulten, Spencer, and Domingos (2001) published an adaptation of VFDT to non-stationary environments. Street and Kim (2001) were the first to use ensembles within

non-stationary environments within their Streaming Ensemble Algorithm (SEA). Gama, Medas, Castillo, and Rodrigues (2004) introduced one of the first drift detection methods with a formal foundation. The ideas of drift Detectors were later combined with ensembles to trigger the replacement of outdated learners in case of drift (Bifet, Holmes, & Pfahringer, 2010; Elwell & Polikar, 2011).

An important contribution was done by Bifet, Holmes, Kirkby, and Pfahringer (2010) who introduced Massive Online Analysis (MOA), an open source learning framework which is widely used today. It contains numerous incremental algorithms and benchmark datasets, facilitating the comparison of state-of-the-art methods.

2.6 CURRENT STATE

The unprecedented performance of Deep Learning removed any doubts about ANNs and lead to their wide renaissance. The success story was based on large datasets, drastically increased computational capabilities via Graphical Processing Units (GPU) and algorithmic improvements (LeCun, Bengio, & Hinton, 2015). It lead to many breakthroughs and is omnipresent in many commercial applications such as object-, face-, and speech recognition, recommender systems and autonomous driving.

Incremental learning remains a secondary aspect, as the main focus remains on what is generally possible, and how to move towards general intelligence. The field of reeinforcement learning is an interesting example. Even though online learning is its core foundation it is widely applied in batch fashion, trained on large clusters for weeks as the goal remains to showcase the "final performance". Even though remarkable milestones were recently reached in different games (Mnih et al., 2015; Silver et al., 2016), the incremental learning capabilities are still limited as millions of learning iterations are required. Obviously, more efficient learning is essential for further progress.

Relatively little is known of whether incremental supervised learning is used in commercial applications. Tasks based on non-stationary data streams that require real-time adaptation seem to be a viable area in future. Nonetheless, online adaptation is nowadays mostly handled on basis of repetitive batch learning from scratch. It remains an open question whether the constantly rising amount of generated data and the demand for less processing delay will lead to a change in paradigm. In general, a fusion of incremental and batch approaches seems a necessary step, as incremental methods are adaptive but less stable, requiring a fall back solution.

Currently, research of incremental supervised learning on data streams is mainly focused on developing more efficient and powerful methods. In regard to non-stationary environments, the goal is to design robust and efficient algorithms that perform well for various types of drift.

The huge success of Deep learning showcased that hierarchical architectures are able to extracted representations that enable a human-

like discriminative performance. However, as humans have further remarkable abilities, for instance rapid learning based on few examples, the transfer of concepts to different domains, and quick adaption to changing environments, they clearly indicate the great potential of incremental learning that is yet unmatched.

This thesis contributes at three different frontiers of incremental learning. First, novel approaches are proposed that improve the efficiency and performance of incremental learning. Second, a new architecture is developed that is able to handle different types of drift out-of-the-box on basis of consistent memories, a novel way to approach the stability plasticity dilemma. Lastly, it contributes to a wider application of incremental learning by an overview which assesses the qualities of current approaches and streamlines the choice of an algorithm for a given task. Moreover, a diverse set of own applications is presented, where the advantages of incremental learning are explicitly shown, including a higher classification performance in two different personalization tasks.

3 INCREMENTAL LEARNING

Summary This chapter characterizes incremental learning and elaborates on the different challenges posed by its learning paradigm in stationary environments. The important aspects of how and when to add model parameters are tackled for two different algorithms. The most common evaluation metrics to assess the classification performance are introduced and applied in an extensive survey, which empirically compares state-of-the-art methods and guides the choice of a suitable algorithm for a given task.

Source Code

- Python-based sources of the Incremental Learning Vector Quantization (ILVQ) with COst MinimizatiOn Sampling (COSMOS) (Losing, Hammer, & Wersing, 2015) are available at https://github.com/vlosing/ILVQ.
- Source code of the split-time prediction method One-Sided Minimum (OSM) (Losing, Hammer, & Wersing, 2018a) can be found at https://github.com/vlosing/splitTimePrediction.
- Links to all algorithms analyzed in the practice-oriented survey are provided at https://github.com/vlosing/algorithms.
- Links to all datasets used within this thesis can be found at https://github.com/vlosing/datasets.

Parts of this chapter are based on:

- Losing, V., Hammer, B., & Wersing, H. (2015). Interactive online learning for obstacle classification on a mobile robot. In *2015 international joint conference on neural networks (ijcnn)* (pp. 1–8). IEEE.
- Losing, V., Hammer, B., & Wersing, H. (2016a). Choosing the best algorithm for an incremental on-line learning task. In *2016 24th european symposium on artificial neural networks (esann)*.
- Losing, V., Hammer, B., & Wersing, H. (2018a). Enhancing very fast decision trees with local split-time predictions. In *2018 ieee 16th international conference on data mining (icdm)*.
- Losing, V., Hammer, B., & Wersing, H. (2018b). Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, *275*.

INCREMENTAL learning constitutes an attractive alternative to classical batch learning, offering various advantages which gain more and more practical relevance. It enables the processing of infinite datasets at a low time and space complexity, particularly interesting in the age of Big Data where humanity is generating more data then it currently can handle. The main difference from batch learning is the instance-by-instance processing scheme, which results in continuously evolving models, enabling lifelong-learning in a natural way. Another difference is that models have not to be pretrained with large training sets, which try to simulate the actual application environment on the basis of assumptions that are not always met. Instead, models adapt during the actual application and thus elegantly avoid a discrepancy between training and test data.

In this chapter, we will shed some light on learning paradigms which enable lifelong learning in the presence of streaming data, where we yet assume that data are emitted from an underlying stationary distribution of priorly unknown complexity. In this context, quite a few questions arise, the most prominent ones being the following:

- What are possible application scenarios where incremental learning provides a strong benefit? Which incremental learning strat-

egy is best suited for which tasks?

- How can a model incrementally adapt to subsequent novel data, whereby adaptation concerns model parameters, model meta-parameters, as well as model complexity? How can this be achieved with limited memory resources and in real time?

- How can the learning process be evaluated? What is the best possible convergence speed and how can we avoid spurious effects and overfitting to the current data points?

In the following, we will center around these questions and provide answers by investigating and designing incremental learning algorithms in typical model and real-world application scenarios. More specifically, incremental learning algorithms are formally defined in Section 3.1.1.

Section 3.1.2 elaborates on the challenges incremental algorithms face, thereby continuously contrasting them against batch methods. One practical challenge is to get the ground truth in an online way, which reduces the range of tasks incremental learning can be applied at. We discuss necessary conditions for real-world application and describe exemplary scenarios.

Incremental learning algorithms generate a temporal sequence of models, which opens various possibilities in terms of performance evaluation. There are two main metrics that allow not only to measure the generalization ability but also contain information about the learning speed. These are formally introduced and analyzed in terms of their advantages and drawbacks. Both metrics are repetitively applied within the experiments in Sections 3.2, 3.3, 3.4.

The instance-by-instance processing forces an adaptation based on a subset of the data. The question of how to optimally adapt the model-parameters under such circumstances is largely unsolved. Closely related is the stability-plasticity dilemma, which demands two contradicting model properties at the same time. We reflect on these issues in a detailed way.

Similar to the human brain which grows new neurons and connections during learning, various algorithms are able to add model parameters to match the complexity of the task. We discuss the related challenge of how to efficiently grow the model, by adding new model parameters, in respect to the learning performance. Aspects as the resulting processing time and memory consumption are considered as well. Another related research question is when to grow the model, since the a priori unknown amount of data forces algorithms to continuously balance their learning efficiency against the complexity of the model. Both challenges are separately targeted in terms of algorithmic contributions. The first (how to grow the model) is addressed by a new prototype placement strategy for the Incremental Learning Vector Quantization (ILVQ) in Section 3.2, whereas a split-time prediction, proposed for the Very Fast Decision Tree (VFDT) in Section 3.3, is

tackling the latter challenge (when to grow the model).

Taking into consideration that numerous incremental learning algorithms have been published, naturally the question arises regarding the strengths and weaknesses of state-of-the-art methods. A related problem in practice is to choose a suitable learning algorithm for a given task at hand. Unfortunately, there is a lack of research which could offer some guidance. Section 3.4 provides insights in form of an extensive survey, which empirically compares state-of-the-art methods in regard to major aspects. Apart from the classification performance, we also consider the model complexity and their development during learning, addressing questions as how much classification performance can be expected to be lost when simple models are applied instead of complex ones, or what is the training and evaluation complexity of a specific algorithm and how does it scale with an increasing amount of model parameters? Another treated criteria is the learning speed with associated questions like how much data is required until a reasonable performance or convergence can be expected and which methods are the most efficient ones? Tightly connected is the setting of hyperparameters. We analyze which methods are easy to apply in practice. Specifically, we tackle concerns such as the sensitivity of a method regarding different settings of these hyperparameters and whether they can be robustly estimated based on a small amount of data. Finally, methods are examined in terms of their suitability for lifelong-learning scenarios, which presupposes that the time and space complexity is strictly bounded. In other words, models are not allowed to grow endlessly, but at the same time are required to learn further even when these bounds are reached. Hence, a mechanism to compress the representation or to forget irrelevant information is necessary. The survey concludes with a table which summarizes the results of the experiments and provides a quick overview to streamline the choice of an appropriate algorithm.

3.1 OVERARCHING LEARNING SCENARIO

In the following, incremental learning algorithms are defined in the context of streaming data, establishing the foundation of this thesis. Associated challenges are extensively discussed and point to respective research contributions.

3.1.1 *Definition*

We focus on supervised learning in classification tasks. The aim in supervised classification is to predict a target variable $y \in \{c_1, \ldots, c_C\}$, that can be one of C classes, given a feature vector $\mathbf{x} \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$. Thereby, the domains are either real-valued $\mathcal{X}_j = \mathbb{R}$ or consist of discrete $\mathcal{X}_j = \{a_1^j, \ldots, a_{v_j}^j\}$ with $v_j \geq 2$ different attribute values a^j.

A lot of ambiguity is involved regarding the definition of incremental and online learning in the literature. Some authors use them interchangeably, while others distinguish them in different ways. Additional terms such as lifelong- or evolutionary learning are also used synonymously. In the context of this thesis, an *incremental learning algorithm* is defined as one that generates on a given stream of data $s_1, s_2, ..., s_t$ a sequence of models $h_1, h_2, ..., h_t$. In our case, s_i is labeled training data $s_i = (\mathbf{x}_i, y_i) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \times \{c_1, \ldots, c_C\}$ and $h_i : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \mapsto \{c_1, \ldots, c_C\}$ is a model function solely depending on h_{i-1} and the recent p examples $s_i, \ldots, s_{i-p+1}$, with p being strictly limited. In other words, the model has to adapt gradually without a complete retraining. In particular, learning from a data stream implies one-pass learning, meaning the data is processed only once. Therefore, we exclude for example the training of a Deep Neural Network model based on the Stochastic Gradient Descent (SGD) because the whole data would have to be stored, since plenty repetitions through the dataset are required to achieve a reasonable performance.

All incremental learning algorithms are also *online learning algorithms* because they are able to perform in the *online learning setting*, which denotes one typical processing scheme where the model h_{i-1} first issues a predicted label $\hat{y}_i$ for each input $\mathbf{x}_i$ before the true label y_i gets revealed and a subsequent model h_i is constructed. The classification performance is measured based on the equality between $\hat{y}_i$ and y_i. In contrast, classical batch algorithms are applied within the *offline learning setting* where they construct one static model on basis of one large training set. Subsequently, the performance is evaluated on a separate test set. Incremental algorithms can also be used within the offline learning setting, which is rarely done in real-world applications, since batch algorithms usually yield a higher performance (see Section 3.1.2).

Lifelong-learning algorithms are incremental algorithms which are additionally bounded in model complexity and run time, capable of endless learning on a device with restricted resources. This chapter focuses solely on independent and identically distributed (i.i.d.) data, meaning we rely on the assumption that the environment is yet unknown regarding its complexity, but it essentially is fixed. Non-stationary environments are discussed in detail in Chapter 4.

3.1.2 *Challenges*

Even though the incremental learning paradigm has various advantages in comparison to classical batch learning, it poses also a number of different challenges. These are not only concerning the learning itself, but also imply practical challenges as well. In the following, we discuss such challenges in detail.

How to Get Online Ground Truth?

Streaming data can be found in plenty domains such as robotics, manufacturing, health care (Quigley et al., 2009; Chen et al., 2014; Raghupathi & Raghupathi, 2014) and the uprising IOT technology fosters its prevalence to new areas (Gubbi, Buyya, Marusic, & Palaniswami, 2013).

However, streaming data often lacks supervised information. Getting supervised information is in general challenging, and particularly large models, e.g. Deep Neural Networks, require huge amount of labeled data. Still nowadays, the majority of the data is manually labeled by human annotators and therefore very expensive. In the case of offline learning, the labels can be collected over a long time period, whereas online learning requires the ground truth on the fly. Fortunately, most real-world tasks can tolerate a certain delay in model adaptation and consequently a delay in accessing the ground truth, opening up various possible scenarios where online learning can be applied at. In the following, four conditions are discussed that enable the extraction of label information on the fly:

1. *The ground-truth information can be automatically extracted in retrospective.*
   This is often the case for prediction tasks. One example is the application described in Section 5.2. Here, we used incremental learning to predict the course of actions a car driver will take at an intersection. As soon as the car has passed the corresponding intersection the recorded data can automatically be labeled and fed back into the incremental algorithm.

2. *The classification is possible by means of machine learning models starting from a certain point in time, but difficult in advance.*
   One exemplary application scenario is given in Section 5.3, where the objective is to classify human motions as quickly as possible. Even after the motion has been performed, it is still difficult to classify the motion. Nonetheless, this task is substantially easier than classifying it on the fly, since the model is allowed to peek in the future. Hence, one approach in such a scenario is to use the output of another machine learning model, which classifies the event after it has occurred, as ground truth. Figure 3.1 shows the corresponding system architecture. Since the model used for ground-truth extraction might be faulty, such approaches induce the requirement that the chosen incremental algorithms can handle noisy training samples.

3. *The feedback is explicitly provided by the user.*
   This is particularly relevant for product personalization. For instance, an individual user marks emails as spam for spam classification, but also in human-robot interactions the labels may be explicitly demanded. An interactive human-robot scenario is presented in Section 5.1. The robot incrementally learns new objects and classifies them in an outdoor environment. Thereby,

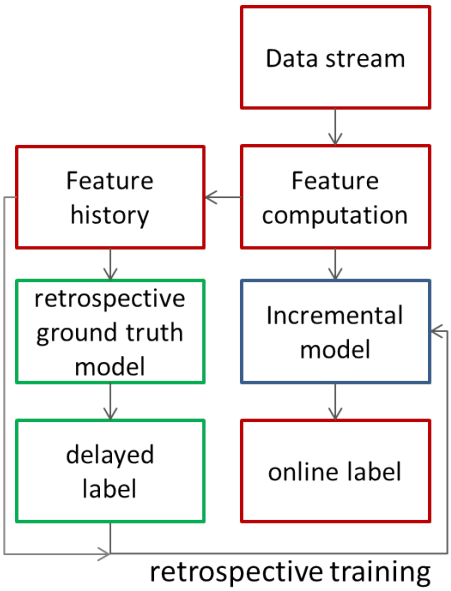

Figure 3.1: Online learning architecture. The ground truth is determined with an additional model which is allowed to buffer some data and classify with delay. These delayed labels are used to train the incremental model in retrospective.

a human can conveniently label objects live or in retrospective on a tablet.

4. *The supervised signal is provided by other systems or sensors which rely on cues that may not always be available.*
   One example is the estimation of road lanes, which can be directly detected by a vision-based system relying on corresponding markings. However, lane markings are sometimes missing in case of rural roads for example. One possible solution is to use the lane-detection system to extract ground-truth information when the markings are present. This enables the model to learn the estimation of lane positions based on alternative features such as road width, the position of the ego car as well as those of other traffic participants. Ideally, both approaches are fused and applied depending on the availability of lane markings, increasing the robustness of the overall system.

How to Evaluate the Model Performance?

In contrast to static models, generated by offline learning algorithms, the evaluation of an incremental model that dynamically evolves can be quite challenging due to the temporal factor. We discuss two diverse evaluation metrics which allow the inference of different aspects regarding the algorithmic performance and provide together even a deeper insight.

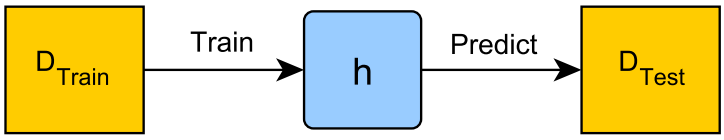

Figure 3.2: Classical scheme of evaluating a batch algorithm in off-line mode.

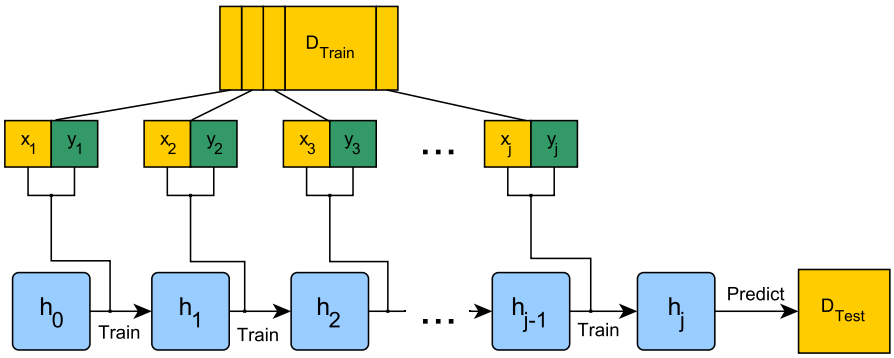

Figure 3.3: The process of testing an incremental algorithm in the off-line setting. Noticeably, only the last constructed model is used for prediction. All data used during training $(\mathbf{x}_i, y_i)$ is obtained from the training set D_{train}

Offline Evaluation Let us remind that the classical offline learning paradigm works as follows. A batch algorithm generates a model h based on a training set $D_{\text{train}} = \{(\mathbf{x}_i, y_i) \,|\, i \in \{1, \ldots, j\}\}$. In the subsequent test phase, the model is applied on another set $D_{\text{test}} = \{(\mathbf{x}_i, y_i) \,|\, i \in \{1, \ldots, k\}\}$, whose labels are kept hidden. Figure 3.2 depicts the process. The model predicts a label $\hat{y}_i = h(\mathbf{x}_i)$ for every point $x_i \in D_{\text{test}}$ and the 0-1 loss $\mathcal{L}(\hat{y}_i, y_i) = \mathbb{1}(\hat{y}_i \neq y_i)$ is calculated. The average error on the test set enables an analysis in terms of the generalization ability to unseen examples.

Incremental algorithms also can be applied in this setting as it is shown by Figure 3.3. The training data is sequentially processed in a predefined order. The algorithm generates for the sequence of tuples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots (\mathbf{x}_j, y_j)$ a corresponding sequence of models $h_1, h_2, \ldots, h_j$. Thereby, a model h_i is solely based on the previously constructed model and a limited amount of p recent tuples

$$h_i = \text{train}(h_{i-1}, (\mathbf{x}_i, y_i), \ldots, (\mathbf{x}_{i-p+1}, y_{i-p+1})).$$

Only the last model h_j is applied on the test set to determine the *Test Error* (TE) $\hat{E}$

$$\hat{E}(D_{\text{test}}) = \frac{1}{k} \sum_{i=1}^{k} 1 - \mathcal{L}(\hat{y}_i, y_i) = \frac{1}{k} \sum_{i=1}^{k} 1 - \mathcal{L}(h_j(\mathbf{x}_i), y_i).$$

This metric evaluates the generalization ability of the last model and neglects all preceding models. Hence, it contains no information about the learning speed, i.e. the slope of the function we would get by evaluating all intermediate models h_i the same way.

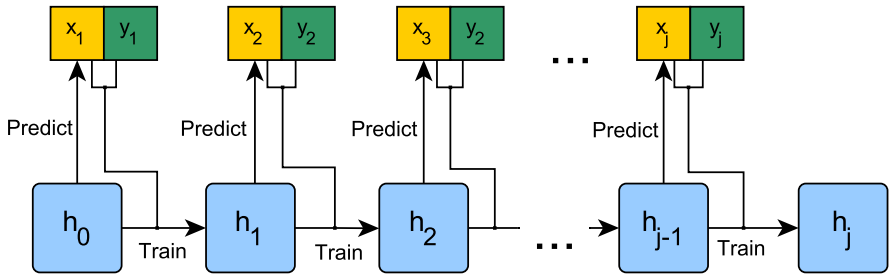

Figure 3.4: The online learning scheme: Data is not split into training- and testing set. Instead, each model predicts subsequently one example, which is afterward used for the construction of the next model.

Online Evaluation Learning from data streams is usually evaluated in the online learning setting, which is depicted in Figure 3.4. A potentially infinite sequence $S = (s_1, s_2, \ldots, s_t, \ldots)$ of tuples $s_i = (\mathbf{x}_i, y_i)$ arrives one after another. As t represents the current time stamp, the learning objective is to predict the corresponding label y_t for a given input x_t, which is supposed to be unknown. The prediction $\hat{y}_t = h_{t-1}(\mathbf{x}_t)$ is done according to the previously learned model h_{t-1}. Afterward, the true label is revealed and the loss $\mathcal{L}(\hat{y}_t, y_t)$ determined. The so called Interleaved-Test-Train Error (ITTE) for a sequence up to the current time stamp t is given by:

$$E(S) = \frac{1}{t} \sum_{i=1}^{t} 1 - \mathcal{L}(h_{i-1}(\mathbf{x}_i), y_i). \tag{3.1}$$

The main difference to the previous setting is that all intermediate models are considered for the performance evaluation, but each of them predicts only the following example. Additionally, the data is used more efficiently because the sets for training and testing are not disjunct, but instead each instance is used for model testing and adaption. In case of i.i.d. data, the ITTE measures the average generalization ability of all constructed models.

The combination of both metrics enables conclusions about the learning curve: Having two different models A and B with the same test error, but A having a lower ITTE implies that A on average has a steeper learning curve than B and vice versa. In Section 3.4, both evaluation settings are applied to thoroughly analyze the aspects of state-of-the-art incremental learning methods.

How to Adapt the Model Parameters?

In contrast to the static models constructed by batch algorithms, incremental models are continuously evolving. The instance-by-instance processing scheme demands a model adaptation on the fly, which can only be optimal in respect to a small part of the data. Hence, the learning performance is often worse than those of batch algorithms which can optimize the parameters in respect to the whole dataset. In particular, the learning itself can be less stable, due to the optimization based on very few instances, leading flatter learning curve with a

higher tendency to oscillations. Consequently, incremental learning is more sensitive to the setting of hyperparameters such as the learning rate and the resulting model often depends on the order of the data.

One fundamental challenge in incremental learning is the so called stability-plasticity dilemma (Grossberg, 1988), which describes the contradicting demands of a high stability to preserve old patterns and a high plasticity to quickly adapt to new ones. The effect of catastrophic forgetting (French, 1999) is one special case of the stability-plasticity dilemma where a model with a high plasticity forgets former patterns over time as it is continuously is fed with new information. The stability-plasticity dilemma has been tackled by approaches which dynamically adapt their plasticity. It is often reduced over time in linear or exponential way as it is done in simulated annealing (Van Laarhoven & Aarts, 1987) or adapted based on gradient information (Zeiler, 2012; Shalev-Shwartz, Singer, Srebro, & Cotter, 2011). However, these methods usually add further hyperparameters to the overall system and are known to perform sub-optimal for various cases. In a nutshell, the problem of how to incrementally adapt the model parameters is yet largely unsolved.

There are some lossless incremental versions of some batch algorithms: Naive Bayes (NB) (Zhang, 2004a) or k Nearest Neighbor (kNN) (Cover & Hart, 1967) as classifiers, but also a least squares regression model can be phrased as incremental variants, which exactly compute the result of their respective batch counterparts without any loss due to approximations. In particular, the resulting models are independent from the order of the data instances.

How to Adapt the Model Complexity?

Machine learning algorithms can be classified into parametric and non-parametric models. Parametric models are static with a predefined number of model parameters. These can be given by the employed model itself as it is the case for linear models or on the basis of some hyperparameters e.g. specifying the structure of a Neural Network. Non-parametric models are dynamic and add model parameters as they process data. Examples for non-parametric models are kNN or Support Vector Machines with Gaussian kernel if represented in dual space. Obviously, a model with a predefined size is limited in the amount of information it can store (Shannon, 2001) and consequently also in the complexity of the tasks it can effectively handle. Non-parametric models have the advantage of being able to align their model complexity to the demands of the task. In case of offline learning, it is often sufficient to manually adapt the model complexity via setting corresponding hyperparameters, since the whole dataset is available. However, in the case of incremental learning from data streams the complexity of the task is a priori unknown, and therefore the ability to dynamically grow the model becomes more important, in particularly it provides an additional degree of freedom to handle the stability-plasticity dilemma.

Model growth induces further challenges such as how and when to adapt the model complexity. A proven approach is to incrementally expand the model in such a way that the corresponding cost function is minimized in a greedy way. Even though prototype-based models allow to locally extend the model complexity without impairing the global stability, it is unclear how to grow the model such that a global cost function is optimized. The ILVQ is such a method which has often been applied in various areas (Biehl, Bunte, & Schneider, 2013; Wersing & Körner, 2003; Carlevarino, Martinotti, & Metta, 2000). However, the addition of new prototypes was yet always based on heuristics. One of our contribution within this thesis will be an incremental model architecture which is driven by the objective of cost optimization (Section 3.2).

The question when to grow the model is strictly coupled with the crucial trade-off between fast model adaptation and high model complexity. While fast model adaptation is desired for quick convergence, a too quickly expanding model has not enough time to utilize the model parameters in an optimal way, since the growth is forced to happen on a small amount of data. Additionally, large models have a high space- and time complexity. The timing of model growth is often based on some hyperparameters which have to be set a priori, requiring the consideration of different aspects such as the amount of expected data, the limits in terms of memory and processing time and the necessity of a quick adaptation. In this context, we provide a contribution for one particularly popular incremental-learning algorithm, the Very Fast Decision Tree (VFDT) (Domingos & Hulten, 2000) (Section 3.3). Concretely, we propose a split-time prediction that uses local information to dynamically decide when to grow the model, replacing the original scheme which is based on regular intervals that have to be predefined. Our approach does not only reduce the time complexity without a loss of classification performance, but it also mitigates the dependence on hyperparameters, increasing the robustness and adaptivity of the algorithm.

3.2 INCREMENTAL LEARNING VECTOR QUANTIZATION

Prototype-based approaches such as Neural Gas (Martinetz & Schulten, 1991) or Learning Vector Quantization (LVQ) (Kohonen, 2001) are popular options among online learning algorithms, with applications ranging from biomedical data analysis (Biehl et al., 2013), image recognition (Wersing & Körner, 2003), to robotics (Carlevarino et al., 2000). The ability of structure adaptation according to the complexity of a given task is helpful and becomes even more crucial with regard to streaming data, often violating the assumption of data being i.i.d. (Carlevarino et al., 2000; Jin & Hammer, 2014). Dynamic prototype insertions and deletions enable this flexibility intuitively. Moreover, these can be done in an efficient and incremental way because the model structure is only locally affected.

Regarding prototype-based methods, the question "How to adapt the model complexity?" can be mapped to "Where to place new prototypes?". In case of LVQ, several proposals for prototype placement strategies were made (Grbovic & Vucetic, 2009; Bermejo, Cabestany, & Payeras-Capellà, 1998; Kirstein, Wersing, & Körner, 2005). Even though they often place new prototypes in reasonable locations, they are based on heuristics without mathematical justification and fail when their assumptions do not apply. In contrast, we propose a placement strategy which relies on the derivation of LVQ as cost function optimization; it optimizes an approximation of these costs for improved robustness. Since we confine the optimization on a limited number of recent samples, the resulting implementation offers the advantages of an adjustable and strictly limited memory consumption paired with linear complexity. Such linear time and constant memory conditions are especially relevant for efficient mobile applications.

3.2.1 *Foundation*

Our method is based on the Generalized Learning Vector Quantization (GLVQ), which frames LVQ as cost function optimization problem.

Generalized Learning Vector Quantization (GLVQ)

Given a classification task of C classes, the training set $X = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{c_1, ..., c_C\}\}_{i=1}^m$ is approximated by an LVQ classifier with a set of p prototypes $W = \{(\mathbf{w}_j, l_j) \in \mathbb{R}^n \times \{c_1, ..., c_C\}\}_{j=1}^p$. The Voronoi region of a prototype $(\mathbf{w}_j, l_j)$ is defined as $V_j = \{\mathbf{x} \in X | \ \|\mathbf{x} - \mathbf{w}_j\| \leq \|\mathbf{x} - \mathbf{w}_k\| \ \forall \ j \neq k\}$. A given data point $\mathbf{x}_i$ is classified according to the label of its closest prototype using a distance measure d such as the squared Euclidean distance $d(\mathbf{x}, \mathbf{w}) = \|\mathbf{x} - \mathbf{w}\|^2$.

Sato and Yamada (1995) introduced the generalized LVQ (GLVQ) which, in contrast to previous heuristic approaches, minimizes the cost function

$$J(X, W) = \sum_{i=1}^{m} \Phi((d_i^+ - d_i^-)/(d_i^+ + d_i^-)), \tag{3.2}$$

where Φ is a monotonically increasing function, e. g. the logistic function. The distance of a sample $\mathbf{x}_i$ to its closest prototype $\mathbf{w}^{\pm}$ of the correct / incorrect class is denoted by $d_i^{\pm}$. By updating the prototypes for each data point as follows

$$\mathbf{w}^{\pm} := \mathbf{w}^{\pm} - \lambda \frac{\partial J(X, W)}{\partial \mathbf{w}^{\pm}}, \tag{3.3}$$

where λ is the learning rate, the cost function is minimized in a stochastic gradient descent scheme.

3.2.2 *Related Work*

Various prototype placement strategies have been proposed for online variants of LVQ. All of them propose new prototypes based on

recently misclassified instances. Kirstein et al. (2005) suggest to select the misclassified samples that are the closest to prototypes of another class. This choice shall lead to insertions along class borders and was demonstrated within various scenarios. Grbovic and Vucetic (2009) cluster misclassified samples per class and select centroids of the biggest clusters as new prototypes. A similar approach was chosen by Bermejo et al. (1998). But instead of clustering, he determines the Voronoi region containing the most misclassified samples of one class. The mean of these samples is chosen as a new prototype.

3.2.3 *Learning Architecture*

The GLVQ in its original form is used as offline algorithm where the number of prototypes is predefined and multiple iterations over the dataset are performed during training. We adapt the GLVQ to an incremental architecture termed Incremental Learning Vector Quantization (ILVQ). It processes each instance only once and inserts new prototypes on the fly. Every prototype $\mathbf{w}_j$ has its own linearly decreasing learning rate λ_j to approach the stability-plasticity dilemma (Kirstein et al., 2005). As an overview, the learning architecture works as follows:

The learning architecture initially does not have any prototype, i. e. $W = \emptyset$. For each new sample $(\mathbf{x}_i, y_i)$ we test whether y_i is a new class. If y_i is not yet represented in the model, this sample is directly added as a prototype. Otherwise, we perform the GLVQ-updates (3.3) and store the sample with its distance information in a short-term memory of limited size, replacing old ones if necessary. If the sample is misclassified, we increment an error count. As soon as a predefined number of errors occurs, the employed placement strategy provides a new prototype which is added to the set of prototypes W. Then, we update distances within the short-term memory. The error count is reset and we start over again. In the remainder of the section, the single steps are described in more detail.

Limited Sample Memory

Apart from the prototypes, we maintain a short term memory which is defined as a list

$$\Psi := [(\mathbf{x}_i, y_i, d_i^+, d_i^-) \mid i \in \{1, ..., \tau\}], \tag{3.4}$$

containing entries for the recent τ samples. Every additional sample leads to the deletion of the oldest entry as soon as the limit of τ stored entries is reached, i. e. $|\Psi| = \tau$. Based upon Ψ, placement strategies propose new prototypes.

The GLVQ-updates (3.3) change prototype positions and corresponding sample distances. However, we neglect these changes within Ψ for the sake of efficiency. Hence, Ψ contains approximations of the actual distances and their quality depends on the magnitude of the learning rates λ_j as well as on the window size τ. During our experiments, this simplification had negligible or even no consequence on

the systems' performance. We used a window size of $\tau = 200$ for all experiments.

Insertion Timing

The moment a new prototype is inserted is based on error counting as already proposed in (Kirstein et al., 2005). Whenever the error count reaches a threshold, a new prototype is added and the error count is reset to zero. This simple and computationally cheap approach couples growing speed strictly with systems' performance on training data. The learning architecture evolves fast in case of high error rate but changes only slightly when few errors are made. Consequently, a system will grow as long as it does not classify perfectly (i.e. infinitely in case of overlapping classes). This can be avoided by adding new prototypes only if the cost function is significantly reduced and / or by removing superfluous prototypes as suggested in (Grbovic & Vucetic, 2009). However, we focus in this section solely on the placement of new prototypes.

Insertion of Prototypes

Whenever a new prototype $(\mathbf{w}, l)$ is proposed by a placement strategy and inserted into the network, i. e. $W := W \cup (\mathbf{w}, l)$, the affected distances stored in Ψ are updated in the following way:

$$\forall\ (\mathbf{x}_i, y_i, d_i^+, d_i^-) \in \Psi : \tag{3.5}$$

$$d_i^+ := d(\mathbf{x}_i, \mathbf{w}),\ \text{if } y_i = l \wedge d_i^+ > d(\mathbf{x}_i, \mathbf{w}) \tag{3.6}$$

$$d_i^- := d(\mathbf{x}_i, \mathbf{w}),\ \text{if } y_i \neq l \wedge d_i^- > d(\mathbf{x}_i, \mathbf{w}). \tag{3.7}$$

3.2.4 *Proposed Placement Strategy: COSMOS*

Our placement strategy minimizes the cost function of the recent past based on randomly sampled prototype positions. We term the method COSt MinimizatiOn Sampling (*COSMOS*). *COSMOS* relies on a minimization of the GLVQ cost function which is approximated on the basis of Ψ. Thereby, prototypes are taken from candidate positions using a random subset $\hat{\Psi} \subseteq \Psi$ of size $|\hat{\Psi}| = \hat{\tau}$ only. For every candidate $(\mathbf{x}_i, y_i)$ in $\hat{\Psi}$, its effect on the costs can efficiently be approximated as follows:

1. Extend $\hat{W} := \{W \cup (\mathbf{x}_i, y_i)\ \}$.

2. Update Ψ as described in (3.5) but store the result temporarily in another list Ψ'.

3. Calculate the cost function value $J(\Psi', \hat{W})$ on the basis of the distances stored in Ψ'.

Algorithm 3.1 The COSMOS placement strategy

Inputs:
Ψ : sliding window data
$\hat{\tau}$: sample size
Output:
Tuple consisting of the prototype position and label
Initialize:
$minCost \leftarrow 1$
$proto \leftarrow null$
$\hat{\Psi} \leftarrow getRandomSubset(\Psi, \hat{\tau})$
for all $(\mathbf{x}, y) \in \hat{\Psi}$ **do**
$\Psi' \leftarrow updateShortTermMemory(\Psi, (\mathbf{x}, y))$
$cost \leftarrow calculateCost(\Psi')$
if $cost < minCost$ **then**
$minCost \leftarrow cost$
$proto \leftarrow (\mathbf{x}, y)$
return *proto*

Table 3.1: The evaluated placement strategies and their abbreviations.

Abbr.	Prototype placement location
Closest	Misclassified instance that is the closest to a prototype of another class (Kirstein, Wersing, & Körner, 2005).
Cluster	Center of the largest cluster, extracted from class-wise clustering of misclassified samples (Grbovic & Vucetic, 2009).
Voronoi	Mean of the misclassified samples within the voronoi cell with the most misclassifications (Bermejo, Cabestany, & Payeras-Capellà, 1998).
COSMOS	Sampled instance which minimizes the cost function on a sliding window the most (proposed method).

The sample with smallest value $J(\Psi', \hat{W})$ is added as a new prototype to W, and Ψ updated accordingly. The complexity is $\mathcal{O}(\hat{\tau} \cdot \tau)$ and pseudo code is depicted in Algorithm 3.1.

An early version of the learning architecture in combination with the COSMOS placement strategy has been proposed in (Losing, 2014). The contribution here, is a concise and formal description as well as a richer analysis.

3.2.5 *Experiments*

We compare state-of-the-art placement strategies with our proposal. These were already described within the Related Work Section 3.2.2 and are listed in Table 3.1. In case of *Closest*, the popular k-Means algorithm was used to cluster the misclassified instances. Since k has to be predefined, the common rule of thumb $k = \sqrt{\frac{n}{2}}$ was utilized. However, this value was multiplied by a factor of two to achieve the

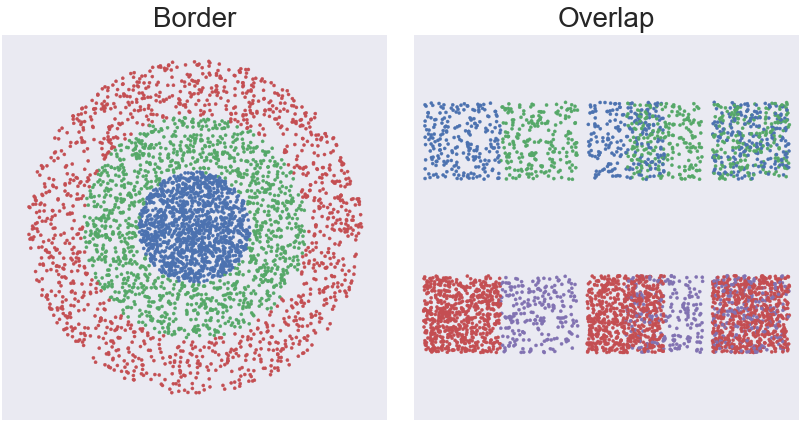

Figure 3.5: Artificial datasets *Border* and *Overlap*. Different classes are coded by different colors. The *Border* dataset (Fig. 3.5, left) consists of three circular classes. Each class has the same number of points and is uniformly distributed points. The dataset *Overlap* (Fig. 3.5, right) contains uniform squared distributions which overlap to various degrees. The upper row classes have the same densities whereas, below, the green class is three times denser than the black.

best experimental results. Each placement strategy was integrated in our learning architecture ILVQ (see section 3.2.3). The resulting four online learning algorithms are trained using identical conditions on two artificial datasets with different characteristics. The main aim of these two-dimensional datasets is to investigate how precisely the strategies can represent the real class borders. The arrangement of these problems is shown in Figure 3.5. We used 70% of the data for training and the rest for testing.

Overlapping class distributions are usually the most difficult to deal with, since a complete separation is not possible. Given an overlap, the densest class should be preferred in the Bayesian optimum. These challenges are incorporated in the *Overlap* dataset, visualized in Figure 3.5 on the right.

Results

One example of a final prototype arrangement of each strategy can be seen in Figure 3.6. Boundaries generated by *Closest* deviate the most from actual borderlines. On the *Border* dataset, samples are separated very accurately at few specific spots, which are clustered with multiple prototypes, but a significant portion of the actual border is uncovered, causing the majority of misclassifications. Whether a sample is promoted to a prototype or not, is solely based on its distance to nodes of other classes. This leads to insertions along class borders and explains why the algorithm often gets stuck. A prototype inserted close to a border is very likely to cause new misclassified samples which are, in turn, closely located to this new prototype. Therefore, many unnecessary insertions are used to represent class boundaries at specific areas

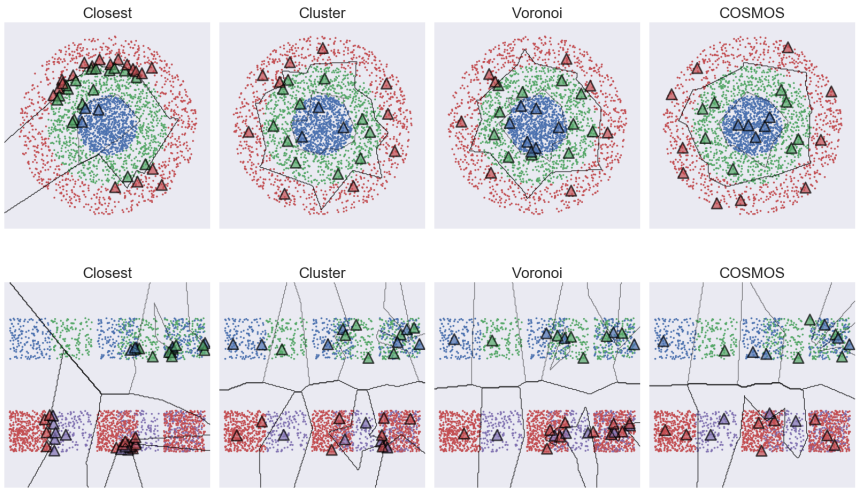

Figure 3.6: Networks of each placement algorithm after the training of the datasets *Border* (top) and *Overlap* (bottom). Prototypes are symbolized by triangles and black lines represent the learned class boundaries.

in contrast to some completely neglected parts. In case of the *Overlap* dataset, this problem is even more severe and entraps the algorithm to insert prototypes in one exclusive area. *Closest* is very sensitive to noise because a single sample can cause multiple and consecutive prototype insertions.

Results of *Cluster* and *Voronoi* are quite similar. Prototypes are spread fairly regular and approximate the actual boundaries well. In case of ambiguous regions, unnecessary prototypes of both classes are placed regardless whether density proportions between classes are existing or not. Initially, classes with higher densities are favored because large clusters are likely to be found among them. But, as soon as they are covered, prototypes for less dense classes are inserted too and deteriorate the error rate again. These strategies perform similarly because both compute clusters of misclassified samples and place prototypes in their centroids. The clustering quality is crucial for the performance, since too fine clustering leads to more prototypes than necessary, whereas too coarse clustering causes a misplacement of prototypes because centroids of too large clusters might be located within samples of another class. *Voronoi* clusters spatially by incorporating the available Voronoi cells, which are shrinking continuously as more nodes are introduced into the net. Therefore, clustering is initially coarse and fine at the end. The few misplaced prototypes (Fig. 3.6) are caused by the coarse clustering at beginning of the training. *Cluster*, on the other hand, utilizes k-Means, which requires the number of clusters as parameter in advance. Since a rather high number is chosen, coarse clustering is avoided and the number of samples per cluster is still high enough to prevent too fine clustering. Both algorithms are robust against noise because a high amount of accumulated noisy points of the same class is necessary to cause an insertion there.

Table 3.2: Results on the artificial data. The averaged test error and net-size of ten repetitions are depicted.

Dataset	*Closest*		*Cluster*		*Voronoi*		*COSMOS*	
	TE	#Nodes	TE	#Nodes	TE	#Nodes	TE	#Nodes
Border	9.83	58.2	8.07	42.9	8.29	46.4	**6.49**	**38.2**
Overlap	34.24	29.8	25.15	25.6	25.92	26.4	**21.26**	**21.3**
∅	22.03	44.0	16.61	34.3	17.10	36.4	**13.87**	**29.7**

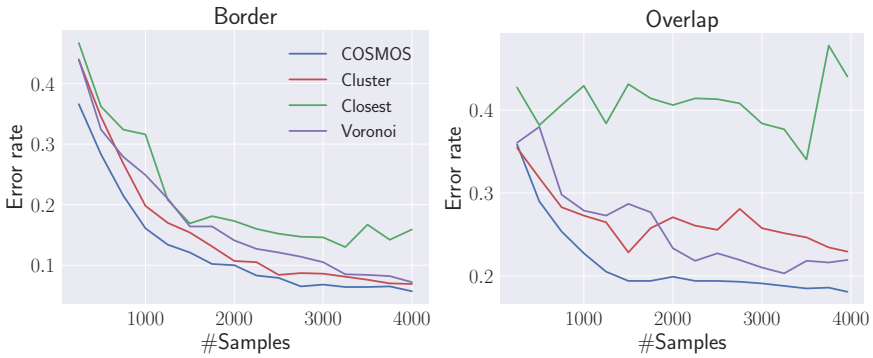

Figure 3.7: The learning curve for the artificial datasets. *COSMOS* continuously has the lowest error rate and particularly excels on the *Overlap* dataset.

COSMOS reproduces the true class borders the most accurately. It distributes nodes regularly over the samples and balances well the trade-off between prototypes being placed as close as possible to samples of its own class and far from those of another class. Its cost function optimization regards performances of all classes, which is crucial for dealing with overlapping distributions as it can be seen by the result on the *Overlap* dataset. It is able to assign ambiguous and non-ambiguous regions correctly and thereby considers also density differences.

Table 3.2 gives an overview of the achieved test error rates as well as the final net size. *Closest* is the worst strategy on both datasets, even though it uses the most prototypes. *Voronoi* and *Cluster* perform similarly with the latter being slightly better. *COSMOS* delivers the lowest error rates in spite of requiring the least number of nodes. Its lead is especially significant on the *Overlap* dataset where it outperforms the rest in every aspect.

The achieved error rates during training are depicted in Figure 3.7. *Closest* quickly loses touch, whereas *COSMOS* is in the lead throughout training.

Comparison against the Incremental Support Vector Machine (ISVM)

To get a better picture of the performance of the online learning system in combination with the proposed *COSMOS* insertion strategy,

Table 3.3: Comparison against the ISVM on the COIL dataset. *COSMOS-GMLVQ* learns additionally the metric of the input space. The test error (TE) and the number of used nodes are given after an increasing amount of used training examples. Pairs of TE and the number of nodes which are not Pareto dominated are marked in bold.

Method	TE / #Nodes 500 samples	TE / #Nodes 1000 samples	TE / #Nodes 1700 samples
COSMOS-GLVQ	16.5 / 337	9.6 / 560	6.4 / 810
COSMOS-GMLVQ	**15.0 / 330**	**8.1 / 539**	**5.3 / 760**
ISVM	14.4 / 1834	7.3 / 2139	**4.4 / 2501**

a comparison with the Incremental Support Vector Machine (ISVM) was done (Cauwenberghs & Poggio, 2001; Diehl & Cauwenberghs, 2003; Laskov, Gehl, Krüger, & Müller, 2006). We used the ISVM implementation provided by Diehl[1] (Diehl & Cauwenberghs, 2003) with Gaussian kernel ($C = 1024$, $\sigma = 0.05$). The ISVM stores the support vectors, and additionally a set of reserve vectors which are the inputs that are most likely to become support vectors in the future. This can be seen as a counterpart to our short-term memory of recent samples Φ. Therefore, we also limited the number of reserve vectors to a maximum of 200 samples. The ISVM was trained in a one-vs-all scheme. We increased considerably the prototype insertion frequency to achieve maximum performance. We evaluate COSMOS also in combination with the Generalized Matrix LVQ (GMLVQ) (Schneider, Biehl, & Hammer, 2009), which is more powerful in comparison to the GLVQ because it incorporates metric learning of the input space.

We evaluated our approach on the basis of the well-known real-world benchmark COIL-100 (Nene, Nayar, & Murase, 1996). This RGB-image dataset consists of 72 views for 100 objects each. These are placed in the coordinate origin and rotated around the Z axis in steps of 5°. The images have a size of 128×128 pixel and we encoded them on the basis of the RG-chromaticity space (Jain & Li, 2005) using 21 dimensions. As done by Wallraven, Caputo, and Graf (2003), we use a subset of 17 views per object, resulting in views every 20°, for training. Table 3.3 shows the accuracy as well as the stored number of prototypes / support vectors after an increasing number of training examples.

3.2.6 *Discussion*

Our incremental learning architecture combines GLVQ with the COSMOS placement strategy, optimizing the cost function on the basis of a limited sample history. The comparison on artificial and image based datasets showed the superiority of the proposed placement strategy over current heuristic-based strategies. The optimization has a linear complexity and enables the handling of overlapping distributions.

1 Code at https://github.com/diehl/Incremental-SVM-Learning-in-MATLAB

Paired with the GMLVQ, our architecture achieves similar results as the well-established ISVM, but with a significantly sparser model. Thus, we provide a novel competitive local incremental learning scheme, which is strictly based on cost function optimization, mitigating the challenge how to increase model complexity in a popular distance-based online learning methodology.

The ILVQ architecture can further be improved in terms of a more sophisticated way to determine the timing to prototype insertions. New prototypes should only be added if the performance gain justifies the burden of additional model parameters. Techniques such as the Akaike Information Criterion (AIC) or the Bayesion Information Criterion (BIC) (Akaike, 1998; Liddle, 2007) could be used to quantify the trade-off between performance and model complexity. Furthermore, the number of added prototypes is currently unlimited, which is an issue in case of lifelong-learning scenarios. One approach could be to delete prototypes as soon as a predefined limit is reached. The deletion of prototypes also paves the way to apply prototype-based methods within non-stationary environments, where they can provide a sparse and efficient alternative to current methods, particularly considering their high degree of transparency and interpretability, a crucial requirement for real-world applications. In the context of LVQ, the removal of prototypes has also been considered in a purely heuristic way (Grbovic & Vucetic, 2009; Fischer, Hammer, & Wersing, 2015a). Analogous to COSMOS, a deletion strategy could be implemented which minimizes the cost function in a similar manner.

3.3 LOCAL SPLIT-TIME PREDICTION

Numerous state-of-the-art algorithms for supervised learning from data streams are based on decision trees due to their high efficiency and accuracy (Losing, Hammer, & Wersing, 2018b). One of the most prominent is the Very Fast Decision Tree (VFDT) (Domingos & Hulten, 2000), an incremental, anytime decision tree induction algorithm, capable of learning from massive data streams. Incremental decision trees grow their model in terms of adding new splits which greedily optimize the class impurity within their leaves. The VFDT relies on the Hoeffding bound to decide when to grow its model. Concretely, it verifies whether enough evidence has been collected such that the currently best split-attribute is indeed the best one with a predefined probability. This formal foundation enables the construction of trees which are provably similar to those of classical batch algorithms such as C4.5 (Quinlan, 1993) or CART (Breiman, Friedman, Olshen, & Stone, 1993). However, the continuous repetition of the mention verification process is usually the most time-consuming step of the algorithm (Domingos & Hulten, 2000). To reduce the computational load of these *split-attempts* the VFDT incorporates the hyperparameter $n_{\min}$, controlling the number of examples a leaf has to accumulate before a new split-attempt is performed. Even though this regular approach reduces the computational complexity, it also leads to a *split-delay*, since more

examples are accumulated than the strict minimum. Consequently, the tree growth is hampered, often resulting in a lower classification performance.

Instead of the periodic split-attempts, we propose to predict the local *split-time*, i.e. the number of examples a leaf needs to accumulate before a split is performed, avoiding unnecessary attempts when splits are unlikely to be performed. Concretely, we use the class distributions of previous split-attempts to approximate the minimum time until the Hoeffding bound is met. This cautious approach splits by design with a low delay, but still substantially reduces the number of split-attempts. Our approach does not only increase the efficiency of the algorithm, but moreover its robustness in terms of a reduced dependence on the setting of $n_{\min}$.

Recently, split-time prediction was solely evaluated on the basis of the classification performance (Garcia-Martin, 2017), which is not only imprecise, but can even be misleading in case of overfitting. Instead, we formally define the term *split-delay*, which exactly measures the delay caused by the split-time prediction. On this basis, we perform a detailed evaluation including an analysis of the trade-off between split-delay and the number of split-attempts, the execution time and a statistical assessment of the resulting classification accuracy. Thereby, the largest of the commonly applied stream learning benchmarks are evaluated. The experiments show that our proposal significantly reduces the run time without a loss in performance.

3.3.1 *Foundation*

As usual, decision trees consist of trees where each interior node is associated to an attribute $j \in \{1, \ldots, n\}$. In case of a discrete attribute domain $\mathcal{X}_j = \{a_1^j, \ldots, a_{v_j}^j\}$, each interior node has v_j children which are mapped to the v_j different values of $\mathcal{X}_j$. Leaves maintain a distribution over the class labels $\{c_1, \ldots, c_C\}$. A given data point $\mathbf{x}$ is assigned to one leaf node according to the match of the attributes of $\mathbf{x}$ and the values of the tree's nodes, starting from the root. Since the inference of optimum decision trees is NP-hard, greedy approaches iteratively grow a tree based on impurity measures of the current nodes such as the entropy or the Gini index. In contrast to batch scenarios, where the best split-attributes are based on the whole training data, incremental decision tree induction constructs a tree subsequently based on a small sample of the data and generates splits on the fly. Algorithm 3.2 shows the pseudocode of the basic algorithm. Thereby, the impurity measure per leaf l can be given as the classical information gain, for example: For a set M of labeled data pairs $(\mathbf{x}, y)$, the entropy is defined as $H(M) = -\sum_{j=1}^{C} p_j \log_2 p_j$ where p_j is the probability of class j in M given by its relative frequency. Every discrete-valued attribute j induces a split into v_j leaves according to the possible attribute values

Algorithm 3.2 Incremental decision tree induction

Inputs:
S : stream
Initialize:
Let Tr be a tree with a single leaf l_1 (the root)
Let $S(l_1)$ be the sufficient statistics l_1 collects
Let $C(l_1) := (\frac{1}{c}, \ldots, \frac{1}{c})$ be the initial class distribution of l_1
for all $(\mathbf{x}, y) \in S$ **do**
assign $(\mathbf{x}, y)$ to leaf l_i according to matching attributes
update $S(l_i)$ with $(\mathbf{x}, y)$
update $C(l_i)$ with y
if criterion to split is fulfilled(*) **then**
choose best split attribute j of l_i depending on impurity$(S(l_i))$
split l_i along all discrete attribute values of j

$a_1^j, \ldots a_{v_j}^j$. The weighted entropy of a split of M along j is given by

$$H_j(M) = \sum_{k=1}^{v_j} \frac{|M(a_k^j)|}{|M|} H(M(a_k^j)),$$

where $M(a_k^j) := \{(\mathbf{x}, y) \in M \mid x^j = a_k^j\}$. Correspondingly, the information gain is defined as $G_j(M) = H(M) - H_j(M)$. The attribute providing the maximum gain is chosen for the split. In case of binary splits, we alternatively denote the weighted entropy in terms of both sets $M(a_1^j)$ and $M(a_2^j)$

$$H_j(M(a_1^j), M(a_2^j)) = \frac{|M(a_1^j)|H(M(a_1^j)) + |M(a_2^j)|H(M(a_2^j))}{|M(a_1^j)| + |M(a_2^j)|}. \tag{3.8}$$

Please note, that incremental decision trees usually do not store the raw samples $(\mathbf{x}_t, y_t)$ in the leaf. Instead, they derive sufficient statistics in the form of class-conditional attribute-value occurrences, enabling the calculation of the information gain with a reduced time- and space complexity.

The Very Fast Decision Tree (VFDT)

The VFDT is an incremental decision tree induction algorithm. The moment when a new split is performed is crucial with respect to learning-speed, tree complexity and overfitting. The VFDT splits a leaf only if enough evidence has been seen to guarantee that the chosen split-attribute is the best with a predefined probability. This mathematically sound approach distinguishes it from other incremental learning

trees, which heuristically decide to split based on a predefined number of examples (Saffari et al., 2009) or a predefined impurity threshold (Wang, Wan, Cheng, & Li, 2009). The VFDT relies on the Hoeffding bound, therefore it is also known under the name Hoeffding tree. Given a continuous random variable r of range R and its observed mean $\bar{r}$ after $\hat{m}$ independent observations, the Hoeffding bound states that with probability $1-\delta$ the true mean of r is at least $\bar{r}-\epsilon$ where

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2\hat{m}}}.$$

Concretely, the VFDT determines the two attributes j_1, j_2 with the highest information gain G_{j_1}, G_{j_2}. If their gain difference $\Delta G = G_{j_1} - G_{j_2}$ is larger than ϵ, then G_{j_1} is indeed the best split-attribute with probability $1-\delta$ and a split based on attribute j_1 is performed.

The VFDT also splits if too many examples are necessary to decide on the best attribute. These so-called tie-splits occur when several attributes have a similar gain and are necessary to avoid stunted tree growth, which can lead to a performance degradation particularly at the early stages of learning (Holmes, Richard, & Pfahringer, 2005). The number of examples required for a tie-split is controlled by a parameter τ which is set in terms of the Hoeffding bound. Formally, a tie-split is performed when $\Delta G < \epsilon < \tau$. In other words, a tie-split is performed after $\frac{R^2 \ln(1/\delta)}{2\tau^2}$ examples. Concretely, using the default parameter setting ($\tau = 0.05$, $\delta = 1e-7$) and assuming a range of $R = 1$ a tie-split will be done after 3224 examples, which is rather a small amount particularly considering Big Data tasks. Tie-splits are supposed to occur rarely but tend to happen very often in practice and can even be harmful because they sometimes cause imbalanced trees (Holmes, Richard, & Pfahringer, 2005). The split-criterion ((*) in Algorithm 3.2) of the VFDT is given as pseudocode in Algorithm 3.3.

Algorithm 3.3 Split criterion of the VFDT

Inputs:
$S(l_i)$: statistics of leaf l_i
$n_{\min}$: time interval for split tests
Output:
Boolean whether l_i should be split or not
if $|S(l_i)| \mod n_{\min} = 0$ **then**
determine the attributes j_1, j_2 with highest gain G_{j_1}, G_{j_2}
return ($\Delta G > \epsilon$ or $\Delta G < \tau$)
else
return false

Numeric Attributes

Incremental decision trees are required to store sufficient statistics in the leaves such that the class distribution resulting from a split can be

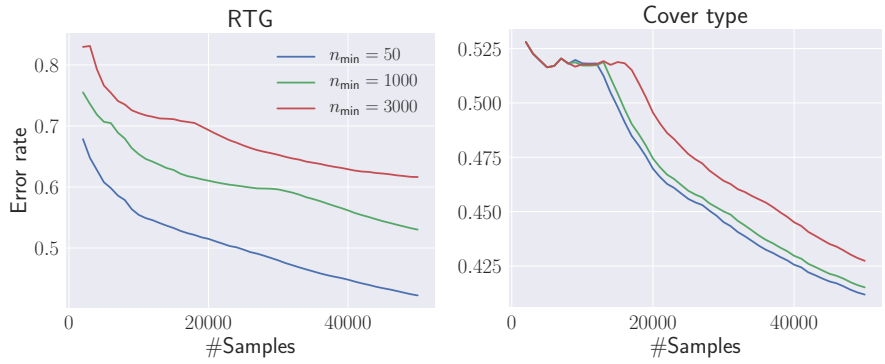

Figure 3.8: The error typically increases with higher n_{min} values, due to a slower tree growth. The gap with respect to the error rate varies depending on the dataset.

determined. This is straightforward for nominal attributes since the number of different values is limited. Usually the algorithm stores for each attribute the class-conditional distribution of the values. However, for continuous attributes this is not applicable due to the potentially infinite number of different values.
The VFDT was originally defined for nominal attributes only, but various proposals have been made to extend it for continuous numeric attributes (Pfahringer, Holmes, & Kirkby, 2008; Jin & Agrawal, 2003; Gama, Rocha, & Medas, 2003; Holmes, Richard, & Pfahringer, 2005). A common approach is to fit a probability distribution for each attribute instead of storing the raw samples to reduce the required memory. We use the default setting in the Massive Online Analysis framework (MOA) (Bifet, Holmes, Kirkby, & Pfahringer, 2010), which approximates attribute distributions with a single Gaussian. A predefined number of v split values (default is $v = 10$) is equidistantly distributed on the observed attribute range. This lightweight approach, initially introduced in (Pfahringer et al., 2008), is efficient and known to perform well in practice (Holmes, Richard, & Pfahringer, 2005).

Split-Delay vs. the Number of Split-Attempts

Ideally, the VFDT would test after each example whether the condition $\Delta G > \epsilon$ is fulfilled. However, as already mentioned by Domingos and Hulten (2000), the necessary recomputation of all information gains is a very time-consuming , and therefore, not applicable in practice. To reduce the load, the authors introduced the hyperparameter n_{min} to define the number of examples a leaf has to accumulate before a new *split-attempt* is done. Even though this periodic approach reduces the computational complexity for tree construction by a factor of n_{min}, it comes at the cost of an effectively slower tree growth and usually a lower performance, since more examples are accumulated than the strict minimum (Kirkby, 2007). Figure 3.8 shows some examples where the mentioned relation can be observed. We refer to the number of surplus examples as *split-delay*. The setting of n_{min} controls a trade-off between computational complexity and classification performance, or

more precisely, between the number of split-attempts and the split-delay.
In the following we fromally define the term split-delay. Given a leaf l, its true minimum split-time $\hat{t}_l$ and a split-time prediction function F which uses some leaf-specific data D_l to estimate the local split-time. D_l is obtained from the previously unsuccessful split-attempt and depends on the corresponding time, however we omit this detail in favor of a clear notation. Whenever the predicted time $F(D_l)$ is passed, a new split-attempt is performed. This process is repeated until l is split. The split-delay is given as

$$d_l = \sum_{i=1}^{b} F_i(D_l) - \hat{t}_l,$$

where b is the number of required split-attempts and $F_1 = n_{\min}$ is predefined. Hence, the initial split-attempt is always performed after n_{min} examples and the split-time prediction is only used if no split initially occurs. Particularly, the split-delay of nodes splitted at the initial attempt ($\hat{t}_l <= n_{\min}$) is the same for all split-time prediction functions. Please note, that tie-splits have by definition no split-delay, since leaves are always splitted as soon as the maximum number of examples is accumulated.
We determine the true minimum split-time $\hat{t}_l$ by using the VFDT with $n_{\min} = 1$, in other word it tries to split after each example, and has therefore by definition no split-delay. Concretely, we construct a reference tree which is associated with its set of L split-nodes at the end of learning. The average split-delay of a F is simply given as $\frac{1}{|L|} \sum_{l \in L} d_l$[1].

Split-time Prediction

The goal of split-time prediction is to minimize the split-delay and the number of split-attempts. In the VFDT algorithm, the parameter $n_{\min}$ globally controls the time interval between split-attempts without utilizing any information of the processed data. Split-time predictions use the local statistics of the leaves to increase the efficiency. The VFDT requires i.i.d. data for the Hoeffding bound to be valid. In such cases, the central limit theorem states that the more examples are accumulated within a leaf the more precise is the estimation of the true local class distribution, the two best attributes and the corresponding difference in information gain ΔG. Using such information, split-attempts can be avoided when successful split-attempts are unlikely or can be encouraged in the contrary case. These guarantees are not given in the case of concept drift, however, past or at least recent data has still some predictive value in these scenarios which in addition benefit from local adaptation.
Split-time predictions are only performed when the initial split-attempt fails (after $n_{\min}$ examples). From then on, we continuously predict the

1 The prediction function F is continuously applied during the construction of the reference tree to measure the split-delay.

local split-time. We bound the maximum predicted split-time with the number of examples required for a tie-split $n_{\max} = \frac{R^2 ln(1/\delta)}{2\tau^2}$. Algorithm 3.4 shows the pseudocode for constructing a VFDT with split-time prediction and the corresponding splitting-criterion is given in Algorithm 3.5.

Algorithm 3.4 VFDT with split-time prediction

Inputs:
S : stream
$n_{\min}$: time interval for split tests
Initialize:
Let Tr be a tree with a single leaf l_1 (the root)
Let $S(l_1)$ be the sufficient statistics l_1 collects
Let $C(l_1) := (\frac{1}{c}, \ldots, \frac{1}{c})$ be the initial class distribution of l_1
Let $\text{pt}(l_1) := -1$ be the predicted split-time for l_1
for all time steps t **do**
assign $(\mathbf{x}_t, y_t)$ to leaf l_i according to matching attributes
update $S(l_i)$ with $((x)_t, y_t)$
update $C(l_i)$ with y_t
if VFDT-split-time-prediction-split-criterion($S(l_i)$,$n_{\min}$,$\text{pt}(l_i)$) **then**
choose best split attribute j of l_i depending
on impurity($S(l_i)$)
split l_i along all discrete attribute values of j

Algorithm 3.5 Split criterion with split-time prediction of the VFDT

Inputs:
$S(l_i)$: statistics of leaf l_i
$n_{\min}$: time interval for split tests
$\text{pt}(l_i)$: predicted split-time for l_i
Output:
Boolean whether l_i should be split or not
if $|S(l_i)|$ mod $n_{\min} = 0$ and $\text{pt}(l_i) = -1$ or $|S(l_i)| = \text{pt}(l_i)$ **then**
determine the attributes j_1, j_2 with highest gain G_{j_1}, G_{j_2}
if ($\Delta G > \epsilon$ or $\Delta G < \tau$) **then**
return true
else
$\text{pt}(l_i)$:=min(split-time-prediction(), $n_{\max}$)
return false
else
return false

3.3.2 *Related Work*

Local split-time predictions have been only recently proposed by Garcia-Martin (2017). The authors assume that ΔG remains constant

between the two best split attributes and simply predict the number of necessary examples given by the Hoeffding bound

$$F_{\text{CGD}} = min\left(\frac{R^2 ln(1/\delta)}{2\Delta G^2}, n_{max}\right).$$

They showed that this intuitive and simple approach drastically reduces the run time but also diminishes the accuracy of the VFDT. In case of an increased ΔG, this method distinctly overestimates the split-time because the number of examples depends quadratically on ΔG. Our approach is more precise, since it incorporates more information for its prediction, namely the resulting class-distributions of the currently best split. Furthermore, its cautious predictions result in a substantially lower split-delay, providing the same classification performance as the original VFDT.

A lot of work has been done to improve the VFDT in different aspects. It was adapted to specifically deal with noisy datasets (Yang & Fong, 2011) as well as to one-class scenarios (Li, Zhang, & Li, 2009). Holmes, Kirkby, and Pfahringer (2005) pointed out that tie-splits, occurring when the best attribute is not determined within a maximum number of examples, are done very frequently and sometimes lead to skewed trees. They mitigated this issue by increasing stepwise the maximum number of required examples after each tie-split.

Gama et al. (2003) incorporated Naive Bayes (Domingos & Pazzani, 1997) classifiers within the leaves to gain more classification power, as proposed by Kohavi (1996) for batch trees. However, Holmes, Kirkby, and Pfahringer (2005) showed for various tasks that the initially more accurate Naive Bayes leaves are outperformed by majority voting in the long run. They applied a simple adaptive strategy, choosing the classification method which was more accurate in the past, to get the best of both worlds. Bifet, Holmes, Pfahringer, and Frank (2010) used the Perceptron algorithm (Rosenblatt, 1958) instead to reduce the computational load while maintaining the classification gain.

Even though the Hoeffding bound assumes i.i.d. data, a lot of publications adapted the VFDT for streams incorporating changes in the distribution. The first was the Concept-Adapting VFDT by Hulten et al. (2001), which grows alternative sub-trees as soon as there is evidence that previously chosen split-attributes are becoming inappropriate and replaces them when they are less accurate than the alternative ones. Bifet and Gavaldà (2009) generalized this approach and used explicit drift detection to initiate the growth of sub-trees and were able to achieve better results. The VFDT was also applied in ensemble methods, a very popular approach to deal with non-stationarity. Bifet, Holmes, and Pfahringer (2010) insert a drift detector in each single VFDT and reset the classifier when the detection fires. The resulting method called Leveraging Bagging performed well on numerous datasets.

Publications tackling the efficiency of the VFDT focused so far rather on the implementation instead of the core algorithm. A parallelized version of the VFDT called Vertical Hoeffding Tree performs

the attribute-specific tasks such as collecting statistics in the nodes as well as calculating the gain of corresponding splits in a distributed way (Kourtellis, Morales, Bifet, & Murdopo, 2016). The gained speed-up scales with the number of attributes and the amount of applied CPU cores. Recently, an optimized but sequential C++ implementation was introduced by Bifet et al. (2017), able to gain a distinct speed-up in comparison to the versions available in Very Fast Machine Learning (VFML), the original framework by Domingos and Hulten, and Massive Online Analysis (MOA) (Bifet, Holmes, Kirkby, & Pfahringer, 2010), a well established open source framework for data stream mining.

Our enhancement of the VFDT is independent from already published work, hence, it can be integrated in any of them, providing the same benefits as it does for the original algorithm. In particular, it can be combined with already optimized or parallel implementations (Kourtellis et al., 2016; Bifet et al., 2017) to increase the efficiency even further.

3.3.3 *Proposed Method: OSM*

One split-attempt implies the calculation of the information gains of all evaluated splits. In particularly, the class distribution after each evaluated split has to be determined. In case of an initially failed split-attempt, we reuse this information to predict the split-time. More precisely, on the basis of the class-distributions resulting from the currently best split, we approximate the minimum amount of required additional examples until the Hoeffding bound is met. Or put differently: assuming that the attributes j_1 and j_2 deliver the highest gain G_{j_1} and G_{j_2}, we estimate how many examples are at least necessary to sufficiently reduce G_{j_1} such that ΔG gets large enough. The estimation assumes that the class distribution resulting from the best split remains similar and that G_{j_2} is approximately constant. We term our method One-Sided Minimum (OSM). More specifically, the weighted entropy of the currently best split j_1 is given by $H_{j_1}(M)$. For simplicity, we consider only binary splits (the approach can be extended to multiway-splits in a straight-forward way) and use $H_{j_1}(M(a_1^{j_1}), M(a_2^{j_1}))$ (see equation 3.8) as alternative to denote the weighted entropy.

How many examples need to be added such that this weighted entropy becomes sufficiently small? The entropy is reduced most if examples are added to the most frequent class. For the weighted entropy, an additional degree of freedom is the choice between $M(a_1^{j_1})$ or $M(a_2^{j_1})$. An exact solution how to best distribute m novel samples needs to calculate 2^m possible solutions. A greedy approach which incrementally adds examples to the class distribution minimizing the weighted entropy the most, requires $2m$ calculations of the entropy. Instead, we propose to initially determine the lowest of both entropies $M(a_1^{j_1})$, $M(a_2^{j_1})$ and to incrementally add examples to this set until ΔG is large enough.

We argue in Section 3.3.3 that this approximation usually yields the same solution as the greedy search. The returned values are equal or larger than the exact solution which is sufficient for our purpose, since predicting the minimum is already a cautious approach. A naive implementation of this approach requires still m entropy calculations. However, the appealing property of this approach is that it can be stated as an equation and solved on the basis of root-finding algorithm. For this purpose we used the proven method of Brent (Brent, 1973), a root-finding algorithm which combines the bisection method, the secant method and inverse quadratic interpolation. In contrast to Newton's method, it does not require derivatives and converges superlinearly for well-behaved functions. Formally, we search a root of the function

$$f(x) = H_{j_2}(M) - H(E(M(a_1^{j_1}), x), M(a_2^{j_1})) - \sqrt{\frac{R^2 ln(1/\delta)}{2|M| + x}} \tag{3.9}$$

in the range $(0, n_{max}]$, where $E(M, x)$ is a function which increases the most frequent class of M by x samples. Thus, m is given as root of $f(x)$. In advance, we verify that $f(0)$ and $f(n_{max})$ have different signs to ensure a solution exists in the desired interval. In our experiments, the solution was usually found in few iterations (on average 12.8). This strategy is very cautious and predicts low values leading to unnecessary split-attempts when a split gets likely. Hence, we bound the minimum prediction by $n_{\min}$. The final prediction is given by

$$F_{\text{OSM}} \begin{cases} n_{max} & \text{if } f(0)f(n_{max}) > 0 \\ max(n_{\min}, m) & \text{otherwise.} \end{cases}$$

The complexity of our prediction function is dominated by the entropy calculations and therefore given as $\mathcal{O}(C)$. The *OSM* approach is in general pessimistic in its predictions, highly valuing a low split-delay in favor of reduced split-attempts. However, the number of split-attempts can be simply reduced by scaling its predictions for tasks where low split-delay are of secondary importance.

Theoretical Analysis

In the following, we show that *OSM* yields for the most part equivalent estimates as the greedy search. To keep the notation clear we define $M_1 = M(a_1^{j_1})$ and $M_2 = M(a_2^{j_1})$. Let us remind that the weighted entropy is given as

$$H(M_1, M_2) = \frac{|M_1|H(M_1) + |M_2|H(M_2)}{|M_1| + |M_2|}.$$

Adding one example to a class distribution has a dwindling effect on the class frequencies and consequently on the entropy. Hence, treating both single entropies as constants, we can determine the partial derivatives of $H(M_1, M_2)$, where with respect to the size of

each class distribution

$$\frac{\partial H(M_1, M_2)}{\partial |M_1|} = \frac{|M_2|(H(M_1) - H(M_2))}{(|M_1| + |M_2|)^2}$$
$$\frac{\partial H(M_1, M_2)}{\partial |M_2|} = \frac{|M_1|(H(M_2) - H(M_1))}{(|M_1| + |M_2|)^2}.$$

The sign of the gradient depends only on the difference between the single entropies $H(M_1) - H(M_2)$ and vice versa. Therefore, if the entropies were constant, the greedy search would yield the same solution as *OSM*, always adding examples to the class distribution having initially the lower entropy.
Taking the reduction of the entropies into account, we determine under which circumstances the solutions deviate. Assume $H(M_1) < H(M_2)$ and we iteratively add examples exclusively to M_1. Suppose we add one example to the most frequent class of M_1, leading to the new distribution M_1' and the corresponding entropy $H(M_1') = H(M_1) + \delta_1$, where $\delta_1 < 0$. Analogously for M_2 we get M_2' and $H(M_2') = H(M_2) + \delta_2$. The denominator of $H(M_1, M_2)$ is unaffected by the choice, thus our solution deviates from the greedy search when

$$\begin{aligned} |M_2|\delta_2 + H(M_2') &< |M_1|\delta_1 + H(M_1') \\ (|M_2| + 1)\delta_2 + H(M_2) &< (|M_1| + 1)\delta_1 + H(M_1) \\ (|M_2| + 1)\delta_2 &< (|M_1| + 1)\delta_1 + H(M_1) - H(M_2) \end{aligned}$$

$(|M_2| + 1)\delta_2$ has not only to be smaller than $(|M_1| + 1)\delta_1$, but it additionally has to make up for the entropy difference $H(M_1) - H(M_2)$, which is negative since we assume that $H(M_1) < H(M_2)$. Since δ_k decreases with increasing size of the distribution, the term $|M(a_k^{j_1})|\delta_k$ remains approximately constant. Hence, the only likely scenario that our method deviates from the greedy search is when initially $H(M_1) \approx H(M_2)$. Please note, that the greedy search unlikely switches between M_1 and M_2 because the entropy difference increases when additional examples are distributed.

3.3.4 *Experiments*

We used large data streams consisting of well known artificial and real-world benchmarks. Table 3.4 shows their main characteristics and we briefly describe them in the following. More information about the datasets is given in Section A.2.

Poker Randomly drawn poker hands are represented by five cards each encoded with its suit and rank. The class is the poker hand itself such as one pair, full house and so forth. We used the version presented by Bifet, Pfahringer, Read, and Holmes (2013), containing virtual drift via sorting the instances by rank and suit.

Random Tree A random decision tree is constructed by randomly splitting along the attributes as well as assigning random classes

Table 3.4: Various Characteristics of the considered datasets. Some real-world datasets are sorted (Poker) or shuffled (MNIST8M) and therefore it is clear whether they contain concept drift or not. In the case of Rialto and Conver Type, concept drift was detected based on the method described in Section 4.3.

Dataset	#Samples	#Feat.	#Class	Type	Concept drift
Poker	829K	10	10	artificial	Yes
Random Tree	5M	200	25	artificial	No
RBF	10M	100	50	artificial	No
LED-Drift	10M	24	10	artificial	Yes
Rialto	82K	27	10	real-world	Yes
Airline	539K	7	2	real-world	?
Cover Type	581K	54	7	real-world	Yes
MNIST8M	8.1M	782	10	real-world	No
HIGGS	11M	28	2	real-world	?

to each leaf. Numeric and nominal attributes are supported and the tree depth can be predefined. Instances are generated by uniform sampling along each attribute. This dataset is i.i.d. and was initially proposed by Domingos and Hulten (2000).

Radial Basis Function (RBF) Gaussian distributions with random initial positions, weights and standard deviations are generated in d-dimensional space. The weight controls the partitioning of the examples among the Gaussians. This dataset is i.i.d.

LED-Drift Each instance is reperesented by 24 boolean features with 17 of them being irrelevant. The remaining seven features correspond to segments of a seven-segment LED display. The goal is to predict the digit displayed on the LED display, where each feature has a 10% chance of being inverted. Drift is generated by swapping the relevant features with irrelevant ones.

Rialto Ten of the colorful buildings next to the famous Rialto bridge in Venice are encoded in a normalized 27-dimensional RGB histogram (Losing et al., 2018a). We obtained the images from time-lapse videos captured by a webcam with fixed position. The recordings cover 20 consecutive days during may-june 2016. Continuously changing weather and lighting conditions affect the representation.

Airline The task is to predict whether a given flight will be delayed or not based on seven attributes encoding various information on the scheduled departure. This dataset is often used to evaluate concept drift classifier and is temporally ordered.

Cover Type Cartographic variables such as elevation, slope, soil type, characterizing 30×30 meter cells, are assigned to different forest cover types. Only forests with minimal human-caused distur-

Table 3.5: The evaluated approaches and their abbreviations.

Abbr.	Method
VFDT	The standard VFDT with periodic split-attempts
CGD	VFDT with split-time prediction based on the gain difference (Garcia-Martin, 2017)
OSM	VFDT with split-time prediction based on the class distribution (proposal)

bances were used, so that resulting forest cover types are more a result of ecological processes.

MNIST-8M Loosli, Canu, and Bottou (2007) used pseudo-random deformations and translations to enlarge the well-known MNIST database (Lecun, Bottou, Bengio, & Haffner, 1998). The ten handwritten digits are encoded in 782 binary features. The dataset is already shuffled in its original publication and therefore contains no concept drift.

HIGGS This task consists of eleven million simulated particle collisions and was initially published by Baldi, Sadowski, and Whiteson (2014). The goal of this binary classification problem is to distinguish between a signal process producing Higgs bosons and a background process. The data consist of low-level kinematic features recorded as well as some derived high-level indicators.

The experiments are performed within the MOA framework and the split-time prediction approaches were integrated within its VFDT implementation. The classification is based on the majority class in the leaves and we used and the default parameters of the VFDT ($n_{\min} = 200$, $\tau = 0.05$, $\delta = 1e-7$) to provide comparable results to other publications. We investigate how our method performs in stationary and non-stationary environments by performing two experiments for each dataset. One uses the original order of the data, whereas the other shuffles the data to ensure stationarity. The datasets RBF, Random Tree and MNIST8M are originally shuffled, therefore, we report their result only in the shuffled experiments. Table 3.5 lists the algorithms that are used in the experiments. The split-delay of *CGD* is too large in comparison to the other methods[2], therefore, its uncompetitive results are neglected for the most part in benefit of reasonably scaled plots. However, we always report its average performances and include it in the crucial analysis of the classification error.

Split-delay versus split-attempts

In the following, we investigate the trade-off between the split-delay and the number of split-attempts achieved by the *VFDT* and our approach *OSM*. We generate Pareto fronts between the split-delay and

2 *CGD* has an average split-delay of 7600, *OSM* has a delay of 96.

Table 3.6: The number of split-attempts as well as average split-delays of *VFDT* and *OSM*. The results for the shuffled data are given at the top, whereas those using the original order are at the bottom. *OSM* requires always fewer attempts and provides similar split-delays. *CGD* has the smallest amount of attempts but also an intolerable high split delay (On average 701 attempts, 7675 split-delay [$n_{\min} = 200$]).

	$n_{min} = 50$				$n_{min} = 200$			
Dataset	#Attempts		∅Delay		#Attempts		∅Delay	
(shuffled)	*VFDT*	*OSM*	*VFDT*	*OSM*	*VFDT*	*OSM*	*VFDT*	*OSM*
Poker	16570	**1732**	**26**	27	4138	**1382**	**92**	96
Random Tree	69900	**13883**	**18**	29	17102	**7660**	**94**	98
RBF	170487	**1921**	**22**	28	42702	**1640**	**118**	125
LED-Drift	199933	**11327**	23	**20**	49897	**9687**	**111**	113
Rialto	1644	**273**	45	**45**	409	**235**	90	**50**
Airline	8904	**6665**	15	**13**	1821	**1811**	103	**103**
Cover type	11613	**1390**	32	**26**	2900	**1049**	86	**55**
MNIST8M	161975	**13283**	**20**	21	40420	**11859**	**104**	107
HIGGS	68966	**24725**	**24**	56	17564	**14889**	**103**	118
∅	78888	**8355**	**25**	**29**	**19661**	**5579**	100	**96**

	$n_{min} = 50$				$n_{min} = 200$			
Dataset	#Attempts		∅Delay		#Attempts		∅Delay	
(original)	*VFDT*	*OSM*	*VFDT*	*OSM*	*VFDT*	*OSM*	*VFDT*	*OSM*
Poker	16133	**2662**	**20**	132	3968	**1750**	**93**	155
LED-Drift	199927	**11154**	23	**21**	49891	**9528**	**111**	113
Rialto	1643	**243**	25	**5**	409	**211**	120	**120**
Airline	8467	**6631**	5	**5**	1618	**1618**	98	**98**
Cover type	11028	**1703**	**21**	1113	2727	**1098**	**100**	934
HIGGS	219624	**74730**	**22**	42	55992	**46559**	**88**	97
∅	76137	**16187**	**19**	219	19101	**10127**	**102**	252

the number of split-attempts by varying the parameter $n_{\min}$ in the range of $[50, 500]$ [3]. Figure 3.9 shows the corresponding results for various datasets, whereas Table 3.6 lists the values for all dataset with $n_{\min} \in \{50, 200\}$. The *VFDT* requires clearly more attempts particularly for low $n_{\min}$ values. Its average split-delay converges as expected around $\frac{n_{\min}}{2}$. Even though it is able to achieve an arbitrary low split-delay by definition, the computational cost increases drastically. In case of stationarity, the Pareto fronts of *OSM* are always more optimal than those of *VFDT*. *OSM* achieves a similar split-day with substantially fewer split-attempts. The discrepancy is especially high for low $n_{\min}$ values, where the number of split-attempts performed by *OSM* is hardly affected. The split-delay smoothly adapts to corresponding $n_{\min}$ values. This is confirmed by the average results given in Table 3.6.

3 Please note, that *OSM* still uses $n_{\min}$ as minimum prediction bound.

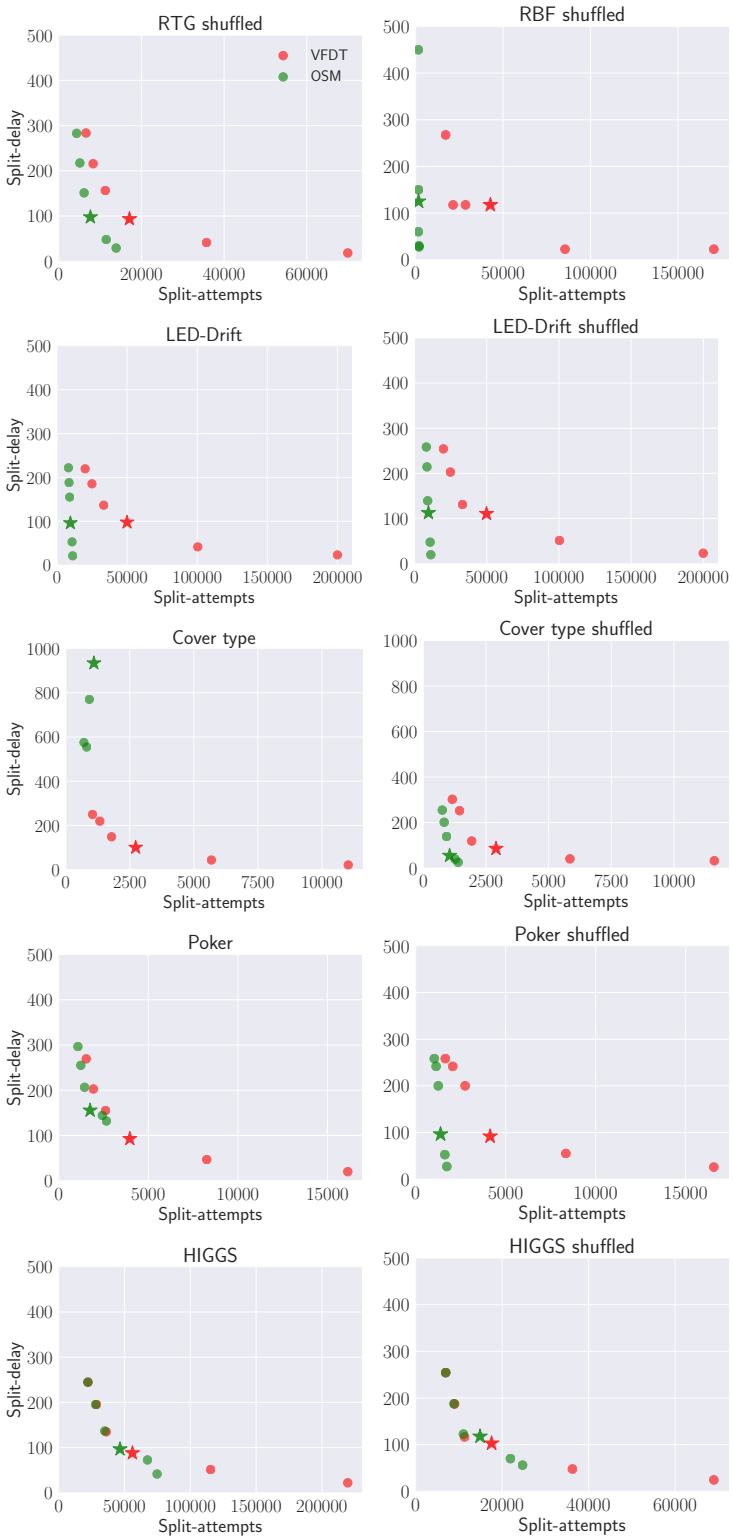

Figure 3.9: Trade-off between the number of split-attempts and the split-delay resulting from varying $n_{\min}$ in $[50, 500]$. A star highlights results which are achieved with the default setting of $n_{\min} = 200$. *OSM* also provides better results for data with concept drift (original order) with the exception of *Cover type*.

Table 3.7: Characteristics of the split-time distributions of VFDT ($n_{\min} = 200$). The average split-time varies due to the difficulty of the task, the number of classes as well as the amount of noise / overlap.

Dataset (original)	∅ Split-time	Bound-splits	Tie-splits
Poker	18.6K	12	21
Random Tree	3.2K	508	5
RBF	78.1K	2	102
LED-Drift	30.1K	41	180
Rialto	19.7K	2	1
Airline	1.9K	27	27
Cover type	19.1K	6	16
MNIST8M	34.1K	13	168
HIGGS	3.2K	50	2546

The split-delay of *OSM* is similar or even lower (for $n_{\min} = 200$), but the number of split-attempts is drastically reduced.
OSM also performs well in non-stationary environments with the exception of *Cover type*, where the split-delay is higher in comparison to *VFDT*. In contrast to all other datasets, it is not able to reduce the split-delay with lower values of $n_{\min}$. In this particularly case, the best split-attribute drastically changes, violating the assumptions of the approach. Hence, one weak point of the method is that it rarely overestimates the split-time in non-stationary environments.
The relative reduction of split-attempts by *OSM* varies depending on the dataset, mainly caused by the corresponding split-time distribution. A high average split-time increases the leeway for the prediction algorithms to gain an advantage by predicting times far into the future. Table 3.7 shows the split-time distributions as well as the number of bound- and tie-splits. The correlation between the reduction in split-attempts and the average split-time is quite pronounced, e.g. RBF versus HIGGS. The split-time depends on the data structure itself and the contained amount of noise / overlap. A high amount of noise requires more examples to confidently determine the best split-attribute, whereas splits are generated quickly for data that can easily be discriminated along few dimensions. Datasets with a high proportion of bound-splits and a low average split-time as Random Tree are rather easy because the best split-attributes are quickly determined. The opposite is the case for datasets with a high percentage of tie-splits such as RBF and HIGGS. However, the fact that RBF and HIGGS have still very different split-times is explained by the different amount of classes within the datasets. The number of examples the algorithm waits until a tie-split is performed depends on the number of classes within the leaf[4]. Hence, the split-time of tie-splits correlates with the number of classes in the corresponding dataset, making split-time predictions particularly appealing for multi-class classification tasks.

4 The range of the entropy depends on the number of classes ($R = \log c$).

Table 3.8: Top: Average error rate (%) and corresponding standard deviations obtained from 100 repetitions, each time reshuffling the data. The best results are marked in bold. Results which are significantly worse are underlined (p-value < 0.05). Bottom: Error rates achieved with the original order of the data. The significance test is not possible for the "drift setting", since VFDT is a deterministic algorithm and reshuffling the data removes the drift.

Dataset (shuffled)	*VFDT*	*CGD*	*OSM*
Poker	38.34(0.3)	38.32(0.3)	**38.26(0.4)**
Random Tree	15.32(0.2)	17.69(0.5)	**15.30(0.2)**
RBF	19.26(0.5)	19.34(0.5)	**19.23(0.5)**
LED-Drift	27.35(0.1)	27.46(0.1)	**27.34(0.1)**
Rialto	84.26(0.3)	86.11(0.6)	**84.26(0.3)**
Airline	39.57(0.3)	39.57(0.3)	**39.54(0.3)**
Cover type	33.80(0.2)	34.14(0.4)	**33.79(0.3)**
MNIST8M	40.53(0.5)	40.78(0.6)	**40.53(0.5)**
HIGGS	30.63(0.2)	30.63(0.2)	**30.59(0.2)**
∅	36.75	37.67	**36.73**

Dataset (original)	*VFDT*	*CGD*	*OSM*
Poker	**31.39**	41.2	31.69
LED-Drift	27.17	27.24	**27.17**
Rialto	**86.04**	86.36	86.18
Airline	38.76	38.78	**38.72**
Cover type	**30.27**	32.61	31.32
HIGGS	**30.3**	30.58	30.67
∅	**40.66**	42.8	40.96

The high proportion of tie-splits, which are supposed to rarely happen, for HIGGS and MNIST cast doubt on the guarantee of the VFDT to construct similar trees as batch algorithms.

Classification Error

The shuffling of the data enables an analysis for significant error rate differences between the approaches. Welch's t-test (Welch, 1947) was used, a two-sample location test allowing different variances between the samples. We performed 100 repetitions, each time reshuffling the data. Table 3.8 shows at the top the average error rates with corresponding standard deviations. Our split-time prediction achieves the best results for all shuffled datasets and is significantly better than *CGD* on half of them, underlining that *OSM* does not reduce the classification performance. The *CGD* approach performs significantly worse due to its frequent and severe overestimations of the split-time.

Obviously, the original order is changed when the data is shuffled, therefore the significance analysis could not be performed in this

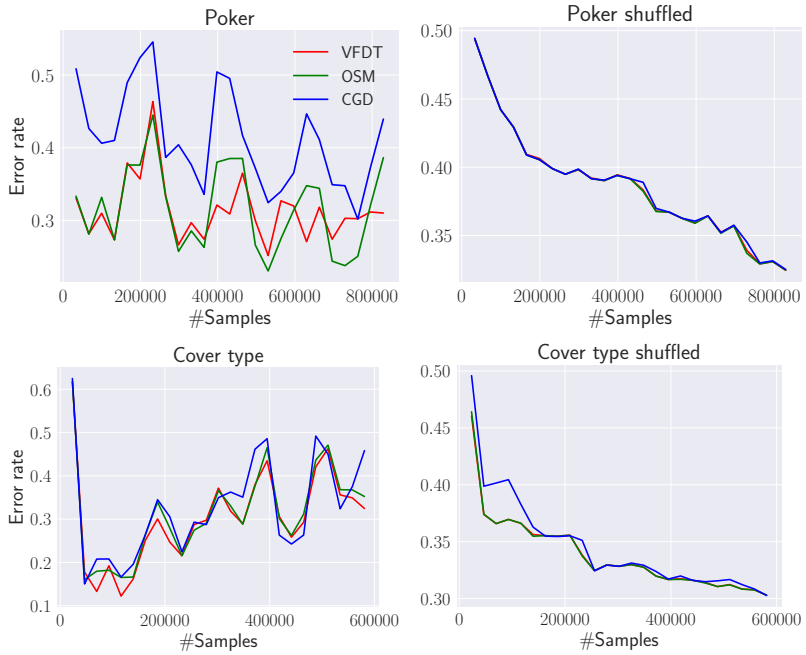

Figure 3.10: The error course of all methods for two exemplary datasets using the shuffled as well as the original order.

case. The results are given at the bottom of Table 3.8. Our method *OSM* delivers also in this case similar results to *VFDT*, proving its robustness in non-stationary environments. The increased split-delay for the Cover type data is only slightly affecting its error rate. The *CGD* method performs worse for each task, particularly in the case of the Poker data. The temporal course of the error rate is depicted in Figure 3.10 for some datasets. In case of non-shuffled data, *CGD* is performing worse than the normal VFDT, whereas *OSM* continuously delivers a similar classification performance.

Computation Time

The run time was measured on a cluster with 256 GB RAM and 40 cores, each with 2.6 GHz. We present only the time used by the VFDT algorithm, excluding processes such as reading the data from the hard drive, parsing the file and so forth. Table 3.9 depicts the time spend on the split-attempts as well as corresponding speedups of *OSM*. Our approach clearly reduces the attempt time for most datasets. The speedup is particularly high for the RBF and MNIST8M and is in general drastically increased for $n_{min} = 50$, highlighting the fact that *OSM* substantially scales better for low n_{min} values.
Table 3.10 shows the total time of the approaches and corresponding speedups achieved by our split-time prediction method. *OSM* reduces the overall run time for all datasets, in some cases as RBF and MNIST8M quite drastically. Figure 3.11 shows the exponentially growing total speedup for very low n_{min} values on the right. The development of the run time depending on the number of instances for

Table 3.9: The attempt-times as well as corresponding relative speedups achieved by *OSM*. Please note, that only the efficiency of this part of the algorithm has been improved. The speedup drastically increases with smaller n_{min} values because *OSM* is less dependent on this parameter, using it only as minimum prediction bound. *CGD* has the lowest average attempt time of 2.5 seconds and a speedup of 41 ($n_{min} = 200$).

	$n_{min} = 50$			$n_{\min} = 200$		
Dataset	Attempt time (s)		Speedup (×)	Attempt time (s)		Speedup (×)
(original)	*VFDT*	*OSM*	*OSM*	*VFDT*	*OSM*	*OSM*
Poker	1.25	**0.18**	6.94	0.35	**0.13**	2.69
Random Tree	590.87	**127.55**	4.63	153.57	**80.53**	1.91
RBF	374.19	**6.12**	61.14	92.87	**5.48**	16.95
LED-Drift	19.67	**1.37**	14.36	5	**1.05**	4.76
Rialto	0.62	**0.11**	5.64	0.15	**0.07**	2.14
Airline	1.98	**1.71**	1.16	0.49	**0.47**	1.04
Cover type	1.63	**0.21**	7.76	0.45	**0.19**	2.37
MNIST8M	2504.54	**205.31**	12.2	644.87	**191.69**	3.36
HIGGS	50.62	**18.19**	2.78	14.26	**11.58**	1.23
∅	393.93	**40.08**	12.96	101.33	**32.35**	4.05

Table 3.10: The runt-time of the whole learning algorithm as well as corresponding relative speedups achieved by *OSM*. The speedup is limited, since the run time of a sub-part is reduced. However, a clear run time reduction is achieved most of the times, particularly in case of $n_{min} = 50$. *CGD* has on average the lowest total time of 106 seconds and a speedup of 1.97 ($n_{min} = 200$).

	$n_{min} = 50$			$n_{min} = 200$		
Dataset	Total time (s)		Speedup (×)	Total time (s)		Speedup (×)
(original)	*VFDT*	*OSM*	*OSM*	*VFDT*	*OSM*	*OSM*
Poker	1.93	**0.79**	2.44	0.95	**0.71**	1.34
Random Tree	706.47	**246.80**	2.86	277.28	**206.56**	1.34
RBF	495.12	**114.30**	4.33	199.09	**91.71**	2.17
LED-Drift	35.18	**16.72**	2.10	19.4	**14.93**	1.30
Rialto	0.72	**0.21**	3.43	0.24	**0.16**	1.50
Airline	3.83	**3.49**	1.10	2.3	**2.29**	1.00
Cover type	3.40	**1.58**	2.15	2.01	**1.41**	1.43
MNIST8M	3097.27	**812.52**	3.81	1255.6	**794.97**	1.58
HIGGS	98.79	**65.5**	1.51	67.39	**61.92**	1.09
∅	493.63	**140.21**	2.64	202.7	**130.52**	1.42

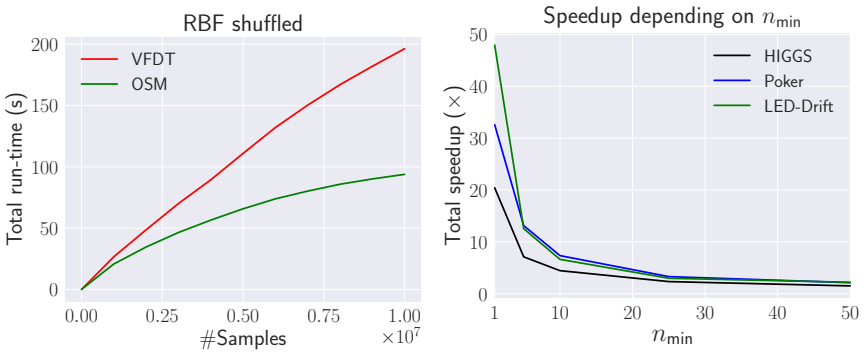

Figure 3.11: On the left: The course of the total run time for the RBF dataset. The run time is not increasing linearly because the accuracy increases over time, reducing the frequency of the splits ($n_{min} = 200$). On the right: The total speedup of *OSM* increases drastically for low n_{min} values.

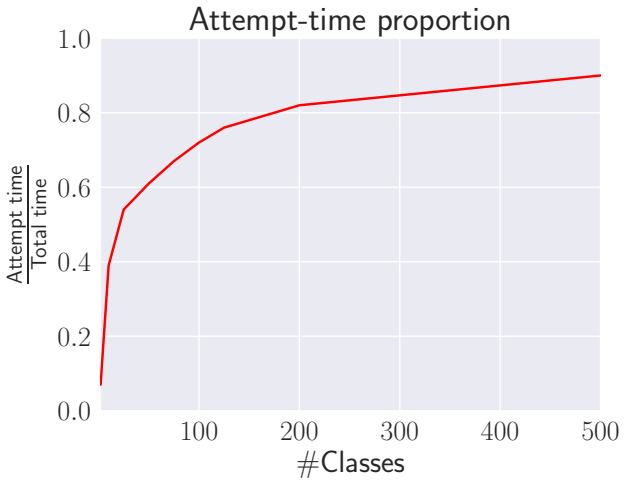

Figure 3.12: The attempt-time in comparison to the total depending on the number of classes in the RBF task. The relative proportion clearly increases with more classes in the dataset, making split-time prediction particularly valuable for multi-class problems.

the RBF dataset can be seen on the left of Figure 3.11. It can be seen that the speedup would be even higher for more instances.
Our approach optimizes only the efficiency of split-attempts, hence the potential overall-speedup is limited by the relative time of the split-attempts in comparison to the total time of the algorithm. Figure 3.12 illustrates on the right that the relative time increases with more classes[5], which is another reason why split-time predictions are particularly valuable for multi-class classification tasks.

5 The run time of split-attempts is dominated by the determination of the resulting class distributions for considered splits, requiring an iteration through all class-dependent feature distributions.

3.3.5 *Discussion*

In this section, the periodic split-attempt scheme of the VFDT was replaced by a prediction approach based on the local leaf statistics. On the basis of the resulting class-distributions from the best split, we approximate the minimum amount of required additional examples until the Hoeffding bound is met. The yielded approximations are for the most part equivalent to those of a greedy approach, but are obtained at a substantially reduced computational cost. The term *split-delay* was formally introduced as a precise measurement for evaluating the accuracy of split-time predictions. The experiments confirmed a distinct speed-up in comparison to the default VFDT, particularly for multi-class classification tasks. Moreover, the method significantly reduced dependence of the run time on the parameter n_{min}, leading to an increased overall robustness of the algorithm. Local split-time predictions are independent from each other and can easily be integrated in parallelized implementations to increase the efficiency even further. They constitute one example how to efficiently design a solution for the question when to increase the model complexity for incremental tree induction algorithms.

3.4 A PRACTICE-ORIENTED SURVEY

An algorithm has to be chosen according to the preconditions of a given task because there cannot exist a method which optimally performs in every scenario (Wolpert, 2002). Different interesting incremental learning algorithms have been published so far with various strengths and weaknesses. However, there are only a few sources providing information when to use them, since basically no comparative in-depth study, experimentally comparing the most popular methods according to the most relevant criteria, is available. Querying the literature usually leads to the original publications of considered algorithms which help only to some extent due to the following reasons:
Authors are naturally focused to demonstrate the merits of their method and apply them often in the settings they have been designed for. Furthermore, the evaluation usually considers only one or two other methods, leading to the conclusion that original publications often provide a limited overall picture of the algorithm's qualities. But even if one takes on the great effort to reproduce the results in the attempt to form an own opinion, it often turns out to be impossible because of proprietary datasets or unknown hyperparameters settings. In the end, either a method is picked based on the own experience, which usually comprises only a fraction of available algorithms, or a lot of resources are invested to try out several approaches.

In this section, we will mitigate the mentioned issue by analyzing the core attributes of eight popular methods. The study aims for a fundamental comparison of the algorithmic overall performance. The performance for specific settings, such as strictly limited hardware

requiring very efficient learning or tasks with abundant resources where the accuracy has highest priority, can be inferred from the rich results provided here. The choice of an algorithm is guided using basic information (e.g. number of dimensions / samples), usually available a priori[6]. The methods are evaluated in offline- and online setting, enabling a precise comparison in terms of classification performance, learning speed and model complexity. Thereby, diverse datasets are used to assess strengths and weaknesses of respective methods and provide guidance on their applicability for specific tasks. We also examine the methods in respect to hyperparameter optimization (HPO), a necessary and crucial process for most applications, which is however often neglected in the literature, leading sometimes to unpleasant surprises in practice. Our study primarily focuses on stationary environments, which is mirrored by the picked datasets and methods. Even though we briefly evaluate and discuss the methods in the context of changing environments, a thorough treatment of the so-called concept drift takes place within an own chapter (Chapter 4).

3.4.1 *Foundation*

The comparison covers a broad range of algorithm families. Bayesian, linear, and instance-based models as well as tree ensembles and neural networks are represented. Model-dependent methods such as the Incremental Support Vector Machine (Cauwenberghs & Poggio, 2001) are denoted by an acronym (ISVM), whereas meta algorithms such as Stochastic Gradient Descent, which can be combined with different forms of classification prescription, are denoted by an acronym with additional index (SGD_{Lin}), specifying the applied model. In the following, the evaluated methods are briefly described.

Incremental Support Vector Machine (ISVM) is one of the most popular exact incremental version of the SVM and was introduced by Cauwenberghs and Poggio (2001). Apart from the set of support vectors, a limited number of examples, so called "candidate vectors", is maintained. These are the recently processed samples that are most likely to be promoted to support vectors in future. The smaller the set of candidate vectors is, the higher is the probability of missing potential support vectors. The ISVM is a *lossless* algorithm - it generates the same model as the corresponding batch algorithm - if the set of candidate vectors contains all previously seen data. Various recent applications of the ISVM can be found in the literature (Biggio et al., 2014; Lu, Boukharouba, Boonært, Fleury, & Lecoeuche, 2014).

LASVM is an online approximate SVM solver and was proposed by Bordes, Ertekin, Weston, and Bottou (2005). In an alternative manner,

6 The number of dimensions as well as the amount of incoming data examples can commonly be estimated. Furthermore, it can be inferred how crucial a quick reaction of the system is. For some tasks it is even possible to guess whether a linear classifier is sufficient (e.g. text classification).

the currently processed example is examined whether it constitutes a support vector and support vectors that possibly became obsolete are removed. Both steps are based on sequential direction searches as it is also done in the Sequential Minimal Optimization (SMO) algorithm (Platt, 1998). In contrast to the ISVM, there is no set of candidate vectors and only the current sample is considered as possible support vector. The resulting approximate solution significantly reduces the training time. LASVM was recently applied by Hsieh, Si, and Dhillon (2014) as well as by Cai, Wen, Lei, Vasconcelos, and Li (2014).

Online Random Forest (ORF) is an incremental version of the Random Forest algorithm (Saffari et al., 2009). A predefined number of trees grows continuously by adding splits whenever enough samples are gathered within one leaf. Instead of computing locally optimal splits, a predefined number of random values are tested according to the scheme of Extreme Random Trees (Geurts, Ernst, & Wehenkel, 2006). The split value optimizing the Gini index the most is selected. Tree ensembles are very popular, due to their high accuracy, simplicity and parallelization capability. Furthermore, they are insensitive to feature scaling and can be easily applied in practice. This method has been lately used by Lakshminarayanan, Roy, and Teh (2014) and by Pernici and Del Bimbo (2014).

Incremental Learning Vector Quantization (ILVQ) is an adaptation of the static Generalized Learning Vector Quantization (GLVQ) to a dynamically growing model, which inserts new prototypes when necessary. The insertion rate is guided by the number of misclassified samples. We use the architecture introduced by Losing et al. (2015) (see Section 3.2), which utilizes the prototype placement strategy COSMOS, minimizing the loss on a sliding window of recent samples. Metric learning can also be applied to extend the classification abilities further (Schneider et al., 2009; Bunte et al., 2012).

Learn++ ($\mathbf{LPP_{CART}}$) processes incoming samples in chunks with a predefined size. For each chunk an ensemble of base classifiers is trained and combined through weighted majority voting to an "ensemble of ensembles" (Polikar et al., 2001). Similar to the AdaBoost (Freund et al., 1999) algorithm, each classifier is trained with a subset of chunk examples drawn according to a distribution, ensuring a higher sample probability for misclassified inputs. LPP is a model independent algorithm and several different base classifiers such as SVM, Classification and Regression Trees (Breiman, Friedman, Olshen, & Stone, 1984) (CART) and Multilayer Perceptron (Rumelhart et al., 1985) have been successfully applied by the authors. As the original author we employ the popular CART as base classifiers. Chunk-wise trained models inherently incorporate an adaption delay depending on the chunk size. The chunk-size parameter constitutes a static way to approach the stability-plasticity dilemma. De-la-Torre, Granger, Radtke, Sabourin, and Gorodnichy (2015) and García Molina et al. (2014) applied this algorithm recently.

Incremental Extreme Learning Machine (IELM) reformulates the batch ELM least-squares solution into a sequential scheme (Liang et al., 2006). As the batch version, it drastically reduces the training complexity by randomizing the input weights. The network is static and the number of hidden neurons has to be predefined. This method is able to process the data one-by-one or in chunks, which significantly reduces the overall processing time. However, a valid initialization of the output weights requires at the beginning a chunk size that is equal or larger than the number of hidden neurons. Recent applications can be found in (Tang, Deng, & Huang, 2016; Tang, Deng, Huang, & Zhao, 2015).

Naive Bayes ($\mathbf{NB_{Gauss}}$) Let $\mathbf{x} \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ be an instance of discrete values represented as a vector $\mathbf{x} = (x^1, \ldots, x^n)$. The Naive Bayes classifier assigns to this instance a probability for each class $c_k \in \{c_1, \ldots, c_C\}$

$$P(c_k) = p(c_k) \prod_{i=}^{n} p(x^i | c_k).$$

Thereby, it assumes x^i to be conditional independent from every other feature x^j for $j \neq i$ given the class category c_k. Usually, $p(x^i|c_k)$ is simply determined by frequency counts in the data. This algorithm is lossless because it can be implemented in an incremental way.

In case of continuous attributes, the values can either be discretized or the likelihood can be assumed to follow a distribution, e.g. being Gaussian:

$$p(x^i|c_k) = \frac{1}{2\pi\sigma_k^2} \exp(-\frac{(x^i - \mu_k)^2}{2\sigma_k^2}),$$

where μ_k and σ_k are estimated on the training data.

The sparse model allows a very efficient learning in terms of processing time and memory requirements. This algorithm learns efficiently from few training examples (Salperwyck & Lemaire, 2011) and has been successfully applied in real-world situations such as Spam filtering and document classification[7] (Metsis, Androutsopoulos, & Paliouras, 2006; Ting, Ip, & Tsang, 2011). The major drawbacks of this algorithm are the conditional independence assumption of the features as well as its inability to handle multimodal distributions. This method was recently used in (Lou et al., 2014; Griffis, Allendorfer, & P. Szaflarski, 2016). In our experiments, we rely exclusively on NB on the basis of the Gaussian likelihood estimation.

Stochastic Gradient Descent ($\mathbf{SGD_{Lin}}$)

is an efficient optimization method for learning a discriminative model by minimizing a loss function such as the Hinge - or Logistic loss. We use SGD to learn a linear model by minimizing the Hinge loss function.

7 In the context of features based on text, the Naive Bayes algorithm is usually applied with the multinomial or Bernoulli event model.

Given a linear model with the weight vector $\mathbf{w}_i = (w_i^1, \ldots, w_i^n)$ at time step i and the learning rate λ, the update of the weights is given by

$$w_{i+1} = w_i - \lambda \nabla \mathcal{L}(\hat{y}_i, y_i).$$

SGD was recently revived in the context of large-scale learning (Zhang, 2004b; Bottou, 2010; Richtárik & Takáč, 2016). In combination with linear models, it performs especially well for sparse, high-dimensional data as often encountered in the domain of text classification or natural language processing. However, linear models are a misfit whenever non-linear class boundaries are required, which is particularly often the case for low-dimensional data. Recently, Akata, Perronnin, Harchaoui, and Schmid (2014) as well as Sapienza, Cuzzolin, and Torr (2014) applied this method.

Even though new versions of the mentioned algorithms are continuously proposed, we argue that the chosen methods reflect the general properties of the respective family. Therefore, conclusions of this survey are commonly applicable for current and upcoming variations of the corresponding algorithm. This is particularly highlighted by both SVMs which perform very similar with the difference that LASVM is able to process slightly larger datasets due to its approximate nature. However, both share the same drawbacks regarding large or noisy datasets, which also applies for the recently proposed extension of LASVM by Ertekin, Bottou, and Giles (2011), albeit in a slightly weaker degree because a mechanism to reduce the number of support vectors is introduced. Various extensions for the LPP and the IELM algorithm have been proposed (Elwell & Polikar, 2011; Ditzler & Polikar, 2013; Zhao, Wang, & Park, 2012; Ye, Squartini, & Piazza, 2013). Most of them are tackling non-stationary environments by introducing forgetting mechanisms. However, the major focus in this study lies on incremental learning in stationary environments where forgetting is rather harmful and deteriorates the performance. Furthermore, the basic principle of the algorithms and the corresponding advantages and disadvantages remain. In case of LPP, it is the flexibility of arbitrary base classifiers on the one hand, and the limited knowledge integration across chunks on the other. Methods for speeding up the convergence of SGD were presented by Bottou (2010) as well as by Johnson and Zhang (2013). However, the results obtained by the SGD algorithm in our experiments are not due to a slow learning speed of the SGD algorithm, but rather highlight the general benefits and limitations of linear models, such as a low model complexity and linear class boundaries.

3.4.2 *Related Work*

Numerous incremental algorithms have been published, often adapting existing batch methods to the incremental setting (Cauwenberghs & Poggio, 2001; Liang et al., 2006). Massive theoretical work has been done to evaluate their generalization ability and convergence speed in the stationary setting, often accompanied by assumptions such as

linearly separable data (Cesa-Bianchi & Lugosi, 2006; Watkin, Rau, & Biehl, 1993; Novikoff, 1962).

Although the field of incremental- / online learning is well established and particularly employed in the context of Big Data or the Internet of Things technology (Atzori, Iera, & Morabito, 2010), there are only a few publications targeting the field in a general way. Most of these are surveys describing available methods and viable application domains (Ade & Deshmukh, 2013; Joshi & Kulkarni, 2012).

Giraud-Carrier (2000) give some motivation for incremental learning and define the notion of incrementality for learning tasks. They argue in favor of applying incremental learning methods for incremental tasks, but also point to arising issues such as ordering effects or the question of trustworthiness.

One survey was recently published by Gepperth and Hammer (2016). They formalize incremental learning in general and discuss theoretical as well as arising challenges in practice. Furthermore, an overview of commonly employed algorithms with corresponding real-world applications is given.

Incremental learning is more frequently treated in the setting of streaming scenarios (Gaber, Zaslavsky, & Krishnaswamy, 2005; Aggarwal, 2014), although most of the work particularly targets concept drift (Žliobaite, 2010; Gama, Žliobaite, Bifet, Pechenizkiy, & Bouchachia, 2014; Ditzler, Roveri, Alippi, & Polikar, 2015). Domingos and Hulten (2003) define key properties for incremental algorithms which are required to keep up with the rapidly increasing rate of data output. They stress the necessity of combining models, strictly limited in terms of processing time and space, with theoretical performance guarantees.

Publications with a practical focus are very rare. One of them was done by Read, Bifet, Pfahringer, and Holmes (2012) in the setting of concept drift. Batch-incremental methods, these add batch classifier in an incremental way, e.g. LPP, are compared against instance-incremental approaches, truly incremental methods that are updated after each example such as incremental decision trees. The authors reach the conclusion that instance-incremental algorithms are equally accurate but use fewer resources and that lazy methods with a sliding window perform exceptionally well.

A massive study comprising the evaluation of 179 batch classifier on 121 datasets was done by Fernández-Delgado et al. (2014). This quantitative study considered also different implementations in varying languages and toolboxes. The best result was achieved by the Random Forest (Breiman, 2001) algorithm closely followed by the Support Vector Machine (SVM) (Cortes & Vapnik, 1995) with Gaussian kernel.

However, such work is still sorely missed for incremental algorithms. We pursue a more qualitative approach and instead of a massive comparison, provide an in depth evaluation of the major approaches within stationary environments. Next to the classification performance, we also inspect the model complexity which allows an inference of required resources in terms of time and space. The consid-

eration of rather neglected aspects such as learning speed and HPO rounds off our analysis.

3.4.3 *Datasets and Implementations*

In case of SGD_{Lin} and NB_{Gauss}, the implementations of the Scikit-learn package were used (Pedregosa et al., 2011). Implementations for the remaining algorithms are derived from code of the respective authors. Only publicly available datasets (Dheeru & Karra Taniskidou, 2017; Chang & Lin, 2011), predefining a fixed train-test-split, were used to enable reproducibility and comparability of our results. Table 3.11 gives the main attributes of the selected datasets. Artificial and

Table 3.11: The evaluated datasets and their characteristics.

Dataset	#Train instances	#Test instances	#Features	#Classes
Border	4000	1000	2	3
Overlap	3960	990	2	4
Letter	16000	4000	16	26
SUSY	4500000	500000	18	2
Outdoor	2600	1400	21	40
COIL	1800	5400	21	100
DNA	1400	1186	180	3
USPS	7291	2007	256	10
Isolet	6238	1559	617	26
MNist	60000	10000	784	10
Gisette	6000	1000	5000	2

real-world problems are included, differing widely in the number of classes, instances and dimensions. Even though the largest dataset has about 4.5 million instances, our evaluation does not specifically target learning from big data. Instead, our focus is the practical evaluation of incremental learning algorithms in terms of different key properties. Datasets which were not yet introduced are described in the following. More details to all datasets are given in Section A.2.

Letter The objective is to categorize black-and-white images, represented by 16 primitive numerical attributes, into one of the 26 capital letters of the English alphabet (Frey & Slate, 1991). The character images were based on 20 different fonts and each letter was randomly distorted.

SUSY The objective is to classify whether simulated particle collisions lead to the generation of new supersymmetric particles or not and was initially published in (Baldi et al., 2014). The data has been produced using Monte Carlo simulations. Each instance consists of eight kinematic features and ten high-level features.

Outdoor We obtained this dataset from images recorded by a mobile robot in a garden environment (Losing et al., 2015). The task is to classify 40 different objects, each approached ten times under

varying lighting conditions affecting the color based representation. The objects are encoded in a normalized 21-dimensional RG-Chromaticity histogram.

COIL Each of the 100 different objects is depicted in 72 different views in 128×128 pixel RGB images. These are encoded within a 21-dimensional RG-Chromaticity space (Jain & Li, 2005).

DNA The objective of this challenge is to categorize a given DNA sequence into the three classes: exon/intron boundaries (EI), intron/exon boundaries (IE), neither of them.

USPS This dataset consist of grayscale images with 16×16 pixels. The images contain handwritten digits and the goal is to categorize them into the corresponding ten classes.

Isolet The objective of this challenge is to recognize the spoken letters of the alphabet based on common features from the audio processing domain. The recording was done based on 150 different subjects.

MNIST This is one of the most popular databases of handwritten digits. Digits are size normalized and centered within grayscale images of 28×28 pixels. The objective is to classify the images into the ten digits.

Gisette This modification of the MNIST database was especially designed for a feature selection challenge. Changes include a randomized order of the pixels as well as the addition of noisy dimensions. The scope was reduced to a binary classification task between the often confused numbers of 4 and 9.

3.4.4 *Hyperparameter Optimization*

The process of model selection varies in complexity depending on the parameter- amount and type. Table 3.12 lists for each method the relevant hyperparameters. Parameters which are related to the feature scale, such as the learning rate or kernel widths are usually very crucial, since they strongly affect the classification performance and the model complexity. For instance, an inappropriately chosen kernel width drastically increases the number of support vectors. A wrongly set learning rate within the ILVQ leads to more errors during training and leads to more inserted prototypes.

Some models allow to control the speed of model expansion directly such as the ILVQ and ORF. This does not only affect the model complexity, but also influences learning speed and classification error and may lead to overfitting when a too aggressive growth is set. Rather uncritical are parameters increasing the leeway of an algorithm. Larger values are in this case always beneficial for the performance and only limited by the amount of available resources. The number of trees of

	Hyperparameter	Task independent
SVMs	Kernel function	✓
	RBF kernel width	✗
	Regularization	✗
	# Candidate vectors (only ISVM)	✓
ORF	Growing speed	✗
	# Evaluated random splits	✓
	# Trees	✓
ILVQ	Learning rate	✗
	Growing speed	✗
	Window size	✓
	(Metric learning rate)	(✗)
LPP_{CART}	Chunk size	✗
	# Base classifier per chunk	✓
	(Parameter of base classifier)	(✗)
IELM	Activation function	✓
	# Hidden nodes	✗
NB_{Gauss}	None	
SGD_{Lin}	Loss function	✓
	Learning rate	✗

Table 3.12: Relevant hyperparameters of the considered algorithms. Most critical parameters are those which cannot be robustly chosen, but require a task specific setting.

the ORF or the window size of the ILVQ are such parameters. Generally speaking, tree based models are easy to tune and perform usually well out of the box, whereas scale sensitive models such as ISVM, LASVM or ILVQ require an accurate, dataset dependent configuration of multiple parameters to deliver competitive results.

Both SVM algorithms are solely paired with the RBF kernel. We use the metric learning of ILVQ only for datasets with up to 250 dimensions (the distance calculations using the metric is quadratic in the number of dimensions and hence not feasible for very high dimensional data). The NB_{Gauss} algorithm is parameterless, hence no tuning is required at all. We minimize the hinge loss function with SGD_{Lin} and adjust only the learning rate. LPP_{CART} requires the number of base classfier per chunk as well as the parameters of the base classifier itself (non-parametric Classification and Regression Trees in our case).

All parameter are set by Hyperopt using the Tree of Parzen Estimators (Bergstra, Komer, Eliasmith, Yamins, & Cox, 2015; Bergstra, Bardenet, Bengio, & Kégl, 2011). Each parameter is individually adjusted by performing 250 iterations of a 3-fold cross validation using only the training data.

3.4.5 *Measure of Model Complexity*

We measure the model complexity in terms of the number of parameters that are required for model representation, enabling a comparison with respect to memory consumption. However, some models are fundamentally different such that this measure, even though there is some correlation, should not generally be equated with training- or run-time. We rather use this measure as an indicator to decide whether an algorithm struggles (unreasonable high amount of parameters) or is especially suited (sparse representation paired with high classification performance) for a given task.

3.4.6 *Evaluation Settings*

Algorithms are evaluated on basis of three different scenarios as it is illustrated by Figure 3.13. In the first setting, we apply the classical offline scheme and use the complete training set for the HPO. This allows a conclusion about the generalization ability of the final model. However, the usage of the whole training set for the HPO is usually not possible in practical applications and contradicts the paradigm of incremental learning. Hence, in the second setting, we optimize parameters only with a small proportion of training examples. This is not only closer to practice, but also, in combination with the results of the first setting, enables to infer whether the hyperparameters can be reliably estimated on a subset of the data. Since the number of training examples vary considerably across the datasets, we decided to use a relative proportion bounded by a maximum number. Precisely, we use 20% of the training data for HPO but never more than 1000 examples. The third setting applies the hyperparameters obtained within the second setting, but examines methods in online scheme. Here, we draw conclusions about the learning curves of the respective algorithms. To keep the number of training instances similar among all three evaluation scenarios, only the training set is processed as stream in online scheme (third setting), whereas the test set is ignored. The initial examples that are used for HPO are excluded.

3.4.7 *Experiments*

The evaluation of LASVM, NB_{Gauss}, ORF and SGD_{Lin} is straightforward, since these access consecutively only the current training example. But methods such as ISVM and ILVQ store additionally a window of recent samples or require chunk-wise training, as LPP_{CART} and IELM[8] do. In both cases, results depend on the window-/chunk size. Therefore, we treated the window-/chunk size as another hyperparameter and used once again Hyperopt to find the best value. We allowed

8 IELM requires for the initialization at least as many samples as it has hidden neurons but afterwards it can be updated after each sample.

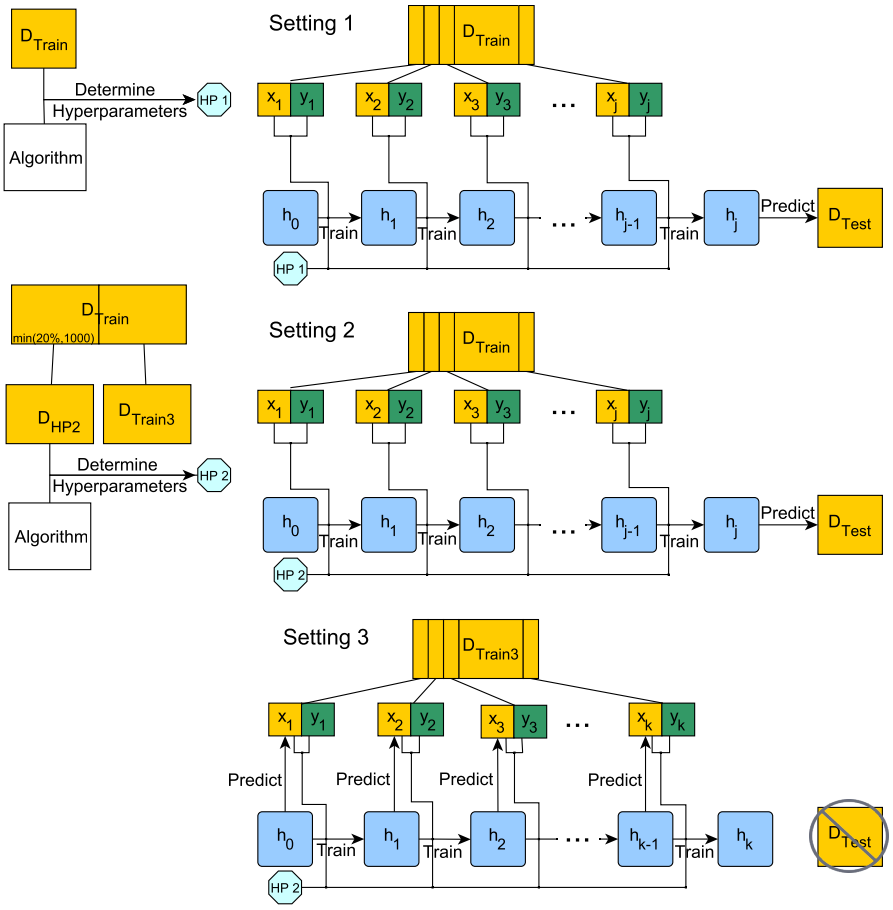

Figure 3.13: In the first setting, hyperparameters are optimized on basis of the whole training set. The evaluation is done according to the offline scheme where the test error is determined on a test set after initial training. The second setting uses a small subset of the training set for HPO. In the third setting, the same hyperparameters are used as in the second one, but instead of the TE, algorithms are evaluated based on the ITTE determined via online scheme. Thereby, only the training set is used and samples that were used during HPO are excluded.

a maximum size of 500 samples. All methods were trained single-pass, in the same order after initial shuffling.

Offline Setting - HPO with all training samples

Table 3.13 shows the test error and corresponding model complexities at the end of training. Both SVMs achieve on average the lowest error rates, often with a large margin, but at the expense of having by far the most complex models. The large amount of parameters is partly due to the fact that the model is designed to discriminate two classes, resulting in multiple SVMs to perform schemes such as one vs. all. Another reason is the linear growth of support vectors with the amount of samples. The model gets exceedingly complex for noisy or overlapping datasets such as *Isolet* or *Overlap*.

	Test error								Complexity							
Setting 1	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Border	**0.6**	1.6	2.4	3.2	3.5	4.0	64.5	3.6	797	251	3.7k	193	1.9k	750	**9**	12
Overlap	17.8	19.0	**16.3**	17.0	18.3	16.6	32.4	33.6	10k	7.2k	2.3k	235	1.9k	900	**12**	16
Letter	3.0	**2.9**	6.8	6.1	13.0	30.0	43.6	36.6	131k	140k	168k	14k	51k	8.4k	**442**	832
SUSY	DNF	DNF	**20.5**	21.0	DNF	DNF	21.3	26.5	DNF	DNF	86.0	51.2	DNF	DNF	**19**	72
Outdoor	29.1	**28.6**	29.0	33.1	31.5	29.1	76.2	29.5	43k	42k	8.8k	11k	6.2k	12k	**880**	1.7k
COIL	**3.5**	6.8	7.1	5.7	10.8	8.5	87.6	10.0	58k	93k	61k	18k	9.2k	36k	**2.2k**	4.2k
DNA	**5.1**	**5.1**	10.4	7.9	9.5	11.2	7.0	11.6	237k	190k	5.0k	33k	1.6k	55k	**543**	1.1k
USPS	4.6	**4.4**	7.5	8.6	9.7	7.9	11.0	24.6	710k	520k	33k	15k	9.6k	106k	**2.6k**	5.1k
Isolet	3.8	**3.3**	7.5	8.0	10.0	8.1	8.5	19.9	2.8M	3.4M	31k	21k	**12.0k**	322k	16k	32k
MNist	DNF	**1.5**	5.7	5.2	7.6	10.9	14.0	44.4	DNF	14M	111k	315k	73k	397k	**7.9k**	16k
Gisette	**2.0**	2.1	5.4	7.0	5.8	8.6	6.9	24.6	7.0M	6.2M	4.7k	263k	**2.5k**	2.5M	5.0k	20k
∅	7.7	**7.6**	10.8	11.1	12.0	13.5	34.0	25.0	1.2M	2.5M	54k	64k	17k	344k	**3.2k**	7.3k
∅ rank	**1.9**	2.0	3.0	3.7	5.3	5.3	6.3	6.8	7.1	7.0	5.1	4.0	3.4	5.3	**1.3**	2.5
∅ $\widehat{\text{rank}}$	2.7	**2.3**	3.0	3.7	5.3	5.3	6.3	6.8	7.0	6.8	5.1	4.0	3.5	5.3	**1.3**	2.5

Table 3.13: Test error (left) and model complexity (right) after training, measured by the overall number of parameters, averaged over 10 repetitions. In the first setting, the Hyperparameters were optimized using the whole training data. The processing was canceled whenever it took longer than 24 hours and corresponding experiments are denoted as DNF. We calculated two different rankings. The first is the average rank based on all datasets the algorithms were able to deliver a result. The second rank ($\widehat{\text{rank}}$), however, punishes algorithms with DNF entries. In this case, they are ranked as the last in the respective dataset.

	Test error								Complexity							
Setting 2	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Border	**0.6**	1.2	3.7	2.9	3.6	3.8	66.3	3.6	1.1k	1.5k	8.0k	286	2.6k	875	**9**	12
Overlap	18.2	19.3	18.8	17.9	19.0	**17.8**	31.6	33.6	10.9	9.2 k	31k	496	1.9k	546	**12**	16
Letter	3.6	**2.5**	7.8	4.9	15.1	32.6	46.1	36.6	163	173k	473k	11k	70 k	6.5k	**442**	832
SUSY	DNF	DNF	**20.7**	26.6	DNF	DNF	23.0	26.5	DNF	DNF	1.5M	453k	DNF	DNF	**19**	72
Outdoor	**28.1**	31.0	42.4	36.8	30.6	28.4	74.3	29.5	40k	53k	925	11 k	7.3k	8.3k	**880**	1.7k
COIL	**4.4**	7.5	10.1	6.4	11.9	16.4	87.9	10.0	180k	113k	14k	17 k	6.1k	15k	**2.2k**	4.2k
DNA	**4.8**	4.9	11.8	9.2	8.0	13.3	6.7	11.6	333k	327k	500	42k	1.8k	76k	**543**	1.1k
USPS	**4.4**	5.3	8.6	8.6	11.1	10.3	10.2	24.6	744k	978k	61k	91k	12k	50k	**2.6k**	5.1k
Isolet	**3.5**	3.6	8.0	10.6	9.3	11.5	10.4	19.9	6.2M	4.2M	123k	67k	**14k**	159k	16k	32k
MNist	DNF	**1.9**	5.0	5.7	8.7	15.8	12.3	44.4	DNF	16M	253k	1.2M	162k	169k	**7.9k**	16k
Gisette	2.2	**2.1**	7.3	5.9	6.0	16.2	4.6	24.6	7.0M	6.7M	16k	470k	**3.0k**	1.0M	5.0k	20k
∅	**7.8**	7.9	13.1	12.3	12.3	16.6	33.9	25.0	1.6M	2.9M	226k	216k	28k	151k	**3.2k**	7.3k
∅ rank	**1.4**	2.4	4.4	3.6	4.7	5.6	5.7	6.4	7.1	7.0	4.8	4.6	3.6	4.7	**1.3**	2.4
∅ $\widehat{\text{rank}}$	**2.4**	2.6	4.4	3.6	4.7	5.5	6.7	6.4	7.0	6.8	4.8	4.6	3.7	4.7	**1.3**	2.5

Table 3.14: Test error (left) and model complexity (right) after training, measured by the overall number of parameters, averaged over 10 repetitions. In the second, the Hyperparameters were optimized on the basis of a small subset. The processing was canceled whenever it took longer than 24 hours and we mark the corresponding experiments as DNF.

Both SVMs perform similarly and mainly differ in the run time during training. The high training complexity of ISVM, resulting from the computation and incremental update of the inverse kernel matrix, prevents an application for datasets consisting of substantially more than 10000 samples such as *MNist*. The approximate nature of LASVM allows it to process the *MNist* dataset but it also reaches its limit for significantly larger datasets as *SUSY*. The instance-based ILVQ constructs far sparser models and achieves high accuracies throughout all datasets. It efficiently handles noisy datasets by sustaining a sparse model.

As expected, tree-based models require a comparably large amount of parameters for low-dimensional data, but are eminently efficient in high-dimensional spaces, due to their compressing representation. The opposite applies for instance-based models[9]. The ORF has the third highest accuracies and constantly beats LPP_{CART}. One explanation, already noticed by He, Chen, Li, and Xu (2011), is that LPP_{CART} trains each base classifier with samples of only one chunk. Therefore, knowledge integration across chunks is limited, since it is exclusively established by the weights of the base classifier. Furthermore, ORF benefits more from the sub-linear tree complexity because it generates a few deep trees instead of numerous, shallow ones as done by LPP_{CART}. Both SVMs could not process the large dataset SUSY due to their naturally high training complexity. Whereas, IELM and LPP_{Cart} were only limited by the inefficient MATLAB implementation.

The linear model of SGD_{Lin} uses the fewest parameters and performs especially well for high dimensional data. However, it struggles by design with non-linear separability as it is the case for the *Border* dataset, or whenever a small amount of examples is available per class (*COIL*, *Outdoor*). The last rank of NB_{Gauss} obscures the fact that it performs reasonably well without severe hiccups, incorporating a simple and sparse model. Nonetheless, the restriction to unimodal distributions is reflected in the results of the *MNist* and *Isolet* datasets.

Typical effects of different window-/chunk sizes are shown in Figure 3.14 exemplary for the *Overlap* dataset. Usually, algorithms benefit from an increased window-/chunk size. For instance, a larger window enables the ILVQ to find better positions for new prototypes and the ISVM to miss less support vectors. Simultaneously, the model complexity of ILVQ is reduced, since the insertion rate is coupled with the training-error. The IELM benefits from large chunks due to a more stable initialization of the output weight matrix. In case of LPP_{CART}, however, larger chunks reduce the overall number of base classifier, but at the same time each of them is trained on more training data, requiring a balancing of these criteria.

9 The number of parameters for instance based models can often be clearly reduced with a sparse representation for sparse high dimensional data as *MNist*. However, our results rely on a dense vector representation.

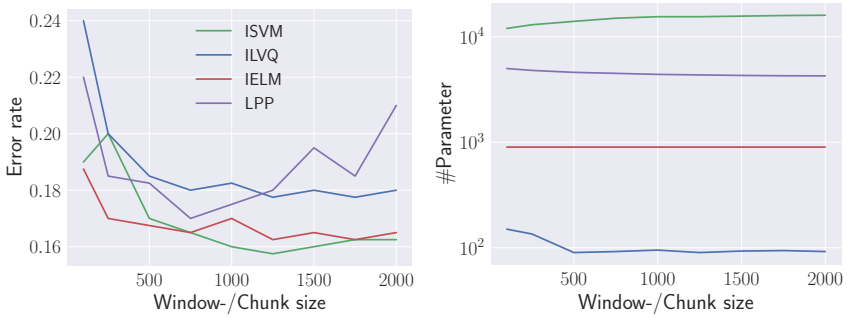

Figure 3.14: Influence of the window-/chunk size on the test error (left) and model complexity (right) for dataset *Overlap*.

.

Offline Setting - HPO with a small set of training samples

Table 3.14 gives an overview of the results in terms of classification performance and model complexity. The results of NB_{Gauss} are reported for the sake of consistency, since it incorporates no meta parameters and consequently achieves similar results. Regarding the error rate, most methods perform slightly worse than in the first setting, leading to the conclusion that hyperparameters can be robustly chosen based on few samples. However, the method losing the most performance is the IELM. This can be explained by the drastically sparser constructed model which is sufficient for the classification of a few examples but not complex enough for the whole dataset. Hence, the number of hidden neurons is underestimated in the optimization.

By contrast, all dynamically growing models tend to use significantly more parameters for various reasons: The kernel width of the SVMs is estimated less accurate with few examples, leading to an increased number of support vectors. In case of the ILVQ and ORF, growth is explicitly controlled via one meta parameter. However, the rate is overestimated because the model is obliged to learn faster when few instances are available. This leads to a higher complexity than necessary and can lead to overfitting. One solution could be to adjust the growth rate during learning guided by a supervised signal, e.g. the current classification performance.

SGD_{Lin} is the only algorithm which incorporates hyperparameters and achieves, nonetheless, similar results as in the first evaluation. Its model complexity is exclusively determined by the number of dimensions and the amount of different classes in the dataset. The only considered parameter, the learning rate, is reliably estimated on a subset of the data.

Online Setting - Same hyperparameters as in setting 2 (section 3.4.7)

The resulting ITTEs are given by Table 3.15. In general, the ITTEs are slightly higher accounting for the relatively high number of false classifications done at the beginning of learning. The SVMs maintain also in this setting the upper hand, albeit with less dominance. Tree-based

					ITTE			
Setting 3	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Border	**1.5**	3.4	6.0	5.3	11.6	12.0	62.5	5.6
Overlap	**18.3**	21.2	21.8	18.9	27.3	25.2	32.1	32.5
Letter	8.7	**7.3**	24.6	11.6	20.7	64.6	59.0	35.8
SUSY	DNF	DNF	**20.7**	21.5	DNF	DNF	21.3	26.5
Outdoor	**13.6**	17.7	65.8	17.6	31.5	26.7	82.0	35.0
COIL	24.6	33.7	33.4	**20.9**	41.3	36.9	90.4	29.8
DNA	**10.5**	**10.5**	26.9	15.4	22.1	50.9	15.3	13.9
USPS	**3.3**	3.4	15.5	7.3	13.4	11.2	11.5	24.0
Isolet	**6.4**	7.1	30.8	15.3	23.7	19.3	25.7	24.8
MNist	DNF	**2.5**	12.9	9.2	11.0	23.5	16.3	43.5
Gisette	3.7	**3.6**	9.7	8.9	23.3	19.5	7.9	26.0
∅	**10.1**	10.9	24.4	13.6	22.6	28.0	38.6	27.0
∅ rank	**1.3**	2.1	5.1	2.7	5.5	5.8	6.0	5.7
∅ $\widehat{\text{rank}}$	**2.7**	2.8	4.7	3.2	5.0	5.5	5.8	5.8

Table 3.15: ITTE averaged over ten repetitions. The ITTE uses each example of a given input stream first for testing and afterwards for model construction (see equation 3.1). We used the hyperparameters of the second offline setting, which are optimized on a small set of training examples (see section 3.4.7). The model complexity is neglected because it is similar to those of the second offline setting, due to the same hyperparameters. Only the training set of the original data was utilized as input stream (samples that were used in the HPO are excluded).

methods lose the most classification performance, indicating that the construction of an accurate tree model requires distinctly more examples than an instance-based one. This is due to the fact that split nodes are only added when they are necessary for the classification of the data seen so far. A few training examples can already be differentiated along one or two dimensions. Sophisticated tree models consisting of multiple splits are only required for larger amounts of training data.

In contrast, instance-based methods immediately classify examples along every dimension. Figure 3.15 highlights the different adaption rates between both model types by depicting exemplary learning curves. The ITTE is expected to be slightly above the test error for i.i.d. data because more mistakes are made at the beginning. However, in case of the *Outdoor* dataset the algorithms have partly a 20% lower ITTE. Figure 3.16 depicts the learning curves in both settings. The only explanation for this discrepancy is that data in the training set is quite different from those in the test set, implying that data is actually not i.i.d. As noted by Losing et al. (2015), this visual dataset consists of objects recorded outdoors. The lighting conditions significantly vary within the dataset and affect the underlying color based representation.

Restriction of the Overall Classifier Complexity

Methods as SGD_{Lin}, NB_{Gauss} and IELM are online algorithms and viable in lifelong-learning applications, since they are constant in their

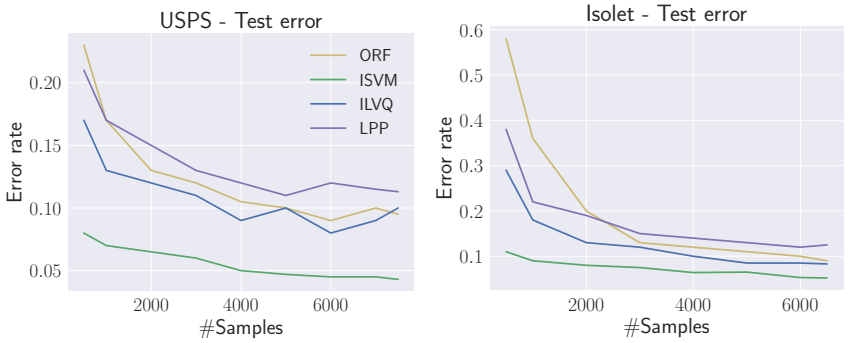

Figure 3.15: Learning curves of tree - and instance based models in comparison. Instance based methods are particularly at the beginning more accurate and with a steeper learning curve.

.

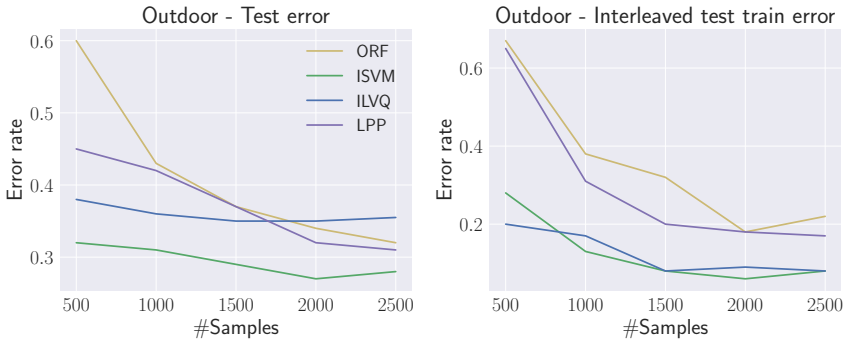

Figure 3.16: Learning curves for the Outdoor dataset in the off- and online setting. The dramatic discrepancy is subject to distinctly different training examples compared to those of the test set, implying the overall data of being not i.i.d.

.

complexity. ILVQ and LPP_{CART} can be easily restricted to a certain limit by strategies such as removing the "worst" prototype/classifier (Grbovic & Vucetic, 2009; Elwell & Polikar, 2009). In case of the SVMs, however, it is less trivial. Even though approaches as proposed by Downs, Gates, and Masters (2002) do reduce the number of support vectors, there is to the best of our knowledge no practical method to bound them strictly. This applies to a lesser degree also for the ORF. It learns by growing its trees continuously. Therefore, a depth reduction or pruning mechanism would be necessarily at some point.

Training- and Run-time

The algorithm implementations vary in the written programming languages as well as their efficiency. For instance, the fastest method NB_{Gauss}, written in C, required four seconds for the *Isolet* dataset, whereas the ISVM, implemented in Matlab, took ten minutes. Simply measuring the run time results not in a fair comparison, since the impact of the specific implementation is unclear. Therefore, we do not

explicitly compare training- and run-time but instead give a broad categorization based on complexity analysis and practical experience.

The training of both SVMS take by far the most time, since a quadratic programming problem is solved. However, LASVM is due to its approximate manner significantly faster than ISVM, but has the same worst case complexity. Clearly faster is LPP_{CART} and, since we use it in combination with CART its complexity is $\mathcal{O}(n \log(n))$, with n being the number of training examples. By performing the training chunk-wise, n is kept small and the training time is significantly reduced. The ORF has the same complexity class but the random splits drastically reduce the time in practice. The ILVQ and IELM have a similar training complexity $\mathcal{O}(np)$, where p is the number of prototypes / hidden neurons and usually $p \ll n$. However, the dynamic insertion of new prototypes in ILVQ requires additional calculations slowing it noticeably down. SGD_{Lin} and NB_{Gauss} are clearly the quickest with linear complexity $\mathcal{O}(n)$. In general, the train- and run time of growing models (LASVM, ISVM, ILVQ, LPP, ORF) naturally increase with model size, affecting the processing time, particularly, for large datasets. The run time of tree-based methods is sub-linear in regard to the model size $\mathcal{O}(\log l)$, l being the number of leaves, which makes them extremely efficient. All remaining algorithms have a linear relation between model complexity and run time. Nonetheless, the sparse models of SGD_{Lin} and NB_{Gauss} are usually the fastest in the field.

Concept Drift

Learning from data streams in non-stationary environments is a crucial part of incremental learning. Even though, the considered methods are not especially designed to handle concept drift, we exemplary investigate their robustness in respect to different types of drift. It is typically distinguished between *real drift*, referring to a changing class posterior distribution, and *virtual drift*, implying only a varying input distribution. These types of changes can occur in an *abrupt* or *incremental* way. Non-stationary environments and corresponding methods are extensively covered in Chapter 4.

Figure 3.17 illustrates the setting of the experiments. We optimized the meta parameters using the first 1000 instances and performed an evaluation in online scheme on the remaining instances. Various algorithms have been published explicitly tackling this challenge (Gama et al., 2004; Kolter & Maloof, 2007; Elwell & Polikar, 2011).

Datasets

The experiments relied mostly on artificial datasets with known drift characteristics. We also included two commonly used real-world benchmark, where generally the drift characteristics are not known. However, in the context of this thesis, we tackle this challenge and propose a method that determines the drift type within real-world tasks (see Section 4.3). Characteristics of the evaluated datasets are given in Table

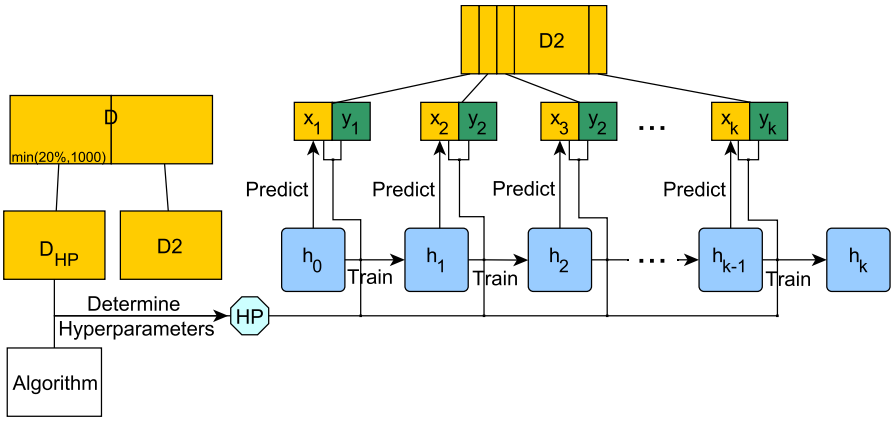

Figure 3.17: The concept drift experiments are using streaming datasets, which have a predefined order and there is no splitting into train- and test examples. The evaluation is performed in the online setting and the first 1000 samples are used for HPO.

3.16 and we briefly describe them in the following. More information about the data can be found in Section A.2.

Dataset	#Instances	#Feat.	#Class	Drift type
Inter. RBF	200000	2	15	abrupt real
Electricity	45312	5	2	real
Moving RBF	200000	10	5	incremental real
Cover Type	581012	52	7	real

Table 3.16: The evaluated datasets and their characteristics. The drift type for the real-world datasets is adopted from the experiments in Section 4.3.6.

Interchanging RBF Fifteen Gaussians with random covariance matrices are regularly swapping their position in feature space. Thereby, the number of swapped Gaussians increases each time by one until all are simultaneously changing their location. Each Gaussian is represented by an own class.

Electricity Each sample describes the current state of an electricity marked based on attributes such as day of week, time stamp, and market demand. The challenge is to to predict the relative change of price (higher or lower) compared to the last 24 hours. This problem is often used as a benchmark for concept drift classification (Baena-Garcıa et al., 2006; Kuncheva & Plumpton, 2008; Bifet et al., 2013; Gama et al., 2004).

Moving RBF Gaussian distributions with random initial positions, weights and standard deviations are moved with constant speed in 10-dimensional space. The weight controls the partitioning of the examples among the Gaussians.

			ITTE			
Drift setting	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Inter. RBF	54.1	**23.2**	70.6	70.5	65.7	70.1
Electricity	30.1	27.5	32.5	45.2	**15.4**	36.8
Moving RBF	54.4	**23.4**	82.0	84.1	59.4	82.9
Cover Type	10.4	11.9	60.3	48.7	**5.4**	45.4
∅	37.3	**21.5**	61.3	62.1	34.0	58.8
∅ Rank	2.3	**1.8**	5.0	5.5	2.0	4.5

			Complexity			
Drift setting	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Inter. RBF	762 k	46 k	166 k	900	**45**	60
Electricity	140 k	1.4 k	30 k	560	**6**	20
Moving RBF	721 k	2.6 k	32 k	1.0 k	**45**	60
Cover Type	1.3 M	292 k	76 k	6.0 k	**385**	756
∅	729 k	85 k	76 k	2.1 k	**123**	234
∅ Rank	6.0	4.3	4.8	3.0	**1.0**	2.0

Table 3.17: Achieved ITTE (top) and model complexity (bottom). Hyperparameters were optimized on the first 1000 samples.

Forest Cover Type The objective is to categorize cartographic variables as elevation, slope, soil type, ... of 30×30 meter cells to different forest cover types. This data set is a popular benchmark for drift algorithms (Bifet et al., 2013; Gama et al., 2003; Oza & Russell, 2001).

Results

The resulting ITTEs as well as model complexities are given by Table 3.17. We excluded the SVMs from the ranking, since the highly overlapping distributions led to an extensive growth of support vectors and to a processing time above 24 hours in all but one task. Methods that learn a model for all seen data instances, without regarding the temporal order such as NB_{Gauss} and LPP_{CART} are inappropriate for non-stationary environments as can be seen by the poor results. In general, a mechanism to forget obsolete knowledge is crucial to be able to deal with concept drift. This is given to some extent for the ILVQ and the SGD_{Lin}. Both incorporate a learning rate, which, if set flexible enough for the rate of drift, allows the model to adapt to new concepts. A common technique to deal with concept drift is the sliding window (Bifet et al., 2013; Widmer & Kubat, 1996). The ILVQ utilizes one to insert new prototypes such that the classification on recent examples is optimized. Hence, it weights new information stronger by design and has, therefore, the highest capacity of the methods to deal with concept drift. Our brief evaluation shows that some methods yield surprisingly accurate results for the considered datasets, while others simply fail. However, as we will see in Section 4.4.4, methods designed

	SVMs	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Endless learning	✗	✗	(✓)	(✓)	✓	✓	✓
Classification performance	⋆⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆	⋆
Learning speed	⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆
Model complexity	⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆
Training time	⋆	⋆⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆
Run time	⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆	⋆⋆⋆	⋆⋆⋆
Complexity of HPO	⋆	⋆⋆⋆	⋆	⋆⋆⋆	⋆⋆⋆	⋆⋆	-
Subset based HPO	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆⋆	-
Viable for concept drift	✗	✗	(✓)	✗	✗	(✓)	✗

Table 3.18: Discretized assessment of the core algorithmic properties. Especially, the major categories classification performance and model complexity are highly affected by the evaluated datasets and represent the average results on the diverse tasks considered in our experiments.

for non-stationary environments yield clearly better results.

3.4.8 *Discussion*

In this section, a diverse set of popular algorithms was investigated on different tasks. The results of the experiments are summarized in Table 3.18. It provides a fast overview of the core attributes of the methods, guiding the choice of an appropriate algorithm for a given task. We conclude that the SVMs often deliver the best classification performance at the expense of the most complex model. The approximate nature of the LASVM reduces the training time and allows the processing of larger datasets than feasible with the ISVM. The ORF performs slightly worse but has a very fast training- and run-time. However, its model as well as those of both SVMs grows linearly with the number of samples and cannot be limited in a straightforward way. Therefore, these algorithms are not suited for lifelong-learning in endless streams in contrast to the remaining methods, having either constant or an easily boundable complexity. The ILVQ offers an accurate and sparse alternative to the SVMs. LPP_{CART} is quite flexible, since the base classifier can be arbitrary selected, however, it may struggle by its limited knowledge integration across chunks. Tree-based models are especially suitable for high dimensional data because of their compressed representation as well as their sub-linear run time, which is independent from the number of dimensions. However, the compressed representation infringes the learning speed, such that instance-based models are learning faster and are more appropriate for tasks comprising only a few examples. The sparse models of SGD_{Lin} and NB_{Gauss} make them to particularly viable choices for large-scale learning in high-dimensional space on the one hand, but too simple for low-dimensional tasks that require non-linear separation, on the other. NB_{Gauss} and tree based methods are the easiest to apply in practice requiring no or just a little HPO, whereas the SVMs and the ILVQ require the most delicate

setting. Thus, depending on the actual demands, e.g. the available time and memory resources of the actual device and the possibility to interact for an optimum hyperparameter selection, the choice of different methods is advisable and our findings provide guidelines for such choices.

The chapter shed light on incremental learning in stationary environments. Nowadays, incremental learning is particularly relevant to efficiently process huge data streams that are continuously generated in various areas. Here, the assumption of stationarity is mostly violated, and data distributions change over time in various ways. In this section, algorithms assuming stationary settings were briefly examined within non-stationarity. The results revealed the necessity of specific approaches that explicitly deal with changing distributions. These are extensively discussed in the next chapter as our focus shifts to incremental learning in non-stationary environments.

4 CONCEPT DRIFT

Summary In this chapter, we focus on incremental learning under concept drift, a term to denote changes of the underlying data distributions, which is a prevalent phenomena in data streams. In particularly, we define concept drift and describe typical forms of change as well as the induced learning challenges. In this context, we contribute one method that is able to characterize drift in real-world data which is crucial from practical point of view. Furthermore, a new learning architecture, based on dedicated memories, is proposed which tackles the challenge of handling different types of drift in an adaptive way.

Source Code

- The Python source code for the Self-Adjusting Memory (SAM) is available at https://github.com/vlosing/SAMkNN.
- SAM is also integrated within the open source framework Massive Online Analysis (MOA) https://moa.cms.waikato.ac.nz/.
- The SAM-Ensemble (SAM-E) will be shortly integrated in MOA. Corresponding JAVA code is accessible at https://github.com/vlosing/SAM_E.
- Links to all datasets can be found at https://github.com/vlosing/datasets.

Parts of this chapter are based on:

- Losing, V., Hammer, B., & Wersing, H. (2016b). Dedicated memory models for continual learning in the presence of concept drift. In *Advances in neural information processing systems (nips) 29, continual learning workshop*.
- Losing, V., Hammer, B., & Wersing, H. (2016c). Knn classifier with self adjusting memory for heterogeneous concept drift. In *2016 ieee 16th international conference on data mining (icdm)* (pp. 291–300).
- Losing, V., Hammer, B., & Wersing, H. (2017b). Self-adjusting memory: How to deal with diverse drift types. In *Proceedings of the twenty-sixth international joint conference on artificial intelligence, IJCAI-17* (pp. 4899–4903).
- Losing, V., Hammer, B., & Wersing, H. (2018c). Tackling heterogeneous concept drift with the self-adjusting memory (sam). *Knowledge and Information Systems*, *54*(1), 171–201.
- Losing, V., Hammer, B., Wersing, H., & Bifet, A. (2019). Tackling concept drift with a diverse self-adjusting memory ensemble. In *Submitted to international conference on data engineering (icde) 2019*.

LEARNING from data streams demands more than an incremental model adaptation. The classical assumption of data being independent and identically distributed (i.i.d.), which is dominant in batch learning scenarios, usually does not hold for data streams (Žliobaite, 2010). As processes follow temporal patterns so do streams which try to capture these. For instance, forecasting tomorrow's weather being similar to today's one is usually an accurate and reasonable prediction because they are temporally related. Different temporal patterns are governed by different underlying distributions that generate the observations. For instance, seasons are overarching temporal patterns and the weather in winter is differently distributed than the weather in summer. In other words, data instances can be highly interdependent and follow evolving distributions. Changes of the underlying distribution are named *concept drift*, or the environment is termed as *non-stationary*, and sophisticated learning algorithms are required to perform under such circumstances.

In this chapter, we will describe useful algorithmic properties for

learning in the presence of drift. Consequently, we will discuss existing approaches and propose new ones. Various associated research questions have to be addressed, the most crucial ones are the following

1. How can we evaluate algorithms when the underlying distribution is constantly changing? How can the performance be monitored over time?

2. What are the strengths and weaknesses of state-of-the-art algorithms?

3. Which types of drift are contained in real-world data and how strongly are they pronounced? How can we extract such information from a given dataset?

4. How can we efficiently handle the stability-plasticity dilemma, i.e. how to decide whether past knowledge is still valid? What is necessary to deal with drift in lifelong-learning applications? How to design an architecture that is able to deal with different types of drift?

Hereafter, these challenges are addressed in an explicit way and answers are provided in form of algorithmic contributions or newly designed architectures, which are analyzed in diverse drift constellations.

1. Definition and Evaluation Concept drift can occur in various types, temporal patterns and different severities. Corresponding definitions and characteristics are provided in Section 4.1. In comparison to learning in stationary environments, numerous additional challenges are arising in changing environments. Section 4.1.5 discusses these in detail. One of those is the evaluation of incremental models facing changing distributions, which cannot be done on basis of a single test set.

2. Taxonomy of State-of-the-Art Methods Since learning under concept drift is a challenging and appealing field of research, numerous publications have been published tackling various aspects. Modern approaches are increasingly sophisticated in terms of combining several proven building blocks. Prominent modules are drift detectors, sliding windows and ensemble methods, mostly combined with incremental decision trees or k Nearest Neighbor (kNN). Section 4.2 introduces them in detail, providing a solid foundation for the subsequent treatment of state-of-the-art drift algorithms. These are briefly described and presented in tabular form, where the methods are categorized on basis of a previously defined taxonomy.

3. Concept Drift in Real-World Data In real-world applications, it is crucial to have information about drift characteristics in the data, since it facilitates the choice of suitable methods, preventing the costly and time-consuming process of trying out numerous algorithms with

the risk of excluding the best one based on wrong assumptions. Furthermore, it can provide more insight into the dynamics of the process itself. From a scientific point of view, it is important to assess the extent of drift in general. For instance, it is yet neither known whether drift is contained in commonly used benchmarks, nor any specifics about its type or degree. In spite of these open questions, there are yet no methodologies able to extract the drift characteristics from a given dataset. Section 4.1.5 elaborates on the research questions that arise in relation to determining drift in data, whereas in Section 4.3, we provide answers in form of a novel approach, able to detect the drift type and degree in a given dataset.

4. Handling Different Types of Drift Effective handling of different drift types demands completely different mechanisms, hence, the design of a system able to handle multiple types of drift is very challenging. Furthermore, underlying models such as kNN or decision trees require specific refinements that exploit their strengths and mitigate their weaknesses. Past knowledge needs to be integrated over time and outdated information has to be filtered or reduced in weight to adapt to new, possibly contradicting patterns in a fast way. In case of lifelong-learning scenarios, a compression of stored information is necessary to continue the learning process as soon as the maximum model size is reached. Hence, a high efficiency is essential in each module for keeping up with the incoming stream of data. Section 4.1.5 discusses each of the mentioned points in detail. We tackle some aspects within our proposed Self-Adjusting Memory (SAM) architecture, which integrates past and current concepts in an innovative way (Section 4.5). It explicitly maintains a consistency between old preserved concepts and the current one, thereby selectively filtering outdated instances. The level of abstraction is increased in an adaptive way, continuously balancing memory demands and classification performance. SAM is able to deal with heterogeneous types of drift adaptively switching its knowledge basis.

Ensembles aggregate weak learners and often drastically improve upon a single model's performance based on a high diversity of the overall model. In the context of concept drift, they allow the preservation of different concepts across the learners, which can be flexibly applied according to current drift patterns and types. Section 4.2 discusses them in detail. Their utility with respect to concept drift adaptation is highlighted by our proposed method SAM-Ensemble (SAM-E). It combines the benefits of highly diverse ensembles with the versatility of the SAM algorithm, resulting in an even higher overall flexibility and robustness which is reflected by a better classification performance. Efficiency is maintained by a parallel implementation that exploits the multiple cores of nowadays processing units.

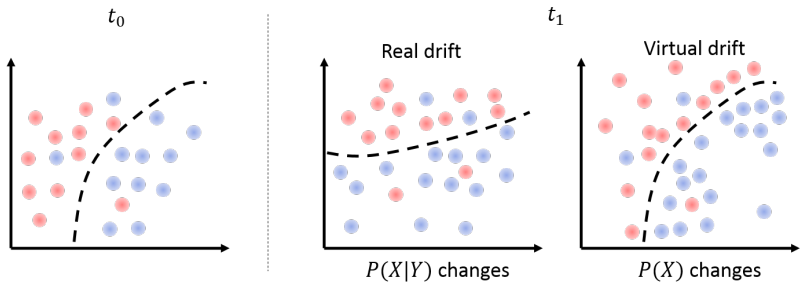

Figure 4.1: Exemplary changes of data distributions for two subsequent points in time t_0, t_1. The changes are either characterized as *real drift*, when the class boundary is affected, or *virtual drift*, when the distribution changes without affecting the class boundary.

4.1 FOUNDATION

In the following, concept drift and its two major forms are defined, as well as associated changing patterns characterized. We discuss the additional issues faced by incremental learning algorithms applied in non-stationary environments and name essential properties enabling the methods to meet the demands. Thereby, we explicitly point to corresponding novel approaches that are proposed in the frame of this thesis.

4.1.1 *Definition*

In data stream learning, concept drift occurs when the joint distribution $P_t(\mathbf{x}, y)$ changes for at least two time steps t_0 and t_1 (Gama et al., 2014):

$$\exists\, \mathbf{x} : P_{t_0}(\mathbf{x}, y) \neq P_{t_1}(\mathbf{x}, y),$$

The joint distribution can also be written as:

$$P_t(\mathbf{x}, y) = P_t(\mathbf{x}) P_t(y|\mathbf{x}),$$

where $P_t(\mathbf{x})$ is the distribution of the features and $P_t(y|\mathbf{x})$ the posterior probability of the classes. This is a broad definition and clearly not all included types of change are relevant in practice. In the context of data stream learning, changes of the distributions that manifest over longer time periods and distinctly affect the performance are usually considered. In the literature, alternative terms for concept drift such as non-stationary environments or evolving data streams are often used interchangeably.

4.1.2 *Types of Concept Drift*

Two major types of concept drift are usually distinguished as illustrated by Figure 4.1. The term *real drift* is used to specify that the

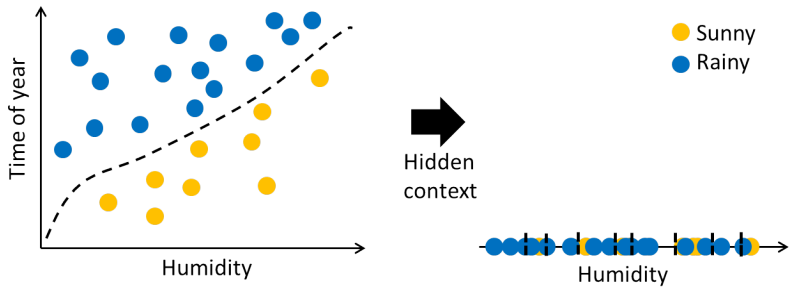

Figure 4.2: Real drift as a result of lacking information. The data overlaps if the time of year information is missing, affecting the class-boundary over time. An incremental system processing this type of distribution in temporal order will encounter changing concepts in form of real concept drift.

relation between observation and labels $P_t(y|\mathbf{x})$ varies over time, in other words, the location of the class boundary is shifting. *Virtual drift* is present when the feature distribution $P_t(\mathbf{x})$, the conditional probability $P_t(\mathbf{x}|y)$ and/or the class priors $P_t(y)$ are changing without affecting the posterior of the classes $P_t(y|\mathbf{x})$. In the context of batch learning, virtual drift is also known as covariate shift and refers to different feature distributions between the training- and test set (Bickel, Brückner, & Scheffer, 2009). One intuitive explanation may be that the sets were collected at different times leading to different captured distributions.

Concept drift can be interpreted as a lack of information about a given task, often termed as hidden context which guides changes, but is not captured in the data (Widmer & Kubat, 1996). For instance, assume the weather tomorrow depends only on the current season and todays average humidity level. A prediction system that has access to both information will not experience real drift, but only virtual one, since the sample independence is still violated. However, as soon as one of the information is missing, the data may overlap over time and contradict past patterns, leading to a shifting class boundary, since the available data is not sufficient to explain the process. Figure 4.2 provides a graphic illustration of this example.

In a nutshell, the more aspects of a problem are captured the less likely real drift occurs. However, the world is very complex, and humanity is still quite ignorant about it (Firestein, 2012), so that it is often unclear what type of information is necessary to completely explain a specific process. Furthermore, from a practical point of view it is mostly neither possible to obtain all required information nor to process them within the available time horizon. For instance, consider the task of a recommender system suggesting relevant news articles to a user. The user has a certain interest distribution about various topics, but is not particularly attracted by articles about cars. However, at some point the user decides to buy a new car and suddenly is keen on reading articles about cars in general, since he/she wants to be up-to-date with the current market. As soon as a new car is bought the

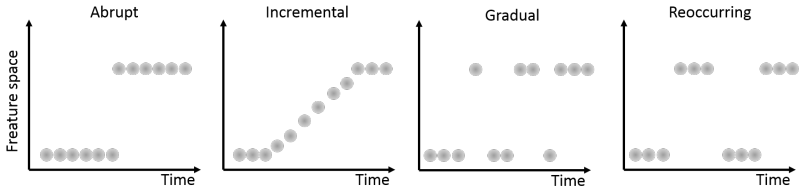

Figure 4.3: Different temporal patterns of change.

user returns to its old habits and declines articles about cars again. In such a case, it is unlikely that the recommender system has access to the cause of the change, i.e. the user wanting to buy a new car. Instead, the system will temporally encounter real concept drift and a quick reaction is crucial to prevent a series of irrelevant recommendations. This showcases one of countless everyday examples in which the data distribution is altered. Hence, sophisticated algorithms are indispensable in the field of data-stream learning to mitigate the repercussions of lacking context information.

4.1.3 *Patterns of Change*

The pattern or rate at which drift is happening can vary and is mostly categorized either into *abrupt*, resulting in a severe shift within the distribution, e.g., caused by a malfunctioning sensor, or *incremental*, an evolving change over time, e.g., evoked by a slowly degrading sensor. Sometimes a *gradual* transition between the former and current concept can happen as well. Figure 4.3 illustrates various types of change. Additionally, concept drift can be characterized as *reoccurring* to describe that previous concepts are becoming active again. It should be noted that these patterns are not formally defined but rather used as intuitive characterization of the way change is taking place, which might be ambiguous in particular cases. For instance, incremental drift, can also be described as a series of subsequent abrupt drifts, since the distinction is solely based on a subjective spacial and temporal scale. Also, gradual drift can be interpreted as reoccurring one.

4.1.4 *Model Evaluation*

Evaluation in non-stationary environments is not as straight-forward as in stationary ones. The classical train-test evaluation based on two disjunct sets for training and testing is inappropriate, since the underlying distribution is changing. In case of a few different and abrupt changes, one test set is required for each concept to estimate the generalization error in each of those. Even though this has partially been done for artificial data (Elwell & Polikar, 2009), it is not a suitable approach for real-world tasks, since it is not clear when different concepts start and how to get corresponding test sets. In particular, it breaks down for numerous concepts or in case of continuous or

incremental changes where the distribution may change after each example.

One common approach is to continuously monitor the performance and to average it over a time period. Concretely, the evaluation is performed in the online learning setting and relies on the ITTE

$$E(S) = \frac{1}{t}\sum_{i=1}^{t} 1 - \mathcal{L}(h_{i-1}(\mathbf{x}_i), y_i)$$

(Section 3.1.2). Precisely estimating the classification performance is particularly relevant, since one popular approach is to infer changes in the distribution based on significant deviations of the classification performance. For this reason, the current performance is estimated by determining the ITTE on a few recent instances.

However, in case of strong auto correlation of the labels, the ITTE can drastically overestimate the performance of an algorithm. For instance, a naive approach can simply predict the previous label to achieve a low ITTE without learning any general structure of the underlying data distribution. One valid option to prevent such exploitation is to train and evaluate algorithms in chunks according to the sequential peculiarities of the tasks. Assume stream S contains m complete chunks, where each chunk $C_j := [(\mathbf{x}_i, y_i), \ldots, (\mathbf{x}_{i+m_j}, y_{i+m_j}))]$ consists of $|C_j|$ datapoints with corresponding labels, then the chunk-wise ITTE is given as

$$E_c(S) = \frac{1}{\sum_{j=1}^{m} |C_j|}\sum_{j=1}^{m}\sum_{\forall(\mathbf{x},y)\in C_j} 1 - \mathcal{L}(h_{j-1}(x), y).$$

Regarding real applications, the exploitation is often naturally inhibited because ground-truth information is usually obtained with a certain delay, often too late for strategies that are based on predicting the most recent ground-truth for the current prediction to be effective.

In this thesis, the chunk-wise evaluation is applied in two different application scenarios, both with strong auto correlation because of their sequential nature. The objective of the first scenario is to predict the maneuver a car-driver will execute at the currently approached intersection. Maneuvers are distinguished into taking a turn, going straight or stopping. Predictions are made while the car is approaching the intersection. Each approach consists of numerous data-points, that are temporally ordered and all have the same label according to the taken action (Section 5.2). Hence, the model is trained and evaluated approach-wise, using all data points of one approach as a single unit. Similarly, we proceed in the second scenario which deals with human-motion classification. In this case, each motion is treated as one unit (Section 5.3). Further evaluation schemes such as weighted variants of the ITTE are discussed by Gama, Sebastião, and Rodrigues (2013), however, they are not considered within this thesis.

4.1.5 *Challenges*

Learning under concept drift is a highly challenging task, since fundamentally different types of drift can occur in various forms and rates and may even emerge simultaneously. In the following, we elaborate on associated research questions.

How to Quantify Drift in Real-World Data?

Drift-handling capabilities are mostly assessed based on artificial datasets because drift characteristics such as the type, pattern, strength, time of occurrence can be predefined. They provide a complete control of the conditions algorithms are facing and thus a precise investigation of their reaction. Additionally, artificial data can be generated with few dimensions to enable a visualization of the data and model structure. Therefore, scientists can get a precise picture about the qualities of methods in low-dimensional space with certain drift constellations. In contrast, estimating the drift-handling capabilities based on real-world data is rather difficult. It is neither known whether concept drift is contained at all, nor any specifics about its form. This also applies to already established real-world benchmarks, therefore, their relevance in assessing drift-handling capabilities remains unclear. Even though the research field of learning in non-stationary environments is quite established, there is rarely work aiming for the extraction of drift characteristics (virtual versus real, abrupt versus incremental, etc.) from a given dataset.

One reason for the lack of such methods is that it is very difficult to determine characteristics due to the high diversity of possibly occurring drift, especially considering the fact that multiple, different forms can simultaneously happen and may superimpose each other. Furthermore, real-world data is commonly high-dimensional, preventing a straight-forward extraction of the specifics based on visual inspection. Approaches trying to reduce the dimensionality beforehand are potentially erasing or distorting information and thus falsify the outcomes (Maaten & Hinton, 2008; Schulz, Gisbrecht, & Hammer, 2015).

A viable approach is to directly model the distribution based on density estimators such or clustering (Hastie & Tibshirani, 1996; Silverman, 2018; MacQueen, 1967). Subsequently, the development of these models can be analyzed over time, e.g. measuring the distance or similarity at different points in time. Dasu, Krishnan, Venkatasubramanian, and Yi (2006) propose a drift detection method by modeling the distributions based on kd-trees (Bentley, 1975) and analyze the Kulback-Leibler divergence between those. Even though this method scales linearly with the number of dimensions, the accuracy of the detections continuously decreases and is already with 10 dimensions significantly reduced.

A common approach, circumventing the problem of high dimensionality, is to monitor the classification performance over time instead of the distribution itself (Gama et al., 2004; Bifet & Gavalda, 2007). Deviations of the performance are then used to draw conclusions about

possibly happening drifts. Clearly, this gives only hints about the true behavior of the distribution and the results depend on the utilized classification algorithm, which cannot be simply generalized to other methods. However, it can still be useful to get rough estimates. For instance, applying a drift detection method delivers already information about the time of drift. Information about the slope of the classification performance before and after detected drift times can indicate the pattern, e.g. abrupt versus incremental. Summarizing such specifics of all detected drift can already deliver limited insights. However, various drift characteristics cannot be extracted by such a method, e.g. drift which has no effect on the performance is missed. From a practical practical point of view this is rather a secondary concern. Furthermore, it is not possible to distinguish between real or virtual drift, since both can affect the performance. Regular patterns in the performance may indicate reoccurring concepts, but this is not clearly ascertainable, since different concepts may lead to similar patterns.

In this thesis, an approach is proposed which sheds some light on the drift characteristics in real-world data (Section 4.3). It is also based on the classification performance, however, instead of monitoring the temporal classification performance, it analyzes the average performance obtained on numerous bootstrap samples of the data. Two statistical tests are applied to detect significant differences between the classification performances, resulting from classifiers equipped with differently sized sliding windows. The method is not only able to infer the presence of drift, but also about its type and severity. The tests are applied on common benchmarks to assess their suitability for concept drift analysis. Clearly, this method is only a first step as it delivers no complete picture of the drift-specifics. Nonetheless, the extracted information provides already valuable insights for researchers and practitioners.

How to Handle Heterogeneous Types of Concept Drift?

Learning under concept drift is an ill-posed problem in the sense that it is not possible to distinguish a burst of noise from a new concept in an online way. In practice, it is not clear when drift occurs and new concepts start. The stability-plasticity dilemma is drastically more difficult to handle because algorithms require ideally the same stability as for stationary data when the underlying distribution is static, but a substantially higher flexibility to adapt quickly as soon as changes take place. Thus, algorithms are required to dynamically balance their plasticity according to the volatility of the underlying distribution.

However, this presupposes the capability to detect change in an explicit or implicit way. The classification performance gives a direct measurement of the quality of a model and as long as the performance is unaffected, it is from a practical point of view usually irrelevant whether concept drift is happening or not. Therefore, methods often continuously monitor their performance and trigger reactions as soon as a change is detected. Different drift detection mechanisms are extensively discussed in Section 4.2.1.

The advantage of non-parametric methods which are able to dynamically adapt their model complexity is even greater in the presence of concept drift. Therefore, most approaches tackling drift in data streams are based on non-parametric models such as decision trees or kNN. As in stationary environments, adding new model parameters enables quicker learning and consequently quicker adaptation to change. However, unlike as for i.i.d. data, the capability to remove existing model parameters in a selective way is particularly crucial as it allows to instantly forget obsolete information and enhances the adaptation. One popular approach to deal with drift is to combine drift detection with forgetting, in terms of erasing past information as soon as drift is detected (Bifet et al., 2013; Kuncheva & Žliobaite, 2009; Klinkenberg & Joachims, 2000; Gama et al., 2004). Even though such a strategy allows to handle abrupt drift patterns, it also irreversibly erases all accumulated knowledge until then. In particular, real-world problems require more sophisticated methods which are able to deal with different types of drift selectively. For instance, reoccurring drift requires a conservation or integration of past knowledge in order to deal with reemerging patterns instead of relearning them always from scratch.

In recent years, a few algorithms have been published able to handle specific types of drift such as abrupt (Gama et al., 2004), incremental (Kolter & Maloof, 2007) as well as reoccurring (Elwell & Polikar, 2011) drift. Some methods can be used for several types of drift when their hyperparameters are set appropriately. However, this requires explicit prior knowledge about the task at hand. In the context of this thesis, a new architecture, the Self-Adjusting Memory (SAM), is proposed to integrate past and current concepts in an innovative way (Section 4.4). In contrast to other methods, SAM can handle a variety of drift types and patterns without any task-specific hyperparametrization. The architecture is based on two dedicated memories, one containing the current concept and the other accumulating all past ones. A consistency between both memories is continuously maintained by selectively filtering contradicting instances. Past knowledge is compressed whenever the upper memory bound is reached, leading to an adaptive level of abstraction making SAM well suited for lifelong-learning scenarios.

Yet, the focus on consistency within one single architecture prevents SAM to store diverse, possibly partially contradicting concepts which can occur when dealing with concept drift over time. Ensembles aggregate weak learners and often drastically improve on their individual performance. They are also very popular in the field of non-stationary stream learning because of their flexibility to selectively add and remove learners which enables a simple and efficient way to adapt to changing concepts. We discuss them in detail in Section 4.2.3. In Section 4.5, we propose the SAM-Ensemble (SAM-E), an algorithm combining the advantages of highly diverse ensembles with the versatility of the SAM algorithm, leading to an even higher classification performance and overall robustness. Even though an ensemble of SAMs is able to deal with concept drift on its own, we also utilize

a drift-detection on top, which triggers the replacement of the worst learners in case of a detection. This addition does not not only increase the adaptation speed of the overall model, but also selectively preserves learners with a suitable parameterization.

Concept drift requires sophisticated models often combining several algorithms as incremental learning, drift detection and ensembles. All these components have own hyperparameters which often crucially affect the overall performance, making it difficult to adjust each of them to the task-specific needs per hand. Particularly in case of changing environments, such parameters ideally require a dynamic configuration. We show for SAM and SAM-E that critical parameters can be avoided or automatically adjusted by using the current and past performance as guiding signal. For instance, based on performance maximization SAM switches between both memories as prediction model or SAM-E removes the currently worst performing learners in case of detected drift. Thus, both our methods are easy to use in practice without the necessity of task-specific adjustments.

4.2 RELATED WORK

Before we elaborate on state-of-the-art approaches and use a taxonomy to categorize them, the fundamental techniques of drift detection and Bagging ensembles are introduced, which are widely used building blocks within concept drift algorithms.

4.2.1 *Drift Detection*

A common way to handle drift, is to detect it first and then trigger a reaction. Drift detection algorithms explicitly determine the time of change t and enable a direct reaction, i.e. a model specific treatment of the detected change. The detection is mostly based on indirect cues such as the classification error. These are extracted from multiple windows, covering different time periods, and are analyzed for significant deviations (Gama et al., 2004; Bifet & Gavalda, 2007). There are also a few methods that directly detect drift on the input, thereby evaluating the distance or the Kullback Leibler divergence between modeled distributions of different time periods (Dasu et al., 2006; Kifer, Ben-David, & Gehrke, 2004). However, such approaches are only applicable for low-dimensional data because their run-time often increases exponentially with the number of dimensions and / or their performance drops significantly.

One popular detector is ADaptive sliding WINdowing (ADWIN) (Bifet & Gavalda, 2007), which efficiently monitors the binary error history (it could be any i.i.d. value between 0 and 1) in a window containing all values since the last detected change. The window is repeatedly partitioned into two sub-windows of various size. Whenever the difference of their average error exceeds a threshold, depending

on the size of the sub-windows and a confidence parameter, a change is detected and the older sub-window is dropped.

It should be noted that the popular scheme of applying statistical tests on the development of the classification error assumes that the underlying model is already converged. Otherwise, changes in the error distribution may simply be caused by the continuous adaptation of the model, rather than by a change of the underlying data distribution. In practice, this assumption is often violated, particularly in case of fast incremental concept drift, such that resulting p-values have rather an approximate character. Moreover, depending on the model, it is not clear that virtual drift in the data is mirrored by a drift of the classification error - hence some drift might also be missed.

4.2.2 *Sliding Window*

Sliding windows preserve the most recent data instances, since they assume that those are highly valuable for current predictions. Sliding windows are mostly combined with instance-based learners such as kNN (Cover & Hart, 1967) or SVM (Cortes & Vapnik, 1995). The window size governs the trade-off between a high plasticity (small window), resulting in quick adaptation, and a high stability (large window), providing as much information as possible during phases without drift. In case of stationary environments, often static sliding windows with a predefined size are used, whereas dynamic sliding windows, which adjust the size on the fly within a predefined range, are applied to deal with the volatility of non-stationary environments. Thereby, the size can be set on behalf of different techniques. One popular choice are drift detectors, where size is reduced such that examples before the latest detected drift are erased (Bifet et al., 2013). The size can also be adapted based on heuristics as done by Widmer and Kubat (1996), or by minimizing the amount of errors in the past: Klinkenberg and Joachims (2000) present a theoretically well-founded approach for SVMs. They set the size such that an estimation of the leave-one-out error is minimized without any parametrization.

4.2.3 *Bagging Ensembles*

Machine learning algorithms based on Bootstrap Aggregating (Bagging) (Breiman, 1996) such as the popular Random Forests (Breiman, 2001) are one of the most powerful state-of-the-art learning methods (Fernández-Delgado et al., 2014; Losing et al., 2018b). Ensemble methods aggregate weak learners based on voting and often boost the performance quite drastically. Concretely, the classification function for an input $\mathbf{x}$ is based on the weighted prediction of N learners:

$$\hat{y} = \underset{\hat{c} \in 1,\ldots,c}{\arg\max} \sum_{i=1}^{N} w_i * h^i(\mathbf{x}), \tag{4.1}$$

where w_i is the weight of learner h_i. One intuitive explanation for the effectiveness of Bagging is that variance is reduced without increasing the bias (Friedman, 1997). In offline Bagging, each model is trained with a bootstrap sample of the training set, and the classification is based on majority vote, i.e. $w_i = \frac{1}{N}$ (Breiman, 1996). However, the performance gain of ensembles crucially depends on the diversity of the learners. Hence, decision tree algorithms are particularly popular, since they generate different models even when the training set is only slightly altered.

Ensembles are also popular within concept-drift algorithms because of their flexibility to add and remove learners, enabling a simple and efficient way to deal with change. Furthermore, they are able to preserve information of different concepts distributed among their learners and selectively apply them by adjusting the corresponding weights w_i. Bootstrap samples of the whole data are not available in the field of data-stream learning, therefore online adaptations of Bagging are used. As an example, the proposal of Oza (2005) is widely accepted. It approximates the bootstrap sample by weighting each instance according to a Poisson distribution with $\lambda = 1$[1]. Another particularly appealing property regarding real-time processing, is the independence of models within Bagging ensembles, enabling a straightforward parallelization to distribute the computational complexity and thus reduces the run time.

4.2.4 *State-of-the-art Methods*

In the following, we first recapitulate a few prominent approaches for learning with drift before discussing overarching criteria which enable us to structure the methods according to underlying principles.

Probabilistic Adaptive Windowing (PAW) (Bifet et al., 2013) is a sliding window approach that uses the kNN classifier. One distinctive feature of this algorithm is that it randomly removes examples from the window, leading to a mix of recent and older instances. The window size is not strictly bounded and varies with high probability around a target size. ADWIN is used as drift detector and the knowledge obtained until the time of change is simply discarded, which makes the method ineffective for reoccurring concepts. Nonetheless, competitive results were achieved on various benchmarks.

Another sliding window approach that is combined with kNN is the *Just-In-Time* (JIT) classifier proposed by Alippi and Roveri (2008a, 2008b). The underlying drift detector, an extensions of the *cumulative sum* (CUSUM) method (Page, 1954), jointly monitors the classification error and the input data to shrink the window when necessary. One appealing feature is that the k parameter of kNN is dynamically adapted depending on the window size and an initial leave-one-out estimate to increase the performance. The approach was also extended to reoccurring concept in (Alippi, Boracchi, & Roveri, 2013).

1 The Binomial distribution of bootstrap samples with replacement converges towards a Poisson distribution with $\lambda = 1$ for large training sets.

The *Drift Detection Method* (DDM) was proposed by Gama et al. (2004). Although the name may suggest otherwise, it is a complete concept-drift algorithm consisting of a drift detector and a prescribed reaction. The detector models the classification error of two different time windows with a Gaussian distribution, based on the fact that the Binomial distribution can be closely approximated by a Gaussian distribution for sufficiently large samples. Drift is detected on two different sensitivity levels. First, a warning is issued if the difference between the mean errors of both time windows surpasses a corresponding confidence interval. This triggers the training of a new model in the background, assuming that drift has already started at this point. Drift is finally detected, when the error difference surpasses the confidence interval defined by the second sensitivity level. At this point the model in the background is used to replace the current one. This methodology is combined with different learning algorithms: a Perceptron, Neural Network and Decision Trees.

Based on the above mentioned Online Bagging algorithm (Oza, 2005), Bifet, Holmes, and Pfahringer (2010) propose to utilize a higher value of $\lambda = 6$ for the underlying Poisson distribution, resulting in distinctly higher instance weights to mitigate the slow learning of the VFDT. Furthermore, the ensemble diversity is increased by output detection codes. ADWIN is used as change detector for every tree such that each detected change leads to the replacement of the worst classifier. The resulting method *Leveraging Bagging* (LVGB) achieves accurate results on various benchmarks.

Recently, Gomes et al. (2017) proposed the *Adaptive Random Forest* (ARF) , also an ensemble of VFDTs, which relies on the same Bagging strategy as LVGB. Additionally, a random projection of the features is used, as done in the batch Random Forest, to increase the diversity. Similar to DDM (Gama et al., 2004), they tackle drift by applying a detector with two different sensitivity levels. The high sensitive setting initiates the training of a new model in the background, whereas the less sensitive one triggers the actual replacement. The assumption is that the drift has started before the less sensitive detection fires, allowing a quicker adaptation.

Jaber, Cornuéjols, and Tarroux (2013) present *Dynamic Adaption to Concept Changes* (DACC), an ensemble algorithm inspired by the Dynamic Weighted Majority (DWM) (Kolter & Maloof, 2007) method. A classifier of the worst half of the pool is randomly removed after a predefined number of examples and replaced by a new one. Newly generated classifiers are excluded from this elimination process for a predefined time. Incoming examples are exclusively classified by the classifier with the highest accuracy. This intuitive approach performed well within incremental and abrupt drift scenarios.

Learn++.NSE processes incoming examples in chunks with a predefined size (Elwell & Polikar, 2011). A base classifier is trained for each chunk and added to an ensemble. The weight of each member is determined by averaging the loss on recent chunks with a sigmoid function. Similar to AdaBoost (Freund et al., 1999), instances are weighted such that misclassified inputs have a higher impact on the calculated loss.

In contrast to other methods, members are not trained further, but preserve their initial state. Hence, this algorithm is able to deal with reoccurring concepts, as its members can be revived at any time to enhance the adaptation as formerly obsolete patterns become current again.

In the frame of this chapter, we will introduce two new architectures, the *Self-Adjusting Memory* (SAM) and the SAM-Ensemble (SAM-E) (Losing, Hammer, & Wersing, 2016c; Losing, Hammer, Wersing, & Bifet, 2019). SAM, extensively discussed in Section 4.4, integrates past and current concepts in an innovative way. It constantly maintains past information as long as it is consistent with the current one. Inconsistent instances are selectively removed, allowing to preserve as much information as possible. Even though the algorithm incorporates a sliding window, past knowledge is not fading out, but maintained by adaptive compression, continuously adjusting the level of abstraction to keep past information within the predefined memory bounds. This mechanism enables the handling of reoccurring concepts even when they reach far back in time. The architecture can be combined with different learning algorithms. However, the kNN algorithm is a great match as it naturally supports a selective editing of the model in an efficient way.

SAM is further improved by combining it within the Bagging ensemble SAM-E, as described in Section 4.5. In contrast to other drift ensembles, it builds on a method which is already capable to deal with various types of drift by its own. The ensemble utilizes SAM on basis of the kNN algorithm. Ensembles of the kNN are known to be ineffective, since kNN is a stable algorithm, meaning the model is robust against small variations of the data, leading to a low ensemble diversity (Breiman, 1996). Hence, the diversity is increased by randomizing the subspace as well as the k of kNN which turns out to be highly effective. ADWIN is used on top of the ensemble and triggers the replacement of the worst performing learners with new randomized SAM models, leading to an adaptive preservation of learners with suitable configurations for current demands.

4.2.5 *Taxonomy*

Design Principles

Concept drift algorithms can roughly be categorized along three overarching design principles.

Active / Passive Methods are often divided into *active* and *passive* approaches (Ditzler et al., 2015). The idea behind *active* approaches is to explicitly detect drift and only then react in a purposeful way. Mostly, drift-detection techniques are used to trigger a reaction which usually consists of erasing the knowledge acquired before the drift. In contrast, *passive* approaches continuously adapt their model, thereby dealing with drift in an implicit way. One example are linear models optimized

on the basis of SGD and a constant learning rate. Another example are static sliding window methods. Both are constantly updating their model with the most recent data, such that old patterns eventually are forgotten. On the one hand, *active* approaches have low computational demands in periods without change, since they only need to monitor the process without updating the model. On the other hand, *passive* methods usually have a simple design and are easy to use because they have no awareness of drift at all. Therefore, they do not have to deal with issues as false detections of drift or missed ones.

In general, methods that solely rely on the detection of drift are able to detect abrupt drift with low delay, however, they struggle with incremental change, which may remain undetected if it is too slow in relation to the monitored time span. Nowadays, purely active methods are basically not published anymore, therefore, the classic definition is rather obsolete. Instead, drift-aware methods are coupled with continuously updated models to get the best of both worlds (Bifet, Holmes, & Pfahringer, 2010; Bifet et al., 2013; Losing, Hammer, Wersing, & Bifet, 2019; Losing et al., 2018a). Hence, we extend the classic definition and term all drift-aware methods as *active* and denote the rest as *passive*.

Single Model / Meta Model Drift can be handled within a *single* model or on the basis of a *meta* model. Algorithms based on sliding windows or SGD handle drift on their own (Shalev-Shwartz et al., 2011; Bifet et al., 2013; Losing et al., 2018a; Alippi & Roveri, 2008b). They are able to react to drift in a very fine-grained way such as editing the examples within the window or changing the learning rate after each example. Meta models combine several models by means of high-level algorithms or rules. Algorithms based on Bagging are very popular meta models. They often use stationary algorithms such as VFDT or NB for their learners. Change is handled on a higher level by replacing outdated learners with new ones. Meta models are versatile because arbitrary underlying learning algorithms can be used, in some cases even batch algorithms (Elwell & Polikar, 2011). Some methods designed for concept drift are even integrating proven *single* drift algorithms within *meta* models, termed *single/meta*, for an increased flexibility and higher performance (Losing, Hammer, Wersing, & Bifet, 2019).

Model- /Data-based Representation The learned representation can either be centered around constructed models or the input data itself. Algorithms of the first type use the input only for model training and discard it afterwards. Thus, information is retained in a compressed way, allowing to condense large amounts of data. Exemplary algorithms include incremental decision trees, or ensemble of those (Domingos & Hulten, 2000; Saffari et al., 2009; Bifet, Holmes, & Pfahringer, 2010; Gomes et al., 2017). Contrary, methods focusing on data, explicitly keep the original input and construct on this behalf corresponding models. They explicitly edit the data, such as remov-

ing obsolete or redundant instances. However, the memory may be quickly exhausted in case of high-dimensional inputs. Typically, sliding window approaches are data centered. In this context, kNN-based algorithms are special, since they seamlessly blend between data and model. Editing the underlying data of a kNN algorithm simultaneously changes the underlying model and enables particular powerful and efficient algorithms. Such methods are denoted as *model/data*.

Application Characteristics

Besides such design principles, one can also identify overarching capabilities of drift algorithms which are important in respect to real-world applications. In the following, three criteria are defined.

Memory Requirement One aspect is whether the amount of allocated memory is *limited*, which is important for any serious real-world application. Particularly, lifelong-learning scenarios as well as applications using mobile platforms require efficient methods, able to perform on basis of strictly bounded resources. The memory demand of data-centered models as sliding windows is naturally bounded by a maximum number of storable instances. Model-centered algorithms are often *unlimited* in their memory demands, as it is the case for various incremental decision tree algorithms, which continuously grow with incoming examples (Saffari et al., 2009). Simply stopping the growth would mean to stop the learning process, which is unsatisfying in most applications. However, the majority of learning methods can be bounded by rather simple mechanisms. For example, the VFDT limits the memory by pruning the least used branches and, therefore, frees resources when necessary.

Drift Type Specificity Algorithms that are dependent on a specific configuration regarding their hyperparameters to handle certain drift types are not viable for scenarios where multiple drift types occur or where changing patterns are a priori unknown. For example, methods relying on drift detection have usually a sensitivity parameter which predefines the range of drift speed an algorithm can handle. Hence, they need an adaptation to the application at hand. Whereas algorithms that automatically adapt to different types of drift can be applied out-of-the-box in a wide range of applications.

Handling of Reoccurring Drift Applications where data that follow repetitive patterns benefit from algorithms that are able to reuse past information over long time periods. Some patterns may have been temporarily obsolete but eventually reemerge again. In case of reoccurring concepts, methods that simply discard former knowledge in case of detected drift are as unfit as plain sliding-window approaches where knowledge is slowly fading out due to the predefined maximum size. Instead, sophisticated techniques are demanded that conserve former concepts and revive them when necessary. One possibility is to preserve past knowledge in separate models and to apply them when

Table 4.1: Taxonomy of state-of-the-art algorithms according to design principles (top) and application characteristics (bottom).

Algorithm	Approach	Drift Handling	Basis of Representation
ARF	Active	Meta	Model
DACC	Passive	Meta	Model
DDM	Active	Single	Model
JIT	Active	Single	Data
L++.NSE	Passive	Meta	Model
LVGB	Active	Meta	Model
PAW	Active	Single	Model/Data
SAM	Active	Single	Model/Data
SAM-E	Active	Single/Meta	Model/Data

Algorithm	Memory	Drift Type Specificity	Reoccurring Concepts
ARF	Unlimited	Yes	No
DACC	Limited	Yes	No
DDM	Limited	Yes	No
JIT	Limited	No	Yes
L++.NSE	Unlimited	Yes	Yes
LVGB	Unlimited	Yes	No
PAW	Limited	Yes	No
SAM	Limited	No	Yes
SAM-E	Limited	No	Yes

corresponding concepts emerge again (Elwell & Polikar, 2011; Alippi et al., 2013).

Having introduced these overarching principles, Table 4.1 provides a high-level characterization of the mentioned algorithms. The categorization is based on the originally published version. It does not consider extensions of the methods themselves or transferable mechanisms from similar methods. For example, most algorithms can be limited in terms of memory consumption with various adaptations. Also the chosen learning algorithm within ensembles does affect some criteria. The overview shows that most methods nowadays are drift-aware. It also reveals two common issues, which are addressed within this thesis:

- Methods often require a manual configuration to specific drift types.
- Reoccurring concepts are not especially considered.

4.3 QUANTIFYING CONCEPT DRIFT

This section focuses on the challenge of identifying the drift characteristics in real-world data. In particular, we investigate whether commonly applied real-world benchmark contain any drift at all. While concept drift is explicitly generated in artificial datasets, it is rather difficult to identify in real-world data. The common approach is simply to

reason about whether one expects drift being present due to domain knowledge. However, any reliable information about the type of drift contained in the data is crucial. From the scientific point of view, it enables a more detailed analysis of drift handling capabilities. In terms of practical use, information about the drift characteristics clearly facilitates the choice of an appropriate algorithm for a given task. For instance, whenever no real drift is present at all, classical incremental / online algorithms such as NB or incremental decision trees may be the better choice in regard to the efficiency as well as the classification performance.

We propose a method to determine the drift type, real or virtual, and its degree in a given dataset. Hence, it directly tackles the challenge of how to determine drift characteristics in a given data stream. To the best of our knowledge, this has not been done so far in literature. The method processes data in online fashion, however, multiple passes are required. In case of very large or potentially infinite streams, we assume a sufficiently large subpart to be available, containing similar drift characteristics as the whole stream.

The method consists of two consecutive tests. The first detects real drift. If that is not the case, we apply the second one to test for virtual drift. No drift is present whenever both tests are negative. We use sliding windows of various sizes applied on bootstrap samples (Wilcox, 2012) of the dataset and analyze the resulting classification errors for significant deviations. The degree of drift is inferred from the magnitude of the error differences.

The type of classifier coupled with the sliding windows is exchangeable. However, the results of the tests depend on the classification performance and consequently on the utilized algorithm. From a practical point of view, the tests should be performed with the same or at least a closely related algorithm as the one used for the application. In the context of this thesis, kNN is used because its result are particularly interesting in respect to the drift-handling architecture SAM (Section 4.4), which is mostly combined with it. Furthermore, kNN is a popular algorithm in the domain of learning from data streams such that our findings are directly relevant for numerous methods.

4.3.1 *Prerequisites*

Assume a representative set of streaming instances is given $S = (s_1, s_2, \ldots, s_n)$ with n tuples $s_i = (\mathbf{x}_i, y_i)$, whereby n is chosen as large as possible, respecting computational restrictions.[2] The tests heavily rely on bootstrap samples from the given data. In the following, we will use two types of bootstrap samples: Samples $B_1, \ldots, B_m$ with $|B_i| = |S|$ are randomly drawn from S with replacement. Hence, the samples have a *random order*. In contrast, $\tilde{B}_1, \ldots, \tilde{B}_m$ with $|\tilde{B}_i| = |S|$ are drawn from S with replacement whereby the *ordering in S is explicitly preserved*.

2 The complete datasets was used in the experiments. However, in real-world tasks this is often not possible.

For every such data stream B_i and $\tilde{B}_i$, we evaluate the classification results of classifiers induced by sliding windows of different size. We consider a small number of different representative sliding window sizes $W = \{w_1, \ldots, w_b\}$, where b is usually small for computational efficiency and to prevent false positives of the tests[3]. Given the current time t, a window size w and data stream $(s_1, s_2, \ldots, s_n)$, a sliding window induces kNN models $h_t = \text{kNN}_{\{s_{t-w+1},\ldots,s_t\}}$ and a corresponding ITTE. The errors induced by models of window size w and data stream B_i are referred to as e_i^w, and for data stream $\tilde{B}_i$ as $\tilde{e}_i^w$, respectively.

4.3.2 *Test for Real Drift*

This test is inspired by the observation that whenever real drift is present, an algorithm using a sliding window approach tends to make fewer errors with smaller windows than with very large ones. This contradicts the classical assumption for i.i.d. datasets as stated in the PAC model (Mitchell, 1997): The error rate of a consistent learning algorithm decreases with increasing number of examples towards the Bayes error. However, streaming data containing real drift is not i.i.d. More mistakes are done when outdated instances are in conflict with examples of the current concept. In such a case, small windows contain less outdated instances leading to more accurate results.

Within the test, the behavior of large windows is evaluated by a reference model $\widehat{\text{kNN}}_t := \text{kNN}_{\{s_{t-\hat{w}+1},\ldots,s_t\}}$. It is coupled with a sliding window of size $\hat{w}$, chosen as large as possible regarding computational restrictions[4]. Its classification error is compared with those of models with smaller window size w, as evaluated by $\tilde{e}_i^w$. More precisely, we test whether any test model yields a significantly lower error rate than the reference model. We refer to the error of the reference model $\widehat{\text{kNN}}$ on the bootstrap sample $\tilde{B}_i$ as $\tilde{r}_i$. A one-sided hypothesis test with an α-value of 1% is used. Thereby, the null hypothesis is given as: The error of the reference model $\tilde{r}$ is smaller or equal to the error $\tilde{e}^w$, induced by a model with window size w. The first percentile β_w of the error differences $\{\tilde{r}_i - \tilde{e}_i^w | i \in \{1, \ldots, m\}\}$ is determined and the null hypothesis is rejected as soon as $\beta_w > 0$. Real drift is inferred to be within the data if the null hypothesis is rejected for any window size $w \in W$.

4.3.3 *Test for Virtual Drift*

In case of a negative real-drift test, a detection for virtual one is performed. Here, we test whether the error rate of any test model is lower for the ordered bootstrap samples in comparison to the unordered ones. Intuitively, randomizing the temporal order of virtual drift datasets leads to a higher error rate because of the following reasons: In case of

3 Throughout all experiments three models were consistently used with $W = \{500, 1000, 5000\}$.

4 In the experiments, $\hat{w} = 20000$ was used.

changing concepts, each concept is processed after another, making the problem less challenging than learning the whole distribution at once. Furthermore, the representation of each concept is richer, since the space of the sliding window is largely used for one concept at a time, instead of being partitioned among all of them.

In this case, the null hypothesis is the error e^w being smaller or equal to $\tilde{e}^w$ for a model with a sliding window size w. Analogous to the previous test, the first percentile β_w of the error differences $\{e_i^w - \tilde{e}_i^w | i \in (1, \ldots, m)\}$ is determined and virtual drift is inferred to be present if $\exists w \in W : \beta_w > 0$.

4.3.4 *Drift Degree*

Whenever one of the tests is positive, the corresponding drift degree is estimated. The degree simply correlates with the magnitude of the relative error differences. An error fraction ψ is used to distinguish between low and high degrees of drift. ψ is differently defined for each of the tests:

Real Drift

Regarding the real-drift test, ψ is the minimum error fraction of all test models in comparison to the reference model:

$$\psi = \arg\min_{w \in W} \frac{\sum_{i=1}^{m} \tilde{e}_i^w}{\sum_{i=1}^{m} \tilde{r}_i}.$$

Virtual Drift

In case of the virtual-drift test, ψ is defined as the minimum fraction between the error achieved on the ordered and unordered bootstrap samples.

$$\psi = \arg\min_{w \in W} \frac{\sum_{i=1}^{m} \tilde{e}_i^w}{\sum_{i=1}^{m} e_i^w}.$$

Subsequently, ψ is compared against a threshold γ to obtain the degree:

$$\psi \mapsto \begin{cases} \text{high} & \text{if } \psi < \gamma \\ \text{low} & \text{otherwise.} \end{cases}$$

4.3.5 *Datasets*

Table 4.2 shows the analyzed datasets. We used various popular artificial and real-world benchmarks. The artificial benchmarks are useful to evaluate the accuracy of the proposal, since the true drift characteristics are known. In the following, the datasets which were were not yet introduced are briefly described. More information about all datasets is provided in Section A.2.

Table 4.2: Characteristics of the considered datasets. A=Abrupt, I=Incremental, R=Real, V=Virtual

					Drift Characteristics			
Dataset	#Inst.	#Feat.	#Class	Type	Type	Pattern	Degree	Reocc.
SEA Concepts	50K	3	2	Art.	R	A	?	No
Rot. Hyperplane	200K	10	2	Art.	R	I	?	No
Moving RBF	200K	10	5	Art.	R	I	?	No
Inter. RBF	200K	10	15	Art.	R	A	?	No
Moving Squares	200K	2	4	Art.	R	I	?	Yes
Transient Chessb.	200K	2	8	Art.	V	A	?	Yes
Chessboard i.i.d.	200K	2	8	Art.	-	-	-	-
Mixed Drift	600K	2	8	Art.	R,V	A,I	?	Yes
Poker	829K	10	10	Art.	V	A	?	No
Outdoor	4K	21	40	Real-world	?	?	?	?
Weather	18.1K	8	2	Real-world	?	?	?	?
Electricity	45.3K	5	2	Real-world	?	?	?	?
MNIST	70K	764	10	Real-world	-	-	-	-
Rialto	82.2K	27	10	Real-world	?	?	?	?
Cover Type	581K	54	7	Real-world	?	?	?	?

SEA Concepts This dataset was proposed by Street and Kim (2001) and consists of 50000 instances with three attributes of which only two are relevant. The two class decision boundary is given by $f_1 + f_2 = \theta$, where f_1, f_2 are the two relevant features and θ a predefined threshold. Abrupt drift is simulated with four different concepts, by changing the value of θ every 12500 samples. Also included are 10% of noise.

Rotating Hyperplane A hyperplane in d-dimensional space is defined by the set of points x that satisfy $\sum_{i=1}^{d} w_i x_i = w_0$. The position and orientation of the hyperplane are changed by continuous addition of a term δ to the weights $w_i = w_i + \delta$. We used the Random Hyperplane generator in MOA with the same parametrization as Bifet et al. (2013) (10 dimensions, 2 classes, δ=0.001).

Moving Squares Four equidistantly separated, squared uniform distributions are moving in horizontal direction with constant speed. The direction is inverted whenever the leading square reaches a predefined boundary. Each square represents a different class. The added value of this dataset is the predefined time horizon of 120 examples before old instances may start to overlap current ones. This is especially useful for dynamic sliding window approaches, allowing to test whether the size is adjusted accordingly.

Transient Chessboard Virtual drift is generated by revealing successively parts of a chessboard. This is done square by square randomly chosen from the whole chessboard such that each square represents an own concept. Every time after four fields have been revealed, samples covering the whole chessboard are presented. This reoccurring alternation penalizes algorithms tending to dis-

card former concepts. To reduce the impact of classification by chance we used eight classes instead of two.

Mixed Drift The datasets Interchanging RBF, Moving Squares and Transient Chessboard are arranged next to each other and samples of these are alternately introduced. Therefore, incremental, abrupt and virtual drift are occurring at the same time, requiring a local adaptation to different drift types.

Chessboard i.i.d. The data was randomly sampled from a chessboard pattern, consisting of eight different classes. This dataset has no drift at all, but is useful in terms of evaluating the method regarding false positives.

Weather *5975223* introduced this. In the period of 1949-1999 eight different features such as temperature, pressure wind speed etc. were measured at the Offutt Air Force Base in Bellevue, Nebraska. The target is to predict whether it is going to rain on a certain day or not. The dataset contains 18159 instances with an imbalance towards no rain (69%).

4.3.6 *Experiments*

Apart from real-world datasets, the tests were applied on artificial benchmarks as a proof of concept. Furthermore, datasets without any drift are used as well (*Chessboard i.i.d.*, *MNIST* (Lecun et al., 1998)).

500 bootstrap samples were generated per dataset, but the results are often unambiguous such that a considerably smaller amount would have been sufficient for clear results. The reference model $\widehat{\text{kNN}}$ was coupled with a sliding window $\hat{w} = 20000$ samples, whereas the test models had windows of 500, 1000 and 5000 samples. Each test for each test model was performed with an α-value of 1%. Using the union bound this sums up to a total α-value of 3% for the whole real drift test as well as for the virtual one. The drift-degree threshold γ was set to 0.85. Hence, a *high* degree is inferred when the relative mean error difference is larger than 15%.

Real Drift Test

The results are listed in Table 4.3. In all artificial datasets, real drift is correctly detected. Also its absence in the datasets *Transient Chessboard*, *Chessboard i.i.d.* and *MNIST* is accurately inferred. It is inferred that *SEA Concepts* and *Rotating Hyperplane* have a low drift degree which is reasonable because, in case of *SEA Concepts*, only three minor abrupt changes are occurring, whereas a slowly progressing incremental drift is present in *Rotating Hyperplane*, both having a rather small impact on the classification performance. All remaining artificial datasets a high drift degree is attested due to the substantial error rate deviations. Figure 4.4 depicts some exemplary histograms of error differences.

Table 4.3: Mean error rates and corresponding standard deviations achieved by kNN classifier with differently sized sliding windows on ordered bootstrap samples. A dataset is tested positive for real drift as soon as the first percentile of the distribution of error differences between reference and any test model is larger then 0. Entries leading to the highest positive first percentile β_w are marked bold. Naturally, this strongly correlates with the highest differences between the mean values. We omitted the first percentile values of each model for the sake of clarity.

Dataset	kNN_{500}	kNN_{1000}	kNN_{5000}	$\widehat{kNN}_{20000}$	Real drift	Degree	ψ
SEA Concepts	11.16(.1)	10.76(.1)	**10.52(.1)**	12.12(.1)	✓	Low	.87
Rot. Hyperplane	15.42(.1)	14.36(.1)	**12.67(.1)**	12.82(.1)	✓	Low	.98
Moving RBF	**8.89(.0)**	9.38(.0)	14.51(.1)	19.32(.1)	✓	High	.46
Inter. RBF	**3.63(.0)**	6.66(.0)	25.56(.1)	39.37(.1)	✓	High	.09
Moving Squares	**32.80(.1)**	36.72(.1)	43.18(.1)	42.86(.1)	✓	High	.77
Transient Chessb.	11.22(.1)	10.85(.0)	5.19(.0)	3.18(.0)	✗	-	-
Chessboard i.i.d.	25.48(.1)	17.72(.1)	8.05(.1)	4.33(.1)	✗	-	-
Mixed Drift	**8.49(.0)**	13.67(.0)	18.74(.1)	25.54(.0)	✓	High	.33
Poker Hand	14.21(.0)	13.04(.0)	13.04(.0)	10.76(.0)	✗	-	-
Outdoor	11.37(.2)	11.05(.2)	10.86(.2)	10.86(.2)	✗	-	-
Weather	15.94(.1)	15.58(.1)	15.19(.2)	15.14(.2)	✗	-	-
Electricity	**15.10(.1)**	15.84(.1)	17.96(.1)	19.54(.1)	✓	High	.77
MNIST	12.34(.1)	9.57(.1)	5.39(.1)	3.75(.1)	✗	-	-
Rialto	**16.00(.1)**	16.77(.1)	18.00(.1)	17.89(.2)	✓	Low	.89
Cover Type	3.86(.0)	**2.89(.0)**	3.05(.0)	3.17(.0)	✓	Low	.91

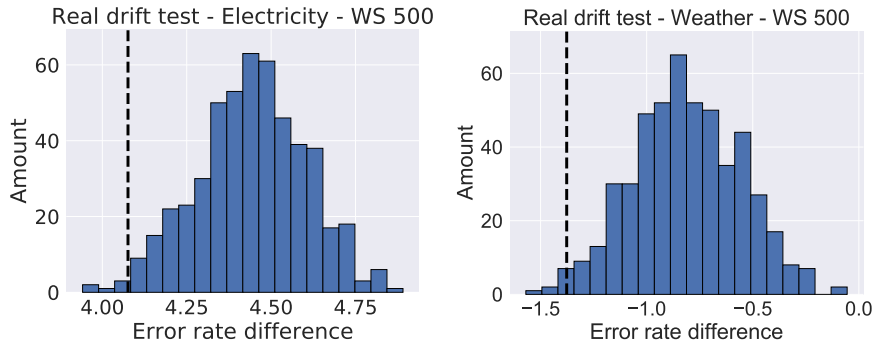

Figure 4.4: Exemplary histograms of error rate differences achieved on 500 ordered bootstrap samples. The differences shown here were obtained between the reference model using a sliding window of 20000 samples and a test model with 500 samples. The test is positive whenever the first percentile of the distribution, illustrated by the dashed line, is larger than 0. This is true for the *Electricity* dataset (left) but not for *Weather*.

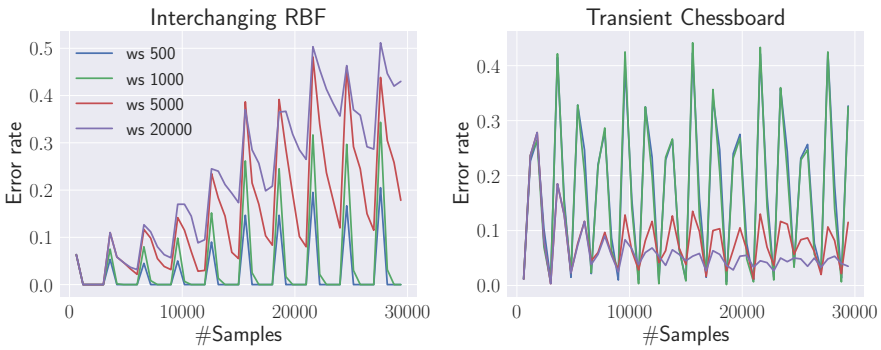

Figure 4.5: Error rate course of kNN models with differently sized sliding windows. In the case of real drift (left), the error rate of larger windows recovers significantly slower after each abrupt drift. In contrast, larger windows achieve lower error rates when no real drift is present (right) because the additional data facilitates the problem.

Table 4.4: Mean error rates and corresponding standard deviations achieved by kNN classifier with differently sized sliding windows on unordered bootstrap samples. A dataset is tested positive for virtual drift as soon as the first percentile of the distribution of differences between the error achieved on ordered (see Table 4.3) and unordered bootstrap samples with the same window size is larger then 0. Entries leading to the highest positive first percentile β_w are marked bold. We omitted the first percentile values for the sake of clarity.

Dataset	kNN_{500}	kNN_{1000}	kNN_{5000}	Virtual drift	Degree	ψ
Transient Chessboard	**25.47(.11)**	17.72(.08)	7.98(.06)	✓	high	.70
Chessboard i.i.d.	25.51(.10)	17.74(.10)	8.04(.07)	✗	-	-
Poker Hand	36.56(.05)	**34.51(.05)**	29.77(.06)	✓	high	.52
Outdoor	**33.94(.67)**	26.39(.69)	-	✓	high	.66
Weather	**23.05(.38)**	22.37(.39)	21.52(.37)	✓	low	.95
MNIST	12.52(.1)	9.63(.1)	5.55(.0)	✗	-	-
Rialto	**55.83(.20)**	49.58(.21)	36.52(.21)	✓	high	.44

Three out of the six real-world datasets contain real drift. Our results confirm the common assumption of real-world tasks having rather little amounts of real concept drift, particularly when compared to artificial ones. Only the *Electricity* task incorporates a high degree of real drift.

Figure 4.5 illustrates the error-rate development of differently-sized sliding windows. In the case of real drift (*Interchanging RBF*), the error rate of larger windows is higher, whereas the opposite is true when real drift is not present (*Transient Chessboard*).

Virtual Drift Test

Table 4.4 shows the results of the virtual drift detection. The test correctly identifies virtual drift within the *Transient Chessboard* data set as well as its absence in the tasks *Chessboard i.i.d.* and *MNIST*. Dataset

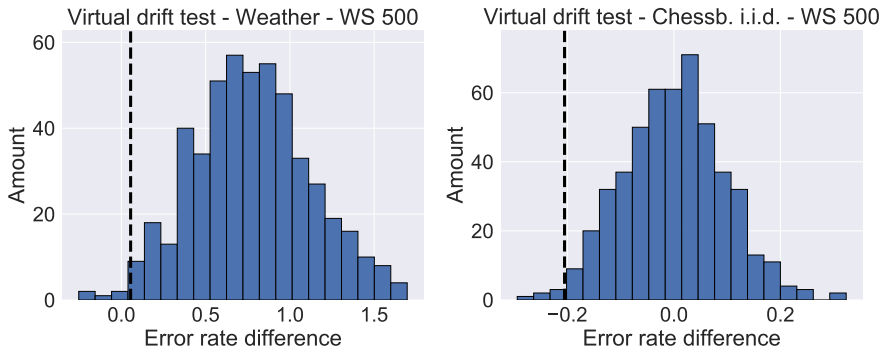

Figure 4.6: Exemplary histograms of error rate differences achieved with the test model using a sliding window of 500 samples. The differences were obtained between the achieved error rate on unordered and ordered bootstrap samples. The test is positive whenever the first percentile of the distribution, illustrated by the dashed line, is larger than 0. This is true for the *Weather* benchmark (left) but not for *Chessboard i.i.d.* (right).

Weather contains a low degree, whereas all others incorporate a high one. Figure 4.6 shows exemplary distributions of the error differences.

In a nutshell, the tests reliably detected real and virtual drift as well as their absence in the artificial data. However, they were not able to infer that the *Mixed Drift* dataset contains next to the detected real drift also virtual one. This issue is extensively discussed in Section 4.3.7. In case of the real-world benchmarks, there is no ground truth, however, the classification in the respective types appears plausible.

4.3.7 *Discussion*

In this section, we tackled the common problem of unknown drift characteristics in real-world datasets. To the best of our knowledge, this has not been done so far. We contributed a methodology of two consecutive tests able to detect the drift type, virtual or real, and its degree. Both tests are based on detecting significant error differences of classification performances achieved on bootstrap samples of the dataset by differently sized sliding windows. The degree of drift is inferred from the magnitude of the differences.

There are some downsides of the approach. The results depend on the classification performance of the chosen algorithm. Other algorithms will deliver different classification performances due to their own characteristics. Hence, the results are particularly relevant if the tests are performed with the same or at least a closely related algorithm as the one intended for the real application. The tests also depend on properly chosen sliding window sizes. The measured error differences hinge on the drift speed as well as the complexity of the task and are

properly reflected in adequate window sizes. In case of the real drift test, too large windows may not lead to significant error differences. Since the drift degree is estimated according to the magnitude of error differences, the choice of the window sizes naturally affects these as well. A possible solution is simply to test more window sizes, but such an approach demands to account for the increased risk of false positives, e.g. a Bonferroni correction of the α-values (Bonferroni, 1936). However, the experiments showcased that as long as a reasonable range of window sizes is covered, the tests deliver reliable results for a broad range of tasks. Naturally, the significance requirement prevents the detection of very small amounts of drift, which from a practical point of view is rather negligible because of an expected small impact on a continuously adapting system. In cases, where virtual and real drift are both present the proposed methodology is only able to detect the real drift type.

In spite of these minor issues, our approach correctly classified all artificial datasets and delivered a reasonable categorization of the real-world benchmarks. We conclude that concept drift is commonly found in real-world streaming and can significantly affect the classification performance, demanding for sophisticated methods which are able to handle diverse types of drift.

4.4 SELF-ADJUSTING MEMORY (SAM)

In recent years, a few algorithms have been published able to handle specific types of drift such as abrupt (Gama et al., 2004), incremental (Kolter & Maloof, 2007) as well as reoccurring (Elwell & Polikar, 2011) drift. Some methods can be used for several types of drift when their metaparameters are set appropriately. However, this requires explicit prior knowledge about the task at hand. In real-world applications, changes often occur in multiple and even concurrent forms at various rates. One example is the field of personalized assistance, in which individual user behavior is taken into account to provide appropriate assistance in various situations (Schiaffino, Garcia, & Amandi, 2008). However, individual behavior in particular can change in arbitrary ways. Systems anticipating only certain forms of drift will perform suboptimal at best, or fail completely at worst, when unexpected forms of change occur.

This section introduces the Self-Adjusting Memory (SAM), a versatile architecture designed to face the challenge of handling different types of concept drift in data-stream learning. It exhibits several analogies to the structure of the human memory as we explicitly partition knowledge between short- and long-term memory. The intuitive combination of those memories as well as the explicitly preserved consistency are the main novelties of this contribution. SAM can be easily applied in practice without any parametrization, because hyperparameters are avoided through dynamic adaptation of various aspects guided by error minimization.

To enable a proper analysis of algorithms in terms of suitability for certain forms of drift, we do not only contribute new artificial benchmarks with ground truth drift type but also vision-based real-world datasets recorded outdoors: an environment where concept drift is naturally present. SAM is mostly evaluated in combination with the kNN classifier, but its flexibility is also showcased by coupling it with NB for the domain of text classification. The extensive evaluation on artificial and real-world benchmarks demonstrates the gain of SAM in comparison with state-of-the-art approaches. As the only method, it achieves highly competitive results throughout all experiments, demonstrating its robustness and the capability of handling heterogeneous concept drift.

4.4.1 *Architecture*

In the research field of human memory the dual-store model (Atkinson & Shiffrin, 1968), consisting of the Short-Term and Long-Term memory (STM & LTM), is largely accepted. Sensory information arrives at the STM and is joined by context relevant knowledge from the LTM. Information getting enough attention by processes such as active rehearsal is transferred into the LTM in form of Synaptic Consolidation (Dudai, 2004). The capacity of the STM is quite limited and information is kept up to one minute, whereas the LTM is able to preserve it for years (Miller, 1956). Immediate processing e.g. remembering the beginning of a read sentence, largely uses the STM. Whereas knowledge recalled from the past either explicitly requires the LTM e.g. consciously remembering events in life, or in an implicit way e.g. how to ride a bike.

The SAM architecture is partly inspired by this model and exhibits the following analogies:

- Explicit separation of current and past knowledge, stored in dedicated memories.
- Different conservation spans among the memories.
- Transfer of filtered knowledge from the STM to the LTM.
- Situation dependent usage.

The basic idea is to combine dedicated models for the current concept $P_t(\mathbf{x}, y)$ and all former ones $P_{t-1}(\mathbf{x}, y), \ldots, P_1(\mathbf{x}, y)$ in such a way that the prediction accuracy is maximized. This is a very general concept, which could be coupled with different type of models, requiring a different realization. In the context of this thesis, the SAM architecture is showcased mainly with kNN, as a non-parametric model, but also with the parametric model of NB. Two different memories are constructed: The Short-Term Memory (STM), containing data of the current concept, and the Long-Term Memory (LTM), maintaining knowledge of past concepts. Figure 4.7 illustrates this approach. SAM shares the common assumption of new data being more relevant for current predictions.

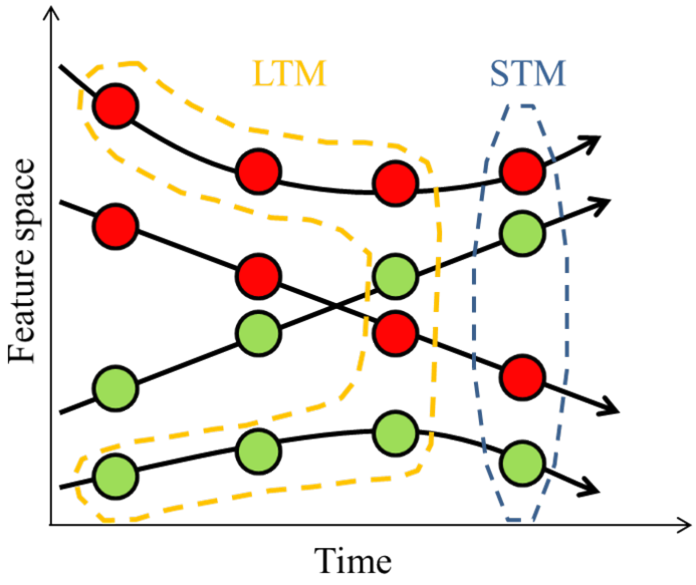

Figure 4.7: Illustration of the general approach. Each column represents one concept, different colors encode different classes. The STM contains only the current concept, while the LTM preserves only knowledge which is consistent in regard to the STM.

Hence, information from former concepts which is in conflict with the one of the current concept is selectively removed, but we explicitly preserve the rest in compressed fashion. Hyperparameters are mostly avoided. Instead, parameters are dynamically tuned at various steps, guided by error minimization on the recent data. The architecture is depicted in Figure 4.8 and described in detail below.

Model definition

Memories are represented by sets M_{ST}, M_{LT}, M_{C}. Each memory is a subset in $\mathbb{R}^n \times \{1, \ldots, c\}$ of varying length, adjusted during the adaptation process. The STM represents the current concept and is a dynamic sliding window containing the most recent m examples of the data stream:

$$M_{\mathrm{ST}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{1, \ldots, c\} \mid i = t - m + 1, \ldots, t\}. \tag{4.2}$$

The LTM preserves all former information, which is not contradicting those of the STM, in a compressed way. In contrast to the STM, the LTM is neither a continuous subpart of the data stream nor given by exemplars of it, but instead a set of p points:

$$M_{\mathrm{LT}} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{1, \ldots, c\} \mid i = 1, \ldots, p\}.$$

The combined memory (CM) is the union of both memories with size $m + p$:

$$M_{\mathrm{C}} = M_{\mathrm{ST}} \cup M_{\mathrm{LT}}.$$

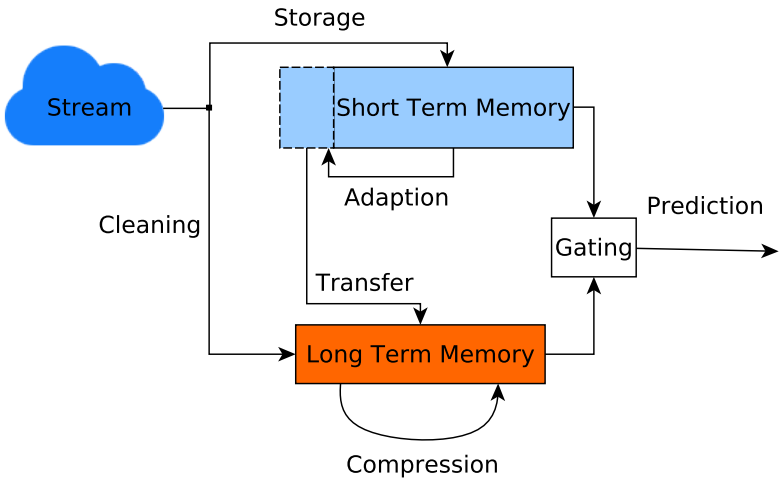

Figure 4.8: SAM architecture: Incoming examples are stored within the STM. The cleaning process keeps the LTM all-time consistent with the STM. Whenever, the STM is reduced in size, its discarded knowledge is transfered into the LTM. Accumulated knowledge is compressed each time the available space is exhausted. Both models are considered during prediction, depending on their past performances.

Every set induces a classifier, the SAM's default choice is distance weighted kNN : $\mathbb{R}^n \mapsto \{1,\ldots,c\}$, $\text{kNN}_{M_{\text{ST}}}$, $\text{kNN}_{M_{\text{LT}}}$, $\text{kNN}_{M_{\text{C}}}$. The kNN function assigns a label for a given point $\mathbf{x}$ based on a set$Z = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{1,...,c\} \mid i = 1,\ldots,m\}$:

$$\text{kNN}_Z(\mathbf{x}) = \arg\max_{\hat{c}} \{ \sum_{x_i \in N_k(\mathbf{x},Z) | y_i = \hat{c}} \frac{1}{d(\mathbf{x}_i, \mathbf{x})} \quad | \hat{c} = 1,\ldots,c \},$$

where $d(\mathbf{x}_1, \mathbf{x}_2)$ is the Euclidean distance between two points $\mathbf{x}_1$, $\mathbf{x}_2$ and $N_k(\mathbf{x}, Z)$ returns the set of k nearest neighbors of $\mathbf{x}$ in Z. Weights w_{ST}, w_{LT}, w_C are representing the accuracy of the corresponding model on the current concept and are determined as described in section 4.4.1.

The prediction of our complete model relies on the sub-model with the highest weight[5] and is defined for a given point $\mathbf{x}$ as:

$$\mathbf{x} \mapsto \begin{cases} h_{M_{\text{ST}}}(\mathbf{x}) & \text{if } w_{\text{ST}} \geq \max(w_{\text{LT}}, w_{\text{C}}) \\ h_{M_{\text{LT}}}(\mathbf{x}) & \text{if } w_{\text{LT}} \geq \max(w_{\text{ST}}, w_{\text{C}}) \\ h_{M_{\text{C}}}(\mathbf{x}) & \text{if } w_{\text{C}} \geq \max(w_{\text{ST}}, w_{\text{LT}}). \end{cases} \tag{4.3}$$

This model is adapted incrementally for every time t as described in section 4.4.1.

Model parameter

During the adaptation phase we adjust the following parameters:

- The size m of the STM.

5 In case of ties, we prioritize the models in the following order: $h_{M_{\text{ST}}}$, $h_{M_{\text{LT}}}$, $h_{M_{\text{C}}}$.

- The data points in the LTM.
- The weights w_{ST}, w_{LT}, w_{C}.

The model has the subsequent hyperparameters, which can be robustly chosen and do not require a task specific setting:

- The number of neighbors k.
- The minimum length $L_{\min}$ of the STM.
- The maximum number of stored examples $L_{\max}$ (STM and LTM combined).

We used for all experiments the same values which underlines the robustness of our approach. The number of neighbors was set to $k = 5$, a common default value in various implementations (Pedregosa et al., 2011). The minimum length was set to $L_{\min} = 50$ which is usually large enough to robustly estimate the current error on the one hand and still allows a quick reaction to drift on the other. $L_{\max} = 5000$ as a reasonable trade-off between a fast run-time / low memory consumption and a high degree of freedom for the approach to prove its qualities. Please note that a too restrictive size makes a precise evaluation of the system's adaptation capabilities impossible.

Model adaptation

The adaptation comprises every memory as well as the corresponding weights. We denote a data point at time t as $(\mathbf{x}_t, y_t)$ and the corresponding memories M_{ST_t}, M_{LT_t}, M_{C_t}.

Adaptation of the Short Term Memory

The STM is a dynamic sliding window containing the most recent examples. Every incoming example of the stream gets inserted such that the STM grows continuously. Its role is to exclusively contain data of the current concept. Therefore, its size has to be reduced whenever the concept changes such that examples of the former concept are dropped. However, we do not explicitly detect a concept change. Instead, we adjust the size such that the ITTE of the remaining STM is minimized. This approach relies on the fact that a model trained on internally consistent data yields fewer errors. We assume that the remaining instances represent the current concept or are sufficiently "close" to it.

Formally, we evaluate differently sized STMs and adopt the one with minimum ITTE E. We use bisection to compare only a logarithmic number of windows. Tested windows are:

$$M_l = \{(\mathbf{x}_{t-l+1}, y_{t-l+1}), \ldots, (\mathbf{x}_t, y_t)\},$$

where $l \in \{m, m/2, m/4 \ldots\}$ and $l \geq L_{\min}$.

$$M_{\mathrm{ST}_{t+1}} = \underset{S \in \{M_m, M_{m/2}, \ldots\}}{\arg\min} E(S).$$

Whenever the STM is shrunk, the set of discarded examples O_t is defined as

$$O_t = M_{\mathrm{ST}_t} \setminus M_{\mathrm{ST}_{t+1}}. \tag{4.4}$$

Instead of the commonly used cross validation error, we rely on the ITTE because it has various advantages in the streaming setting. The former is applied on random splits of the data and requires multiple repetitions to deliver a stable estimation of the error, which significantly increases the computational cost. Whereas the latter uses every example for test and training in the original order and therefore is the natural choice in the incremental learning setting.

Our way to adapt the size of the STM has similarities with the one of Klinkenberg and Joachims (2000). However, the approach is based on SVM specific estimates of the leave-one-out error and the authors do not indicate how to choose evaluated window sizes. In contrast, we propose to directly determine the interleaved test-train error on recursively bisected windows, which is applicable for arbitrary models.

Cleaning and Transfer

The LTM contains all data of former concepts that is consistent with the STM. This requires a cleaning of the LTM according to every seen example. Furthermore, whenever the STM is reduced in size, we do not simply discard the sorted out data, since it still may contain valuable information for future prediction. In case of reoccurring drift, methods preserving past knowledge do not have to relearn former concepts and therefore produce fewer errors.

Instead, we transfer as much knowledge as possible into the LTM. Before doing so, we delete examples from the separated set O_t (see Equation 4.4) which are contradicting those in $M_{ST_{t+1}}$. This adaptation is formalized by two operations.

1. Set A is cleaned by set B regarding an example $(\mathbf{x}_i, y_i) \in B$

$$\mathrm{clean} : (A, B, (\mathbf{x}_i, y_i)) \mapsto \hat{A}$$

where $A, B, \hat{A} \subset \mathbb{R}^n \times \{1, \ldots, c\}$ and $(\mathbf{x}_i, y_i) \in B$. $\hat{A}$ is defined in two steps.
(1) We determine the k nearest neighbors of $\mathbf{x}_i$ in $B \setminus (\mathbf{x}_i, y_i)$ and select the ones with label y_i. These define the threshold

$$\theta = \max\{d(\mathbf{x}_i, \mathbf{x}) | \mathbf{x} \in N_k(\mathbf{x}_i, B \setminus (\mathbf{x}_i, y_i)), \\ y(\mathbf{x}) = y_i\}.$$

(2) The k nearest neighbors of $\mathbf{x}_i \in A$ which are inconsistent to B are cleaned based on the threshold, yielding the result of this operation:

$$\hat{A} = A \setminus \{(\mathbf{x}_j, y(\mathbf{x}_j)) | \mathbf{x}_j \in N_k(\mathbf{x}_i, A), \\ d(\mathbf{x}_j, \mathbf{x}_i) \leq \theta, y(\mathbf{x}_j) \neq y_i\}.$$

2. Furthermore, we require a *cleaning operation* for the full set B

$$\text{clean} : (A, B) \mapsto \hat{A}_{|B|}$$

where $A, B, \hat{A}_{|B|} \subset \mathbb{R}^n \times \{1, \ldots, c\}$. This is defined iteratively by applying the former cleaning for all $(\mathbf{x}_i, y_i) \in B$ as

$$\begin{aligned} \hat{A}_0 &= A \\ \hat{A}_{t+1} &= \text{clean}(\hat{A}_t, B, (\mathbf{x}_{t+1}, y_{t+1})). \end{aligned}$$

The adaptation of the LTM takes place at two different steps. To ensure a consistent model at any time, cleaning takes place according to every incoming sample $(\mathbf{x}_t, y_t)$

$$\tilde{M}_{\text{LT}_t} = \text{clean}(M_{\text{LT}_t}, M_{\text{ST}_t}, (\mathbf{x}_t, y_t)).$$

Whenever the STM is shrunk, the discarded set O_t is transferred into the LTM after cleaning, i.e. the LTM becomes

$$M_{\text{LT}_{t+1}} = \begin{cases} \tilde{M}_{\text{LT}_t} \cup \text{clean}(O_t, M_{\text{ST}_{t+1}}) & \text{if STM is shrunk} \\ \tilde{M}_{\text{LT}_t} & \text{otherwise} \end{cases}$$

Compression of the LTM

In contrast to the FIFO principle of the STM, instances are not fading out as soon as the size limit of the LTM is reached. Instead, we condense the available information to a sparse knowledge representation via clustering. This enables a far longer conservation than possible with simple out fading. Formally, for every class label $\hat{c}$ we group the corresponding data points in the LTM

$$M_{\text{LT}_{\hat{c}}} = \{\mathbf{x}_i | (\mathbf{x}_i, \hat{c}) \in M_{\text{LT}}\}.$$

We use the clustering algorithm kMeans++[6] with $|M_{LT_{\hat{c}}}|/2$ clusters. The resulting prototypes $\hat{M}_{LT_{\hat{c}}}$ represent the compressed original data. The LTM is given by the union of all prototypes

$$M_{\text{LT}} = \bigcup_{\hat{c}} \{(\mathbf{x}_i, \hat{c}) | \mathbf{x}_i \in \hat{M}_{LT_{\hat{c}}}\}$$

This process is repeated each time the size limit is reached leading to a self-adapting level of compression.

Model weight adaptation

The weight of a memory is its accuracy averaged over the last m_t samples, where $m_t = |M_{\text{ST}_t}|$ is the size of the current STM. Hence, the weight of the LTM at time stamp t equals

$$w_{\text{LT}}^t = \frac{|\{i \in \{t - m_t + 1, \ldots, t\} \,|\, \text{kNN}_{M_{\text{LT}_i}}(\mathbf{x}_i) = y_i\}|}{m_t}$$

and analogous for STM and CM.

6 We used kMeans++ (Arthur & Vassilvitskii, 2007) because of its efficiency and scalability to larger datasets.

4.4.2 *Time Complexity*

The adaptation of the STM is by far the most time consuming process of SAM and determines the upper bound of the complexity. Therefore, we exclusively give a complexity analysis of this part and neglect the others.

Each time, we are evaluating up to $\log_2 \frac{L_{\max}}{L_{\min}}$ differently sized STMs, where $L_{\min}$ and $L_{\max}$ are the minimum and maximum lengths of the STM. The complete calculation of the error for one STM is upper bounded by $\mathcal{O}(L_{\max}^2)$. This results in the overall worst case complexity of $\mathcal{O}(nL_{\max}^2 \log_2 \frac{L_{\max}}{L_{\min}})$ for n examples. There is a lot of room to reduce this complexity, for instance, the calculation of the error can be done incrementally whenever the following condition is met:
Given an exemplary STM S_{t-1} (defined as in equation 4.2). If its successor S_t is simply an extension of S_{t-1}, such that $S_t = [S_{t-1}, (\mathbf{x}_t, y_t)]$, the corresponding error is given by

$$E(S_t) = \frac{(t-1)E(S_{t-1}) + \mathbb{1}(h_{t-1}(\mathbf{x}_t) \neq y_t)}{t}. \tag{4.5}$$

This is the case whenever an STM simply is growing by the current example, which happens frequently in practice and clearly dwindles the time complexity of the method.

4.4.3 *Speedup via Approximate ITTE*

In order to speed up the size adjustment of the STM, an *approximate incremental computation* of the ITTE, evaluating the performance of each candidate window, is proposed based on the observation 4.5. Essentially, all possible STM windows of different sizes resulting from the bisection are stored, including their predictions on the corresponding windows as well as the respective approximation of the ITTE. These values can efficiently be computed incrementally over the given stream.

More precisely, at time step t, the following ingredients are required: Let $m = m_t$ be the size of the current STM which is used in SAM. Denote the part of the data stream at time point t of length l as $M_l^t = [(\mathbf{x}_{t-l+1}, y_{t-l+1}), \ldots, (\mathbf{x}_t, y_t)]$ and the classifier induced by it as $h_{M_l^t}$. We store $\log_2 \frac{L_{\max}}{L_{\min}}$ STM candidates $\mathcal{M}_i$, enumerated by $i = 1, \ldots, \log_2 \frac{L_{\max}}{L_{\min}}$, where every candidate $\mathcal{M}_i$ carries the following information:

- A length parameter $m_t^i := \lceil m_t / 2^i \rceil$ corresponding to bisections of the current STM. This parameter uniquely induces the candidate memory $M_{m_t^i}^t$ as part of the current data stream and the respective classifier $h_t^i := h_{M_{m_t^i}^t}$. We use the convention that the memory is empty, i.e. no candidate STM is given, whenever $\lceil m_t / 2^i \rceil < L_{\min}$.
- A sequence of the last m_t^i predictions of the incremental classifier h_t^i given by its predecessor model in the preceding time steps: $\hat{Y}_t^i := [\hat{y}_j^i := h_{j-1}^i(\mathbf{x}_j) \mid j = t - m_t^i + 1, \ldots, t]$.

- The corresponding ITTE $\mathcal{E}_t^i := \frac{1}{m_t^i} \sum_{j=t-m_t^i+1}^{t} \mathbb{1}(\hat{y}_j^i \neq y_j)$.

$\mathcal{M}_i$ can be computed incrementally as follows: at time step $t+1$, the current STM is extended by the data point $(\mathbf{x}_{t+1}, y_{t+1})$ which requires the following updates of the candidate STMs:

- $m_{t+1}^i = \lceil (m_t + 1)/2^i \rceil$, where the corresponding start index can be computed as $(t + 1 - m_{t+1}^i + 1)$. As long as the STM is not reduced the length parameter of each candidate STM is growing. Hence, additional candidate STMs become available as soon as their length parameter surpasses $L_{\min}$.
- The sequence of prediction values $\hat{Y}_{t+1}^i$ can be obtained from $\hat{Y}_t^i$ by adding the prediction $\hat{y}_{t+1}^i := h_t^i(\mathbf{x}_{t+1})]$. In addition, whenever $\lceil (m_t + 1)/2^i \rceil = \lceil m_t/2^i \rceil$, the first entry $\hat{y}_{t-m_t^i+1}^i$ is deleted from the sequence.
- Accordingly, the induced ITTE $\mathcal{E}_{t+1}^i$ is updated on the basis of $\hat{Y}_{t+1}^i$.

These errors are incrementally determined, and, the current STM is replaced whenever a candidate STM yields a lower error. The replacement is also done in an incremental way by a re-enumeration of the given candidate STMs.

The complexity of one such incremental update is given by the effort of computing $\mathcal{O}(\log_2 \frac{L_{\max}}{L_{\min}})$ classifiers h, i.e. the effort is linear and no longer quadratic w.r.t. $L_{\max}$.

However, this computation only approximates the ITTE of the candidate STMs. Hence, one auxiliary step is necessary before actually substituting the STM by a new candidate. More precisely, assume that we are interested in the performance of candidate STM $\mathcal{M}_i$ with window $[(\mathbf{x}_{t-m_t^i+1}, y_{t-m_t^i+1}), \ldots, (\mathbf{x}_t, y_t)]$. The interleaved test-train error for models based on this stream has the form

$$E_t^i := \sum_{j=t-m_t^i+1}^{t} \mathbb{1}(h_{M_{j-1-t+m_t^i}^{j-1}}(\mathbf{x}_j) \neq y_j)/m_t^i \tag{4.6}$$

where $M_{j-1-t+m_t^i}^{j-1} = [(\mathbf{x}_{t-m_t^i+1}, y_{t-m_t^i+1}), \ldots, (\mathbf{x}_{j-1}, y_{j-1})]$.

$\mathcal{E}_t^i$ differs from E_t^i by referring to the model $h_{j-1}^i(\mathbf{x_j})$ based on the ith candidate STM at time step $j-1$, instead of the model $h_{M_{j-1-t+m_t^i}^{j-1}}(\mathbf{x}_j)$ which, in general, relies on a shorter part of the time window only. This difference is due to the fact that the start point of candidate STMs is shifted in time when processing the given data stream. Hence, in particular for stationary settings, the error $\mathcal{E}_t^i$ can underestimate E_t^i, and, in consequence, too short candidate STMs would be selected. The following validation step is performed to prevent such cases:

- Whenever a candidate STM $\mathcal{M}_i$ yields a smaller error than the current STM $\mathcal{M}_0$, we recompute the exact error E_t^j for all STMs

Table 4.5: The compared algorithms and the chosen hyperparameter. Ensembles consisted of 10 members. We used a maximum window sizes of 5000 and 1000 samples but never more than 10% of the whole dataset and k was set to 5 for all kNN-based methods.

Abbr.	Classifier	Parameter
L++.NSE	Learn++.NSE with CART	chunk-size = optimized
DACC	DACC with VFDT	$n = 10$
LVGB	LVGB with VFDT	$n = 10$
kNN_S	KNN with fixed size sliding window	$L_{\max} \in \{1000, 5000\}$, $k = 5$
PAW	kNN with PAW and ADWIN	$L_{\max} \in \{1000, 5000\}$, $k = 5$
SAM	Self adjusting memory with kNN	$L_{\max} \in \{1000, 5000\}$, $k = 5$

with index $j \geq i$, thereby referring to Eq 4.6. We use this opportunity to replace the approximated information of the corresponding candidate STMs with the actual ones, i.e. we set $\mathcal{E}_t^j = E_t^j$, $\hat{Y}_t^j = Y_t^j$ for all STMs with index $j \geq i$. Only if the novel computation confirms the quality of the candidate STM, the current STM is exchanged.

This recalculation is done only if a novel STM would be selected, hence the efficiency is hardly affected.

4.4.4 *Experiments - SAM-kNN*

We compare SAM with well-known state of the art algorithms for handling drift in streaming data. Thereby, we also utilize the previously obtained drift characteristics for the analysis. Initially, SAM is coupled with kNN. Evaluations regarding the NB model are given in Section 4.4.5. Implementations of the other algorithms were either obtained from the original authors or were already available in MOA. Table 4.5 gives an overview of the algorithms as well as the chosen hyperparameter. Apart from the methods already discussed in 4.2, we compare against a distance weighted kNN classifier with a sliding window of fixed size. Learn++.NSE is combined with Classification and Regression Trees (CART) (Breiman et al., 1984) as it is done by its author.

Window based approaches were allowed to store 5000 samples (we also report results for a size of 1000 samples) but never more than 10% of the whole dataset.[7] This rather large amount enables a high degree of freedom and prevents the concealment of their qualities with a too restricted window, in especially the capability of dynamically adjusting the window size is more transparent. The chunk size parameter of L++.NSE has to be predefined, which plays a similar role as the size of sliding windows. To avoid any disadvantage we evaluated several sizes and report the best result. No further dataset specific hyperparameter were optimized, since we wanted to provide as little prior knowledge as possible.

7 Regarding our approach, the available space is shared between the STM and LTM.

Table 4.6: Characteristics of the considered datasets. A=Abrupt, I=Incremental, R=Real, V=Virtual. The drift characteristics of the real-world data were obtained by the approach described in Section 4.3.

					Drift Characteristics			
Dataset	#Inst.	#Feat.	#Class	Type	Type	Pattern	Degree	Reocc.
SEA Concepts	50K	3	2	Art.	R	A	Low	No
Rot. Hyperplane	200K	10	2	Art.	R	I	Low	No
Moving RBF	200K	10	5	Art.	R	I	High	No
Inter. RBF	200K	10	15	Art.	R	A	High	No
Moving Squares	200K	2	4	Art.	R	I	High	Yes
Transient Chessb.	200K	2	8	Art.	V	A	High	Yes
Mixed Drift	600K	2	8	Art.	R,V	A,I	High	Yes
Poker	829K	10	10	Art.	V	A	High	No
Outdoor	4K	21	40	Real-world	V	?	High	?
Weather	18.1K	8	2	Real-world	V	?	Low	?
Electricity	45.3K	5	2	Real-world	R	?	High	?
Rialto	82.2K	27	10	Real-world	R	?	Low	?
Cover Type	581K	54	7	Real-world	R	?	Low	?

Datasets

Table 4.6 lists details of the evaluated datasets. The information about the drift characteristics of the real-world tasks were adopted from the experiments in Section 4.3. Regarding artificial data, commonly published benchmarks were considered, some of those were generated on the basis of Massive Online Analysis (MOA) (Bifet, Holmes, Kirkby, & Pfahringer, 2010) with common parametrizations. Four new datasets were added allowing an extensive evaluation on specific drift types, including reoccurring virtual drift, which is often ignored in the community. The dataset *Moving Squares* is particularly relevant to evaluate dynamic sliding window approaches because the optimal window size is predefined by the problem. In case of real-world data, the largest available benchmarks were useds. Furthermore, two new challenging datasets obtained from visual data are contributed (*Outdoor*, *Rialto*). All datasets were already introduced in context of the determination of the drift characteristics in Section 4.3. A detailed description of all datasetc can be found in Section A.2.

Exact- versus Approximate ITTE

Initially, we compare two versions of SAM. One adapts the STM based on the exact ITTE, whereas the other relies on the proposed approximation in Section 4.4.3. Thereby, $\text{SAM}_{\text{Exact}}$ denotes the exact version. Table 4.7 lists the corresponding results regarding the classification performance and the run time. Regarding the classification performance both approaches are delivering similar results without distinct differences. However, the run time of the approximation is on average clearly reduced. The specific speedup varies from task to task. It mainly depends on the average size of the STM, because the complexity of the exact calculation grows quadratically in respect to the size, whereas a linear dependence is given in case of the approximation. Hence, the

Table 4.7: Comparison between the exact calculation of the ITTE within the STM adaptation (SAM_{Exact}) versus the approximation (SAM) described in Section 4.4.3. The speedup mainly depends on the average size of the STM, which varies from task to task. The algorithm was allowed to store up to 5000 instances but never more than 10% of the whole dataset.

	ITTE		Run Time (s)	
Dataset	SAM_{Exact}	SAM	SAM_{Exact}	SAM
SEA Concepts	12.53	**12.27**	14.1	**12.4**
Rot. Hyperplane	**13.32**	13.42	**65.1**	73.0
Moving RBF	**15.25**	15.28	557.2	**133.9**
Inter. RBF	**5.61**	7.07	468.8	**88.9**
Moving Squares	**2.30**	2.65	35.8	**32.1**
Transient Chessb.	6.40	**6.37**	117.1	**45.0**
Mixed Drift	**13.47**	13.50	548.3	**180.0**
Poker	14.52	**14.26**	390.37	**360.8**
Artificial ∅	**10.43**	10.60	274.6	**115.8**
Outdoor	**11.32**	11.70	0.96	**0.5**
Weather	21.98	**21.86**	4.3	**3.0**
Electricity	**17.69**	17.70	**12.3**	**10.7**
Rialto	18.61	**18.59**	76.7	**41.4**
Cover Type	4.81	**4.80**	1058.1	**1035.5**
Real world ∅	**14.88**	14.93	230.5	**218.2**
Overall ∅	**12.14**	12.27	257.6	**155.2**

larger the STM the larger is the speedup gained by the approximation. Empirically, the approximation has only benefits without any drawbacks, therefore it is exclusively used within this thesis and referred to as SAM.

Results

The evaluation of the classification performance is done by measuring the ITTE. The error rates of all experiments are shown in Table 4.8. To test on significant differences among the methods, we rely on the proven approach suggested in (Demšar, 2006). Concretely, we use the non-parametric, rank-based Friedman test with $\alpha = 0.05$. If the null hypothesis is rejected, we proceed with the Nemenyi post-hoc test to identify the algorithms with significant differences.

The proposed method SAM outperforms the others quite significantly as it almost halves the average error rate of the second best method LVGB. Even more important is the fact that other methods struggle at one or another dataset but our approach delivers consistently robust results. All drift types are handled better or at least competitively. This is particularly clarified in the large gap achieved within the Mixed Drift dataset, which contains incremental, abrupt and virtual drift at the same time. SAM is the only method able to

Table 4.8: Interleaved Test-Train error (ITTE) rates of all experiments. Window-based approaches are considered with the maximum size of 5000 instances for the ranking. For each dataset the lowest value is marked in bold.

				Window size 5000			Window size 1000		
Dataset	L++.NSE	DACC	LVGB	kNN_S	PAW	SAM	kNN_S	PAW	SAM
SEA Concepts	14.48	15.68	**11.66**	13.83	12.39	12.27	13.96	12.79	13.27
Rot. Hyperplane	15.58	18.32	**12.73**	16.49	15.57	13.42	18.44	16.40	15.23
Moving RBF	44.50	54.29	45.62	20.29	25.51	**15.28**	12.89	19.38	12.13
Inter. RBF	27.52	**1.40**	7.14	39.67	8.60	7.07	10.15	6.57	3.09
Moving Squares	65.90	**1.15**	11.74	68.63	62.13	2.65	59.24	57.85	2.64
Transient Chessb.	**1.98**	41.87	14.69	6.19	19.61	6.37	13.72	21.31	11.27
Mixed Drift	40.37	62.11	25.97	28.78	28.88	**13.50**	20.27	25.68	12.26
Poker	22.14	20.11	17.93	17.08	31.62	**14.26**	17.08	31.11	16.89
Artificial ∅	29.06	26.87	18.44	26.37	25.54	**10.60**	20.72	23.89	10.85
Artificial ∅ rank	5.50	3.75	2.55	3.80	4.00	**1.40**	-	-	-
Outdoor	57.80	35.65	39.28	14.02	26.25	**11.70**	14.02	26.25	11.70
Weather	22.88	26.78	22.18	**21.53**	22.37	21.86	22.09	22.54	22.31
Electricity	27.24	**16.97**	17.58	26.59	25.37	17.70	23.08	23.52	17.58
Rialto	40.36	28.93	40.46	22.74	30.45	**18.59**	20.72	29.61	18.23
Cover Type	15.00	9.79	8.54	**4.21**	7.87	4.80	3.96	8.06	5.77
Real world ∅	32.66	23.62	25.61	17.82	22.46	**14.93**	16.77	22.00	15.12
Real word ∅ rank	5.67	3.83	4.17	2.17	3.50	**1.67**	-	-	-
Overall ∅	30.44	25.62	21.19	23.08	24.36	**12.27**	19.20	23.16	12.49
Overall ∅ rank	5.00	4.31	3.19	3.12	3.75	**1.62**	-	-	-

Nemenyi significance: SAM $\succ$ {L++.NSE, DACC, PAW}

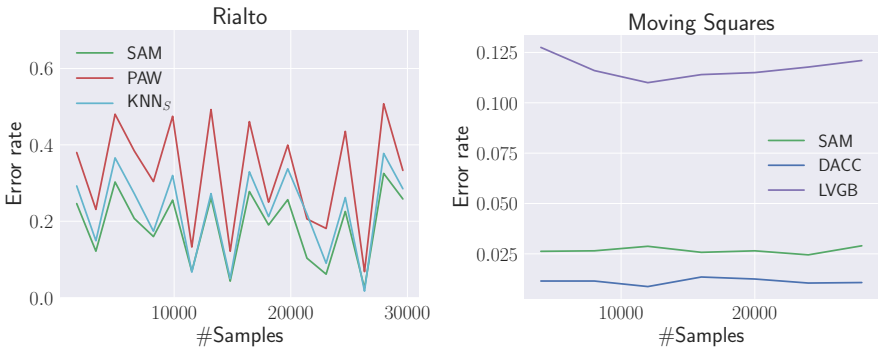

Figure 4.9: Error rate courses of the three best methods in the *Rialto* and *Moving Squares* datasets. The daily pattern of the *Rialto* dataset is reflected in the up and down of the error rates. Whereas, the continuous incremental drift in the *Moving Squares* dataset results in a rather constant performance.

perform significantly better than other algorithms according to the Nemenyi test. Concretely, the rank-based comparison attest SAM a statistical significant difference to the methods L++.NSE, DACC and PAW.

Figure 4.9 depicts for various datasets the development of the error rate, regarding the respectively best methods. The robustness of SAM

is highlighted by the fact that it is always within the three best methods. It performs best in the *Rialto* dataset. The daily pattern is reflected in the up and down of the error rates. During midday the task is comparably easy because the colors are clearly pronounced, whereas the rather blurry conditions of the morning and evening are clearly more challenging. SAM performs second best in the Moving Squares dataset, slightly worse than the best method DACC. Since $\text{kNN}_{\text{W}_\text{A}}$ also uses kNN and actively manages its window, it is closely related to SAM. However, it performs worse in all experiments. SAM uses a distance weighted kNN, whereas $\text{kNN}_{\text{W}_\text{A}}$ relies on uniform weights (majority voting). We evaluated SAM also with majority voting and got an overall average error rate of 14.72%, emphasizing that its advantage is due to the memory architecture.

For the sake of completeness, we also report the error rates of all window based approaches with a window size of 1000 samples as done in (Bifet et al., 2013). Especially the results of kNN_S and $\text{kNN}_{\text{W}_\text{A}}$ are significantly better with the smaller window. SAM also profits sometimes, e.g. for the *Moving RBF* dataset, albeit clearly weakened. The reason is that the smaller window conceals the issue that the methods sometimes fail to shrink the window appropriately. Samples of former concepts are fading out faster of the smaller window and are, therefore, less often contradicting the current concept in the case of real drift. We chose the larger and more informative size of 5000 samples for the ranking.

Our results confirm the fact that kNN is in general a very competitive algorithm in the streaming setting. It is quite surprising that the fixed sliding window approach kNN_S performs comparably well or even better than more sophisticated methods such as DACC or L++.NSE. The fixed sliding window approach only struggles with high real drift tasks and achieves competitive results in all other settings. This is especially emphasized by the second best average result for real-world datasets, which contain rather low degrees of real drift. Hence, this simple approach with a comparably low computation complexity may be a reasonable alternative to sophisticated drift algorithms in a variety of real-world tasks.

Memory Behavior

In this section, we illustrate the adaptation of the memories as well as their task specific roles. Figure 4.10 shows on the top left the size adjustment of the STM during the Interchanging RBF experiment. The algorithm reliably reduces the window size after each abrupt drift. However, we also observe a certain delay, during which wrong predictions are likely to occur. This delay is due to two reasons. Firstly, a certain amount of examples is required to reliably predict the new concept. Secondly, the more examples of the former concept are contained in the STM, the more stable is its representation and the more examples of the new concept are required to deteriorate the overall accuracy sufficiently. Hence, the delay illustrates a self-adjusting trade-off between adaptation speed to new concepts and the robustness

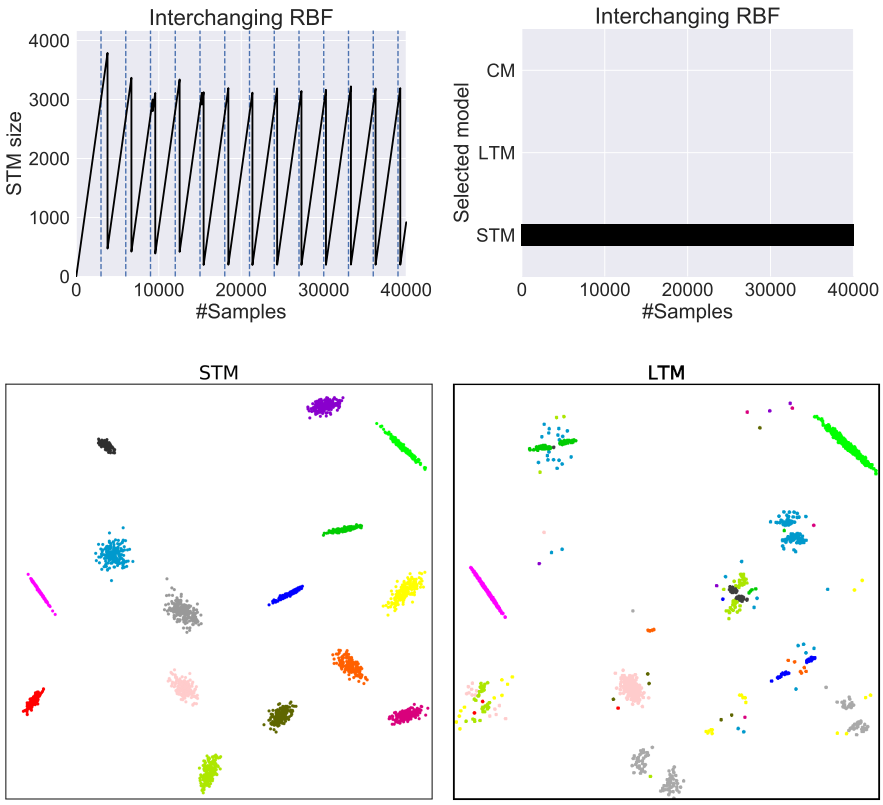

Figure 4.10: Illustrations of the Interchanging RBF dataset. Only a part of the dataset is depicted for the sake of clarity. Top: The size adaptation of the STM is shown on the left. Dashed vertical lines mark the moments of abruptly changed concepts. The delay of the model shrinks with increasing strength of drift. Only the STM is used for prediction (right). Bottom: Snapshots of the STM and LTM. Points of the LTM that are in accordance with the STM are preserved. Different classes are represented by different colors.

against noise, governed by the stability of both concepts. The adaptation delay decreases with increasing drift strength. The selective cleaning of the LTM is also visible in Figure 4.10. The empty spots within the clusters are due to the cleaning of contradicting instances. The remaining samples were harmless and consequently are kept in memory.

As already mentioned, the Moving Squares dataset is designed such that the squares may overlap each other if more than 120 of the recent examples are kept in memory. Hence, the best strategy for this problem is to keep the window as small as possible and to use only the most recent examples for prediction. Figure 4.11 shows how the size of the STM is mostly kept between 50 and 150 samples, allowing a nearly perfect prediction. This is also clarified by its snapshot, illustrating the absence of overlapping instances. The parameter controlling the minimum size of the STM prevents a size reduction below 50 examples.

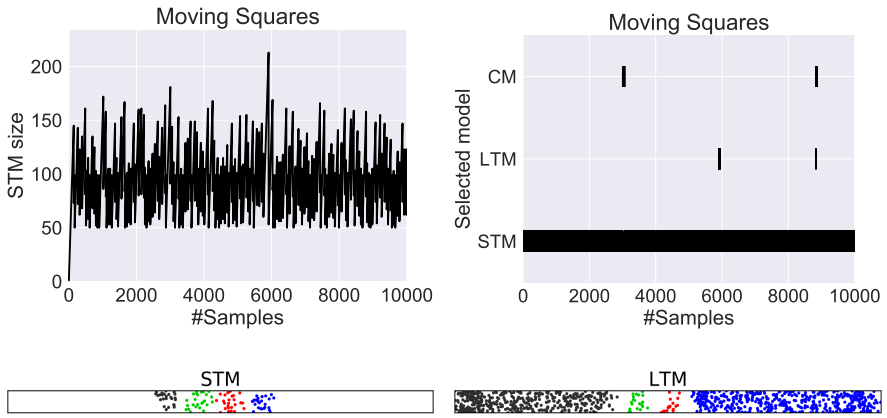

Figure 4.11: Depictions of the Moving Squares benchmark. Top: The STM size is most of the time kept below 120 samples and avoids the overlap of current points by outdated ones. The STM is exclusively used for prediction (right). Bottom: Snapshots of the STM and LTM. Classes do not overlap within both memories.

Otherwise, an even lower error rate could be achieved.

In contrast to the previous two datasets, which basically do not require the LTM, the LTM is crucial for the prediction in the Transient Chessboard task as it is illustrated by Figure 4.12. The STM is used alone whenever the data is presented square by square because it contains solely relevant information for the current square and produces less mistakes than in combination with the LTM. Whereas, during the periods in which examples are distributed over the whole board, the LTM is heavily used, since it contains beneficial information from the past. Its snapshot reveals the compressed preservation of the whole chessboard.

The task-dependent relevance of both memories is exemplified with real-world data in Fig.4.13: While the LTM is the preferred prediction model in the Weather dataset, it is the STM that classifies most of the instances of the *Rialto* task.

Impact of Clustering on Generalization

The class-wise clustering distinctly restructures the LTM. We empirically analyzed its impact on the generalization error. Precisely, we measured the error rate of the LTM for the subsequent 1000 samples of the stream before and after each clustering. This has been done for all datasets with total memory sizes of 5000 and 1000 samples (for the STM and LTM combined). It turns out that the generalization error is only slightly affected. Precisely, the average error rate of the LTM before and afterwards the clustering is 33.62% and 34.10% for the memory size of 5000 samples. The size of 1000 samples results in an error of 33.63% and 35.14%.

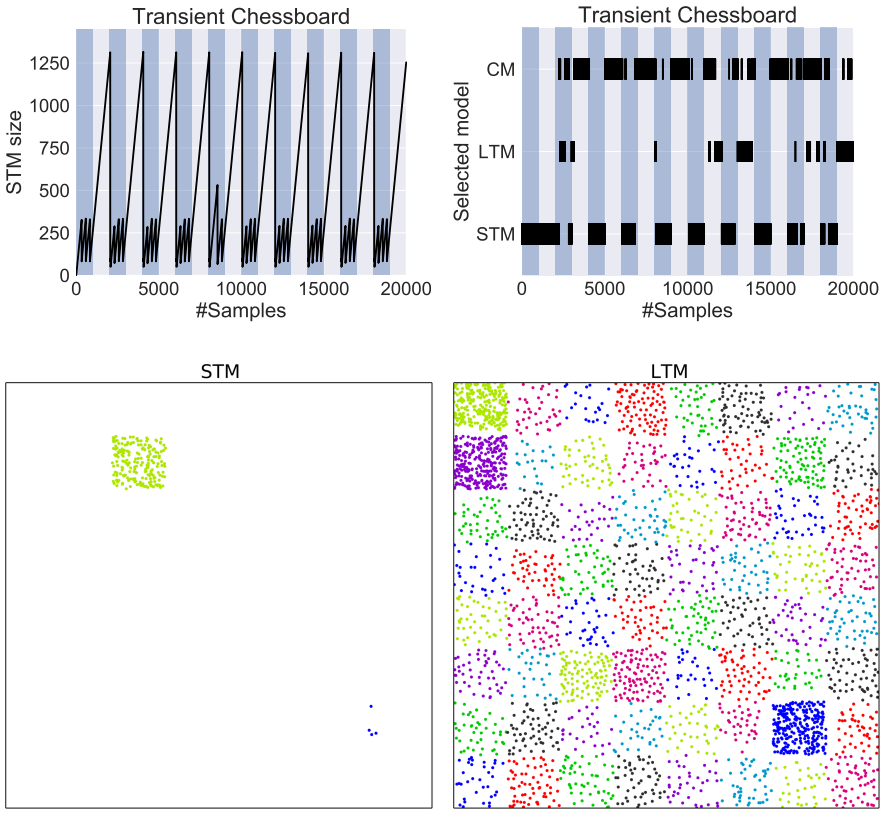

Figure 4.12: Illustrations of the Transient Chessbord dataset. Top: Each 1000 examples the square by square revelation (blue background) is alternated by samples covering the whole chessboard(white background). The STM tracks the current concept: It shrinks after each revealed field in the first phase and grows during the second phase to contain the whole chessboard. Only a part of the dataset is depicted for the sake of clarity. Bottom: The LTM preserves the whole chessboard in compressed fashion, whereas only the current concept is contained in the STM.

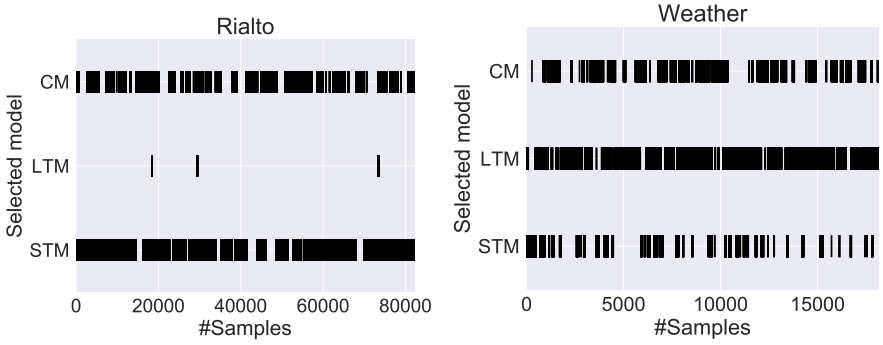

Figure 4.13: Model selection for the datasets Rialto (left) and Weather (right).

Table 4.9: The run time (s) of each algorithm. Window-based approaches were allowed to store up to $\{5000, 1000\}$ instances, but never more than 10% of the whole dataset. The time of L++.NSE is neglected because the Matlab-based implementation was far from being competitive in comparison to other methods which are all implemented in JAVA within MOA. For each dataset the lowest value is marked in bold.

			Window size 5000			Window size 1000		
Dataset	DACC	LVGB	kNN_S	PAW	SAM	kNN_S	PAW	SAM
SEA Concepts	**3.0**	3.7	15.5	58.9	12.4	8.8	20.8	5.8
Rot. Hyperplane	**22.2**	27.1	73.9	681.9	73.0	38.4	180.5	27.5
Moving RBF	**32.5**	42.5	74.5	690.4	133.9	38.5	179.0	38.1
Inter. RBF	26.5	**24.5**	62.2	53.0	88.9	34.8	37.8	31.3
Moving Squares	**11.0**	18.8	62.1	75.5	32.1	35.1	47.2	13.9
Transient Chessb.	**9.7**	15.1	62.5	34.0	45.0	34.8	28.8	19.4
Mixed Drift	**56.6**	62.7	189.9	248.8	180.0	105.7	140.4	85.2
Poker	**64.3**	83.0	330	768.3	360.8	185.2	532.1	102.1
Outdoor	4.4	8.0	0.8	2.8	**0.6**	0.7	3.4	0.6
Weather	**2.6**	2.7	3.7	17.1	3.0	3.4	11.6	3.0
Electricity	**3.8**	5.9	13.9	39.1	10.7	8.3	19.9	5.4
Rialto	**38.3**	43.5	48.7	91.3	41.4	19.1	76.5	18.4
Cover Type	211.6	**200.3**	658.3	2613.3	1035.5	224.4	1054.4	247.7
∅	**37.4**	41.4	122.8	413.4	155.2	56.7	179.4	46.1
∅ rank	**1.36**	2.14	3.64	4.57	3.29	-	-	-

Run Time

The run time of an algorithm is particularly crucial in the context of data-stream learning. Table 4.9 depicts the measured time in seconds. We ignored L++.NSE in this regard, since its Matlab-based implementation was not competitive in comparison to the JAVA-based implementation of the remaining algorithms.

Even though the tree ensembles DACC and LVGB consist in our case of 10 single classifiers, they are still clearly faster than singe kNN-based methods. This is due to their logarithmic query complexity as well as their natural compression scheme. However, SAM is only slightly slower in comparison to the simple kNN_S, highlighting that its partly complex building blocks such as the cleaning procedure or the STM adaptation are efficiently implemented. Concretely, the distances of the samples, which are anyways required for the kNN-based classification, are cached such that no additional distances have to be determined. It is noteworthy that the implementation of the nearest-neighbor search of PAW is comparably inefficient. PAW is a single kNN-based classifier which has rather a low overhead, but still requires a multiple of SAM's run time. As expected, the run time of kNN-based algorithms heavily depends on the size of the sliding window as it is highlighted by the distinctly quicker processing when the upper bound is set to 1000 instances. In this case, SAM's run time is comparable to the tree ensembles, but it still drastically outperforms them in terms of the classification accuracy (see Table 4.8).

Table 4.10: Characteristics of the considered datasets. A=Abrupt, I=Incremental, R=Real, V=Virtual

					Drift Characteristics	
Dataset	#Inst.	#Feat.	#Class	Type	Type	Pattern
SPAM	9.3K	40K	2	Real-world	?	?
20 Newsgroups	16K	62K	20	Real-world	-	-
20 Newsgroups Sorted	16K	62K	20	Real-world	V	A
20 Newsgroups Switched	48K	62K	20	Real-world	R	A

4.4.5 *Experiments - SAM-NB*

The SAM architecture permits a combination with different classifier models. However, the memories are frequently and selectively edited, hence models allowing incremental and decremental adaptations are crucial for a low time complexity. The NB algorithm is next to kNN a viable choice. NB is a classical linear algorithm, which is still very relevant, in particular for very high-dimensional data as commonly found in the domain of text classification. It allows an incremental and decremental update of its sparse model which is represented by the class priors $p(C_k), k \in 1, \ldots, k$ and the conditional probabilities $p(\mathbf{x}|C_k)$. Each SAM memory is coupled with one NB model denoted as $\text{NB}_{M_{\text{ST}}}$, $\text{NB}_{M_{\text{LT}}}$, $\text{NB}_{M_{\text{C}}}$. Changes on the memories are shadowed by incremental/decremental operations on the corresponding NB models. However, the cleaning procedure is yet based on kNN, because it is an open question how this can be done for non-local models as NB, which is not considered within this thesis.

Datasets

Table 4.10 depicts details of the considered text classification tasks, which are then briefly introduced.

SPAM The Spam Corpus dataset was developed by Katakis, Tsoumakas, Banos, Bassiliades, and Vlahavas (2009) on the basis of SpamAssassin[8] data collection covering an extended time period. The goal is to classify between spam or ham (not spam). Each attribute encodes the occurrence of one of the 39917 words within an e-mail.

20 Newsgroups The 20 newsgroups dataset comprises around 18000 newsgroups posts on 20 topics split. The goal is to assign each post to the corresponding category. Posts are encoded using the commonly applied term frequency–inverse document frequency (Salton & McGill, 1986). It is a very high-dimensional datasets with more dimensions than samples.

20 Newsgroups Sorted In this version of the 20 newsgroups data, abrupt virtual drift is created by sorting the instances by the class label. In terms of discrimination, this problem is particularly at

8 http://spamassassin.apache.org/

Table 4.11: Interleaved test-train error rates of different .

Dataset	NB	LVGB	LVGB-NB	SAM-kNN	SAM-NB
SPAM	2.81	7.35	**2.49**	7.00	3.21
20 Newsgroups	22.02	94.16	**21.69**	70.05	27.48
20 Newsgroups Sorted	14.30	22.59	14.30	7.83	**7.71**
20 Newsgroups Switched	70.44	94.7	**22.40**	72.00	41.81
∅	27.39	54.70	**15.22**	39.22	20.05
∅ rank	2.62	5.00	**1.62**	3.50	2.25

the beginning easier than the original one because the number of classes increases over time.

20 Newsgroups Switched The original input is repeated three times. After every iteration each class label is randomly mapped to another class which leads to abrupt real drift.

Results

Table 4.11 lists the results of SAM-NB in comparison to different other algorithms. LVGB-NB, which denotes the combination between LVGB and NB as underlying learning algorithm, achieves the best results. SAM-NB handles the virtual drift task very well, but particularly performs worse within the "20 Newsgroups Switched" task. This is probably due to its crucial cleaning operation which internally still relies on nearest neighbor. However, SAM-NB is still substantially better than ordinary NB, which cannot handle drift at all as it is mirrored within its error rates for both drift tasks. Our experiments confirm that the linear classifier NB performs very well in such high dimensional tasks, whereas more sophisticated methods as decision trees or kNN are struggling. The SPAM dataset seems to have a rather small amount of drift, since the common NB algorithm delivers a competitive performance.

4.4.6 *Discussion*

This section introduced the Self-Adjusting Memory (SAM) architecture, designed to handle heterogeneous concept drift within streaming data. SAM explicitly separates the current concept from former ones and preserves both in dedicated memories combining them according to the demands of the current situation. Thereby, it omits a common weakness of available methods that simply discard former knowledge and produce more mistakes in case of reoccurring drift. Our method is easy to use in practice, since it requires neither meta-parametrization nor related prior knowledge about the task at hand.

In comparison to state-of-the-art methods, SAM is the only algorithm which consistently achieved accurate results for heterogeneous drift types: virtual versus real drift pronounced in various patterns. The flexibility of the architecture allowed us not only a combination with kNN but also with the NB model, which is known to be highly

efficient in very high-dimensional data. We showed that the different memory types fulfill different functions in the overall model. While the STM represents the actual concept, the LTM conserves established knowledge as long as it is consistent with the STM. This decomposition proved to be particularly effective to simultaneously deal with heterogeneous drift types in real-world streaming data. Furthermore, the experiments highlighted that, apart from tasks with a high real drift degree, the simple fixed sliding window approach can be a reasonable alternative to more sophisticated drift algorithms.

The method can be extended in various way, as for instance considering a spacial decomposition to enable a more fine-grained combination of the memory models, enabling a reaction to different concurring drift types for different locations in space. The k parameter of kNN could be adapted as it is done in (Alippi & Roveri, 2008b) to improve the performance further. The SAM architecture can also be combined with other incremental / decremental methods such as ISVM. The cleaning operation is currently strictly coupled with the kNN model, which is sub-optimal when other models such as NB or ISVM are used, since an consistency between the memory is algorithm-specific. Hence, defining the cleaning operation in terms of the corresponding classifier model is beneficial for the overall system.

Currently, the adaptive compression in the LTM is based on unsupervised clustering. Taking into account the label information within this process enables a more specific compression in terms of especially considering the goal of accurate class discrimination.

Even though our memory architecture performs very robustly, there are some cases leading to unnecessary deletions within the LTM. For instance, a large amount of spread noise erases information from the LTM because of the assumption of new data being the most valuable one. Contrary concepts temporarily interchanging each other cause also deletions. This is due to the established consistency, which prevents the storage of contradicting concepts, and requires the relearning in case of such reoccurring patterns. This issue is tackled in terms of creating a diverse ensemble of SAMs as described in the next section.

4.5 SAM-ENSEMBLE (SAM-E)

Machine learning algorithms based on Bootstrap Aggregating (Bagging) (Breiman, 1996) such as the popular Random Forests (Breiman, 2001) are one of the most powerful state-of-the-art learning methods (Fernández-Delgado et al., 2014; Losing et al., 2018b). These ensembles aggregate weak learners and often drastically improve on their single performance. One intuitive explanation by Friedman (1997) reasons that Bagging reduces the overall variance while maintaining the bias. Ensembles are also very popular in the field of non-stationary stream learning because of their flexibility to selectively add and remove learners, enabling a simple and efficient way to handle concept drift. Furthermore, past concepts can be distributed among the learners and specifically applied in a suitable context.

In this section, we propose the SAM Ensemble (SAM-E) algorithm and combine the advantages of Bagging ensembles with the highly robust SAM algorithm to boost the performance further. Breiman pointed out in (Breiman, 1996) that Bagging requires a diverse ensemble to work which is only generated by unstable learners. Unstable in the sense that small variation in the training set lead to very different models. Here, we solely consider SAM in combination with the stable kNN algorithm. Hence, we induce randomization in multiple ways to increase diversity. Even though an ensemble of SAMs is able to deal with concept drift on its own, we also utilize a drift-detection on top, which triggers the replacement of the worst learners in case of a detection. This addition does not not only increase the adaptation speed of the overall model, but also selectively preserves learners with a suitable parametrization.

We provide a parallel implementation which clearly reduces the processing time. It is open source and will be integrated within the popular Massive Online Analysis (MOA) (Bifet, Holmes, Kirkby, & Pfahringer, 2010), facilitating the comparison for other researchers. On this basis, we perform an extensive and transparent evaluation on a large database of common artificial and real-world benchmarks, covering various types and rates of concept drift. All benchmarks are publicly available, providing transparent and easily reproducible results. We first analyze how each of the algorithmic building blocks contributes to the overall performance and diversity. Subsequently, we compare our approach with SAM itself as well as other state-of-the-art methods. SAM-E consistently achieves the best results, emphasizing that it does not only improve on the performance of SAM, but also clearly outperforms all other methods.

4.5.1 *Architecture*

The success of Bagging mainly depends on accurate and diverse learners h^i, in our case N SAM models SAM^i, $i = 1 \dots N$, as well as a proper aggregation of those (Breiman, 1996). Oza (2005) use for the Online Bagging a Poisson distribution with $\lambda = 1$ to determine the weight of each instance for each learner. In other words, at time step t, the ith learner, h_t^i, is adapted to the training sample (x_t, y_t) weighted by p, where p is a natural number distributed according to a Poisson distribution with parameter λ. In particular, the sample is dropped if $p = 0$. On average, 67% of the data are used by every learner if λ is set to 1. LVGB and ARF use $\lambda = 6$ where each learner gets 97% of the data and even more importantly with a distinctly higher weight. Even though this should lead to a lower diversity because all learners see more or less the same data instances, it mitigates the slow learning speed of the VFDT, and outweighs any disadvantage. We also use $\lambda = 6$, however we do not use it to improve the learning speed of kNN, since instance-based models have naturally a high learning speed. Instead, we target a quicker reaction to drift. In the proposed method, the learners mostly handle drift themselves and a low λ leads

to slower adaptation, since each of them has effectively less samples to react.

In contrast to the unstable decision tree algorithm, where Bagging alone creates enough diversity, kNN (and therefore SAM) is a stable method where additional diversity has to be induced. Hence, we randomize for each learner its k and subspace. The performance of kNN is known to depend on the selected k (Hastie, Tibshirani, & Friedman, 2001) and the Random Subspace Method (Ho, 1998) is a proven technique to increase diversity and generalization of ensemble methods (Ho, 1998). Concretely, when a new learner is generated we randomize it by two parameters $k, \hat{X}$, where k is drawn from a uniform discrete distribution, $k \sim \mathcal{U}(a, b)$ and the coefficients of $\hat{X} \in \mathbb{R}^{\hat{n}}$ are sampled with replacement from the original space $X \in \mathbb{R}^n$ where $\hat{n} = \lceil \beta n \rceil$ and $\beta \leq 1$. Varying the subspace and the neighborhood sizes does not only influence the classification itself, but it also affects the cleaning process of SAM, enabling the ensemble to store different types of concepts which might be used at different points in time. Each SAM model is weighted by its accuracy for the current concept, i.e. $w_i = max(w_{\text{ST}}, w_{\text{LT}}, w_{\text{C}})$. Formally, the prediction function is given as

$$\hat{y}_t = \underset{\hat{c} \in 1,\ldots,c}{\arg\max} \sum_{i=1}^{N} w_i * \text{SAM}^i(\mathbf{x_t}), \tag{4.7}$$

where SAM^i is the prediction function of the ith submodel (see Equation 4.3).

Even though each sub model is able to handle drift by its own, we add a drift detection on top of the ensemble, which initiates a selection process among the learners regarding their randomized model parameters k and *beta*. In particular, in the case of strong drift, the performance of the overall model is still affected, triggering a detection and the replacement of the worst performing learners, which has two main benefits. First, it mitigates the inert adaptation to concept drift, a natural consequence of model aggregation. Furthermore, it creates a competition within the ensemble where only members with a suitable parametrization for the current situation are lasting, which can be seen as an adaptive hyperparameter selection. Concretely, we replace a proportion r of the ensemble to be invariant to the ensemble size. SAM-E can be combined with arbitrary drift detection methods, we used the popular detector ADWIN (Bifet & Gavalda, 2007) in the experiments. An overview of the architecture is depicted in Figure 4.14 and the pseudo code is given in Algorithm 4.1.

4.5.2 *Parallel Implementation*

SAM is based on kNN and its complexity is dominated by the distance calculations. It has a complexity of $\mathcal{O}(wn)$ per instance where w is the average window size. SAM-E uses a subspace projection reducing the complexity to $\mathcal{O}(w\hat{n})$. In Bagging, learners are completely independent from each other, allowing a straight forward parallellization to

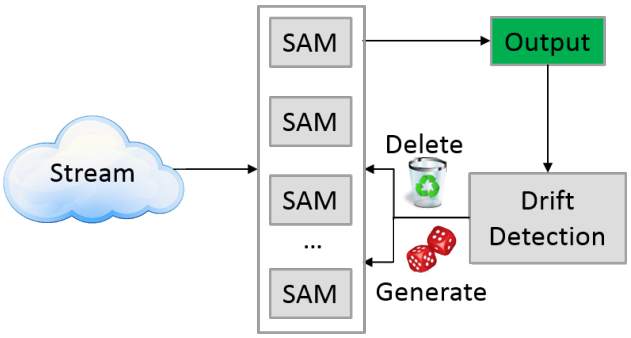

Figure 4.14: Architecture of SAM-E: The ensemble classifies the input of the incoming stream and its performance is monitored by the drift detection. In case of significant changes, the worst performing learners are replaced by new ones with a randomized configuration.

Algorithm 4.1 The SAM-E algorithm

Inputs:
S : data stream
N : ensemble size
a, b :bounds for randomization of k
β : relative size of the randomized subspace
r : proportion of replaced learners in case of detect drift
δ : drift detection sensitivity threshold
$\text{STM}_{\text{max}}, \text{LTM}_{\text{max}}$: maximum bounds for the STM and LTM of SAM

Initialize:
$C \leftarrow \text{CreateSAMs}(a, b, \beta, \text{STM}_{\text{max}}, \text{LTM}_{\text{max}}, N)$
$W \leftarrow \{1/N, \ldots, 1/N\}$

while S.hasNext() **do**
  $(x, y) \leftarrow S$.next()
  $\hat{y} \leftarrow \text{weightedPrediction}(x, C, W)$ *(Equation 4.7)*
  if $\text{driftDetected}(\delta, \hat{y}, y)$ **then**
    $C \leftarrow \text{replaceWorstClassifiers}(C, W, r)$
  $W \leftarrow \text{updateWeights}(C, y)$
  for all $i \in \{1, \ldots, N\}$ **do**
    $p \leftarrow \text{Poisson}(\lambda = 6)$
    if $p > 0$ **then**
      $C_i.\text{train}(x, y, p)$

reduce the computational costs, particularly considering the drastically increasing number of available cores in modern hardware. Therefore, a parallel implementation is provided and the gained speedup is analyzed in Section 4.5.4.

4.5.3 *Datasets*

We used a big variety of artificial and real-world benchmarks as illustrated by Table 4.12.

In the following, we briefly describe the datasets that were not introduced yet. More information about all datasets can be found in Section A.2.

Table 4.12: Characteristics of the considered datasets.

Dataset	#Inst.	#Feat.	#Class	Type
SEA Concepts	50K	3	2	Artificial
Rot. Hyperplane	200K	10	2	Artificial
Moving RBF	200K	10	5	Artificial
Inter. RBF	200K	10	15	Artificial
Moving Squares	200K	2	4	Artificial
Transient Chessb.	200K	2	8	Artificial
Random Tree	200K	200	25	Artificial
LED-Drift	200K	24	10	Artificial
Mixed Drift	600K	2	8	Artificial
Poker	829K	10	10	Artificial
Outdoor	4K	21	40	Real-world
Spam	9.3K	40K	2	Real-world
Weather	18.1K	8	2	Real-world
Electricity	45.3K	5	2	Real-world
Rialto	82.2K	27	10	Real-world
Airline	539.3K	7	2	Real-world
Cover Type	581K	54	7	Real-world
PAMAP	2.7M	52	18	Real-world
KDD99	4.9M	41	23	Real-world

Physical Activity Monitoring (PAMAP) This activity recognition task includes eighteen different activities performed by up to nine different subjects and comprises ten hours of recorded data in total (Reiss & Stricker, 2012). The features are obtained from three inertial measurement units (IMU) and a heart rate monitor. The IMUs have a sampling rate of 100Hz and are located on the chest, the dominant wrist and ankle. It is a frequently used benchmark dataset (Reyes-Ortiz, Oneto, SamÃ, Parra, & Anguita, 2016; Ordóñez & Roggen, 2016; Gan & Tao, 2015).

KDD99 This intrusion detection dataset is well known and widely used to analyze the performance of data stream learning algorithms (Amini, Wah, & Saboohi, 2014; Aggarwal, Han, Wang, & Yu, 2003). It simulates different types of cyber attacks and provides the highly imbalanced data in temporal order. Seventeen different classes are incorporated within the 5 million instances which are encoded with 41 attributes.

4.5.4 *Experiments*

Initially, different configurations of SAM-E are tested where the degree of randomization is varied to investigate its effects on the classification performance and diversity. Afterward, the parallel and the sequential implementation are evaluated in terms of run time and Ram-hours

Table 4.13: The compared variations of the algorithm with different degree of randomization as well as with drift detection and without.

Abbr.	Randomizing K	Feature Subspace	Drift detection
SAM-E_{None}	✗	✗	✗
SAM-E_{k}	✓	✗	✗
SAM-$E_{k,f}$	✓	✓	✗
SAM-$E_{k,f,d}$	✓	✓	✓

(Bifet, Holmes, Pfahringer, & Frank, 2010). Finally, SAM-E is compared with other state-of-the-art methods in detail.

As in Section 4.4.4, the rank-based Friedman test as well as the Nemenyi post-hoc test (Demšar, 2006) are used to analyze for statistical significant differences among the methods. All experiments are executed within MOA to create possibly fair conditions in terms of used evaluation scheme and programing language. We use a cluster consisting of 24 Intel Xeon 2.60GHz cores with 32 GB RAM to perform the experiments. SAM-E is configured as follows for all experiments:

- k is uniformly drawn from the range $[1, \ldots, 7]$.
- The random subspace of each learner $\hat{X} \in \mathbb{R}^{\hat{n}}$ is set to use 70% of the original number of features, i.e. $\beta = 0.7$.
- ADWIN is used as drift detection algorithm with its default settings.
- Each time ADWIN detects a drift 10% of the worst performing learners are replaced.

Variations of SAM-E

We analyze the effects of the suggested randomizations and the top-level drift detection on the classification performance. Table 4.13 lists all compared versions of SAM-E and the corresponding error rates are given in Table 4.14. Randomizing the k clearly improves the performance in comparison to Bagging alone. In general, a higher randomization leads to a higher performance in our experiments. However, the optimal degree of randomization clearly depends on the size of ensemble. The larger the ensemble the more randomness is beneficial. Hence, it is to expect that the performance of substantially larger ensembles could be further improved with more randomization such as a further reduced set of features.

The low error-rate for *LED-Drift* of SAM-E variants incorporating random subspaces suggests that it mitigates the susceptibility to noisy dimensions, one major weakness of kNN models. The comparison of SAM-$E_{k,f,d}$ and SAM-$E_{k,f}$ allows the evaluation of the drift detection on top of the ensemble, which is nearly always beneficial and in case

Table 4.14: Interleaved Test-Train error rates of different variations of SAM-E with an ensemble size of $n = 10$. The best results are marked in bold.

Dataset	SAM-E_{None}	SAM-E_{k}	SAM-$E_{k,f}$	SAM-$E_{k,f,d}$
SEA Concepts	12.61	12.31	12.28	**12.28**
Rot. Hyperplane	13.12	13.49	**12.41**	12.49
Moving RBF	12.02	**11.47**	11.98	11.86
Inter. RBF	3.32	**3.08**	3.37	3.30
Moving Squares	**2.41**	3.12	2.47	2.47
Transient Chessb.	10.92	10.5	**10.08**	10.30
Random Tree	35.36	34.47	32.72	**32.72**
LED-Drift	43.22	43.46	37.52	**35.48**
Mixed Drift	11.98	11.58	11.6	**11.58**
Artificial ∅	16.11	15.94	14.94	**14.72**
Artificial ∅ rank	3.33	2.72	2.17	**1.78**
Outdoor	11.02	**8.48**	8.98	8.98
Weather	21.91	**21.6**	21.81	21.81
Electricity	17.41	**15.36**	16.66	16.36
Rialto	18.06	**15.65**	16.64	15.80
Airline	39.12	38.88	37.53	**35.51**
Cover Type	5.66	**3.78**	8.1	4.69
Poker	15.82	12.59	15.03	**8.79**
PAMAP	0.02	0.02	0.02	**0.02**
SPAM	6.67	**5.37**	5.38	5.61
KDD99	0.01	0.01	0.01	**0.01**
Real world ∅	13.57	12.17	13.02	**11.76**
Real world ∅ rank	3.6	**1.6**	2.7	2.1
Overall ∅	14.77	13.96	13.93	**13.16**
Overall ∅ rank	3.47	2.13	2.45	**1.95**

Nemenyi significance: $\{\text{SAM-E}_{k}, \text{SAM-E}_{k,f,d}\} \succ \text{SAM-E}_{None}$

of *Poker* it nearly halves the error-rate. This is due to the increased variability of the algorithm in case of drift, enabling a faster adaptation to the current concept. The versions SAM-$E_{k,f,d}$ and SAM-E_{k} even deliver statistically significantly better results in comparison to SAM-E_{None}.

Figure 4.15 depicts on the left the temporal course of the error-rate for some datasets. In the most cases, the advantage of SAM-$E_{k,f,d}$ compared to the other variations increases over time. Corresponding kappa-error diagrams (Margineantu & Dietterich, 1997) are shown on the right of Figure 4.15. These diagrams are commonly used to inspect diversity of ensembles. The pairwise kappa statistic is plotted against the classification performance of the learners. Highly diverse learners (low pairwise kappa-statistic) with a low error rate are essential for an performance improvement on the basis of ensembles. Most of the time SAM-$E_{k,f,d}$ is able to achieve that better than others. One exception is the *Electricity* task, where it has a higher diversity in comparison to SAM-$E_{k,f}$, however, the error-rate of the single learner is also higher, resulting in an overall lower classification performance. The relatively high kappa statistics of SAM-E_{None} attest a low diversity, which limits

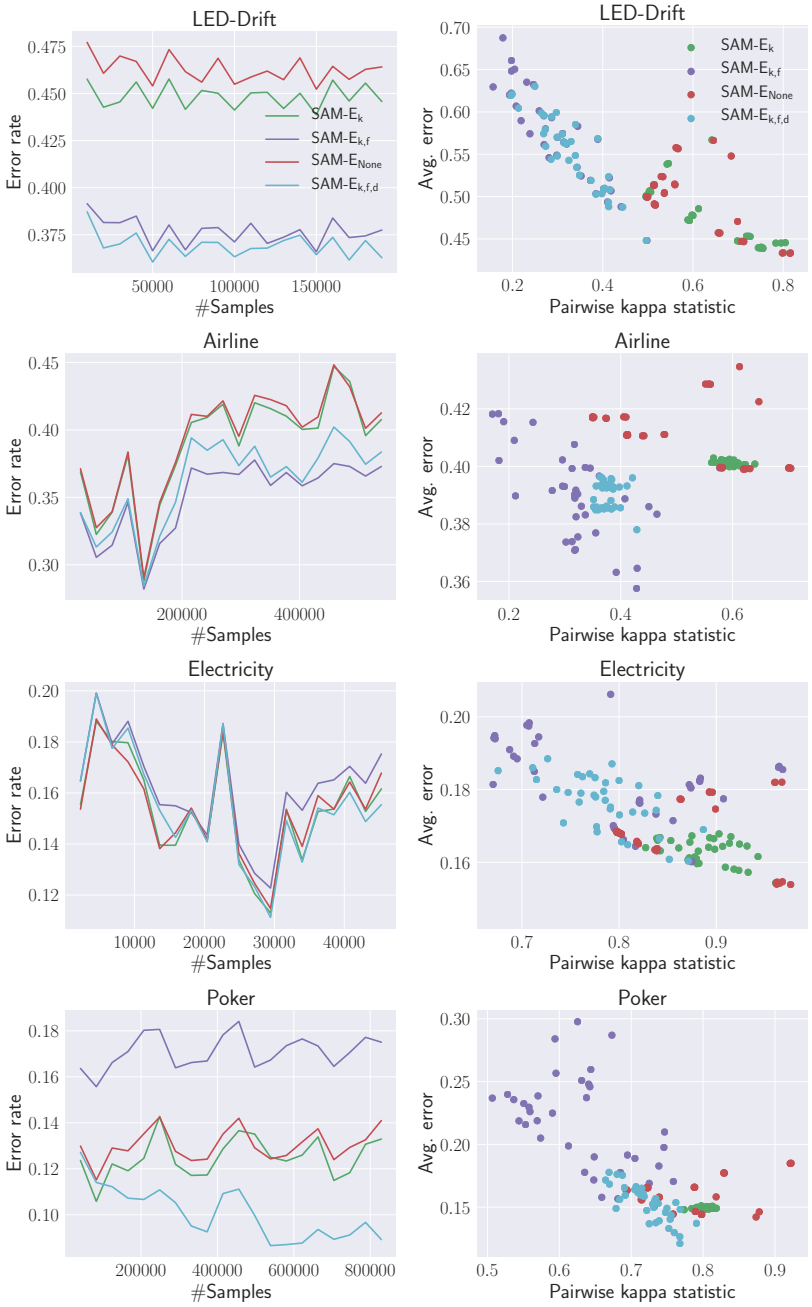

Figure 4.15: The temporal course of the error rate (on the left) as well as corresponding kappa-error diagrams (on the right). High diversity coupled with a low error-rate result in the best ensemble classification performance.

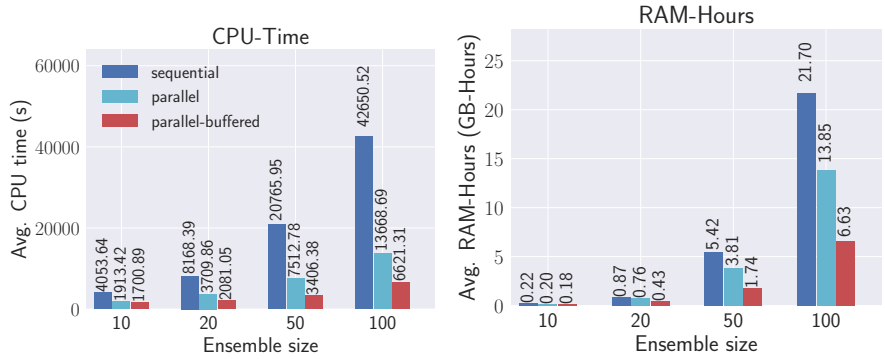

Figure 4.16: Comparison of the parallel and sequential implementation in terms of average run time and RAM-hours. The gained speed-up also reduces the RAM-hours even though more memory is needed due to the multi-threading overhead. Relaxing the test-train scheme to buffering 100 instances clearly increases the gain further.

the leeway to reduce the error rate beyond those of the single classifiers. One example is the *LED-Drift* dataset where its learners have the lowest error rate, but the classification error of the ensemble is comparatively high. The replacement of the worst learners triggered by the drift-detection decreases the error-rates of the learners which is especially pronounced for the tasks *Airline* and *Poker*.

Due to these results, further experiments are solely done by the variant SAM-$E_{k,f,d}$ which we refer to as SAM-E.

Sequential and Parallel implementation

The utilized resources of the sequential and parallel implementation are measured in terms of the average run time and RAM-hours for all datasets. One RAM-hours equals one GB of RAM deployed for one hour (Bifet, Holmes, Pfahringer, & Frank, 2010). We provide results for different ensemble sizes. The interleaved test-train processing enforces the aggregation after each example and especially a completed training of the previous instance. Therefore, the threads are very short and create a comparably large overhead. However, this scheme is only performed for the evaluation. In practice, instances can easily be buffered to small chunks and processed at once. We also consider this processing scheme and buffer chunks of 100 instances. Figure 4.16 depicts the results. The parallel implementation halves the run time for an ensemble size of 10 and achieves a speedup of 3 for the largest tested ensembles with $n = 100$. The buffering mechanism is very effective and doubles the gained speed-up, underlining the benefits of a parallel implementation. Even though parallel processing allocates more memory because to the multi-threading overhead, the drastically reduced run-time makes more than up for it and overall less RAM-hours are required.

Table 4.15: The compared algorithms and the chosen hyperparameter. Ensembles were evaluated with 10 and 100 members. We used a maximum window size of 5000 samples and k was set to 5 for all kNN based methods.

Abbr.	Classifier	Parameter
VFDT	Hoeffding Tree (VFDT)	-
PAW	Probabilistic Adaptive Window with kNN	$w = 1000$
SAM	Self-Adjusting Memory with kNN	$w = 1000$
LVGB	Leveraging Bagging with VFDT	$n = \{10, 100\}$
ARF	Adaptive Random Forest with VFDT	$n = \{10, 100\}$
SAM-E	Self-Adjusting Memory Ensemble	$n = \{10, 100\}, w = 1000$

Comparison with State-of-the-Art Methods

We compare SAM-E not only to other ensemble methods specialized for data stream learning with concept drift, but also include a comparison to single classifiers such as SAM itself, PAW and VFDT. Table 4.15 lists all algorithms as well as important hyperparameter settings. All hyperparameters which are not explicitly listed, such as the split confidence of the VFDT or the k of SAM are simply set to the default values of the respective original publication[9]. Sliding-window approaches (SAM, PAW, SAM-E) were allowed to store 1000 samples but never more than 10% of the whole dataset.

Classification Performance

The error rates are listed in Table 4.16. SAM-E achieves the best classification performance on average. The fact that the single SAM algorithm delivers the second best results highlights in general the superiority of the architecture within this domain. Apart from the task *Interchanging RBF*, SAM-E is always better than the single SAM algorithm and in case of *Random Tree*, *LED-Drift*, *Outdoor*, and *Poker* quite distinctly. To the best of our knowledge, SAM-E achieves the lowest error rate ever reported for the frequently evaluated *Poker* benchmark. The two comparably poor performances of SAM-E in the tasks *Random Tree* and *LED-Drift* are due to the susceptibility of kNN based approaches in regard to noisy dimensions. Incorporating metric learning or dimensionality weighting techniques such as (Jain, Kulis, Dhillon, & Grauman, 2009) would likely lead to further improvements. However, the random subspaces enable SAM-E to perform clearly better in these cases compared to the single SAM algorithm. Our method even outperforms VFDT and PAW with statistical significance.

Both tree ensembles ARF and LVGB deliver several times drastically worse results, mostly due to their comparably slow learning speed (*Outdoor*) or limited adaptation ability in case of incremental fast drift (*Moving Squares*). Furthermore, they do not have an explicit mechanism to deal with reoccurring drift as shown by the results for

9 The hyperparameter setting of the algorithms used in the original publication are usually set as default values in MOA.

Table 4.16: Interleaved Test-Train error rates achieved by single classifiers and ensemble models. Ensembles consisted of $n = 10$ members. The best results are marked in bold.

Dataset	VFDT	PAW	SAM	ARF	LVGB	SAM-E
SEA Concepts	15.16	13.35	13.22	11.68	**11.68**	12.28
Rot. Hyperplane	15.02	18.02	15.22	17.35	12.73	**12.49**
Moving RBF	66.27	17.21	12.10	34.02	45.62	**11.86**
Inter. RBF	74.71	5.79	3.27	**2.68**	10.08	3.30
Moving Squares	66.73	56.72	2.64	36.84	11.74	**2.47**
Transient Chessb.	45.24	17.16	11.26	26.30	14.69	**10.30**
Random Tree	10.36	23.81	37.05	**5.78**	3.93	32.72
LED-Drift	26.30	35.30	45.99	27.39	**26.13**	35.48
Mixed Drift	55.42	38.18	12.27	19.87	25.97	**11.58**
Artificial ∅	41.69	25.06	17.00	20.21	18.06	**14.72**
Artificial ∅ rank	4.89	4.44	3.33	3.17	2.83	**2.33**
Outdoor	42.68	16.88	11.58	61.52	39.28	**8.98**
Weather	26.49	23.40	22.31	21.87	22.18	**21.81**
Electricity	29.00	23.05	17.58	21.13	17.58	**16.36**
Rialto	76.19	24.07	18.27	27.20	40.46	**15.80**
Airline	34.94	33.75	39.84	**34.20**	36.89	35.51
Cover Type	21.85	6.50	5.76	8.33	8.54	**4.69**
Poker	25.88	31.07	16.86	19.23	17.93	**8.79**
PAMAP	1.22	0.03	0.02	0.03	0.11	**0.02**
SPAM	19.09	16.43	7.00	8.18	7.35	**5.61**
KDD99	0.10	0.02	0.01	0.03	0.03	**0.01**
Real world ∅	27.74	17.52	13.92	20.17	19.04	**11.76**
Real world ∅ rank	5.5	3.75	2.55	3.8	4.	**1.4**
Overall ∅	34.35	21.09	15.38	20.19	18.57	**13.16**
Overall ∅ rank	5.21	4.08	2.92	3.5	3.45	**1.84**

Nemenyi significance: SAM-E $\succ$ {VFDT, PAW}

Transient Chessboard and *Rialto*. They excel at the *Random Tree* task, which is known to be rather easy for tree-based methods, since the task is to learn a tree-based model.
Figure 4.17 depicts the temporal course of the classification performance for some datasets as well as corresponding kappa-error diagrams. SAM-E achieves the best classification performance on the basis of more accurate base classifiers in comparison to ARF and LVGB. Its diversity is high and comparable to the tree ensembles, creating enough leeway for the the model aggregation to improve on the single learner performance.

Run Time

The measured run time is given in Table 4.17. Tree-based methods are most of the times clearly faster than those relying on kNN. Particularly, in case of high-dimensional data such as *SPAM*, the evaluation complexity $\mathcal{O}(\log n)$ of the decision tree versus $\mathcal{O}(n)$ of kNN leads to a distinct difference. Furthermore, the learning complexity of incremental decision trees depends on the current classification performance,

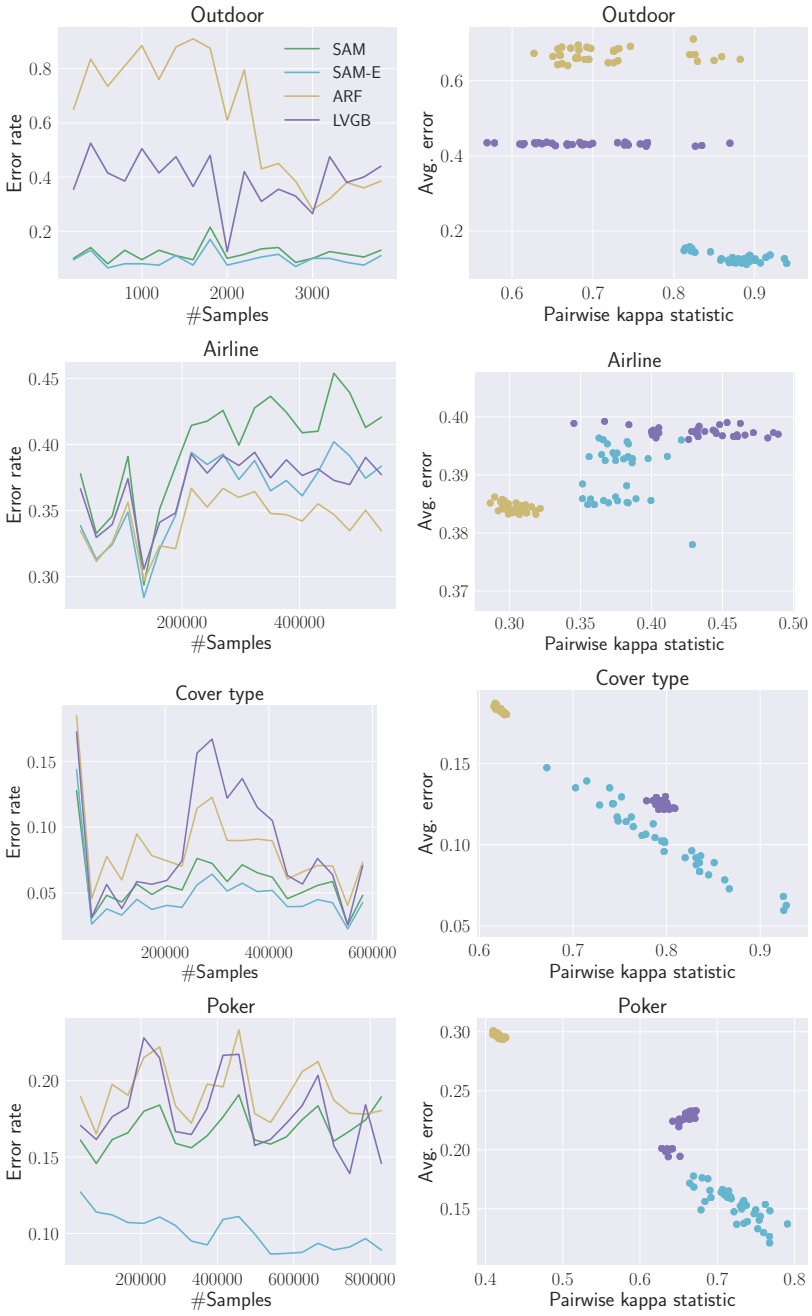

Figure 4.17: The temporal course of the error rate (on the left) as well as corresponding kappa-error diagrams (on the right). For the sake of clarity, only the four best methods are considered. SAM-E achieves the best classification performance on the basis of more accurate base classifiers in comparison to ARF and LVGB.

Table 4.17: The run times (s) of the experiments. Ensembles consisted of $n = 10$ members. The best results are marked in bold.

Dataset	VFDT	PAW	SAM	ARF	LVGB	SAM-E
SEA Concepts	**1.0**	19.0	4.7	8.0	3.9	20.6
Rot. Hyperplane	**3.8**	182.1	23.1	45.6	26.6	85.3
Moving RBF	**4.7**	180.9	35.0	37.6	41.6	135.8
Inter. RBF	**13.7**	103.5	42.2	58.8	171.8	149.1
Moving Squares	**2.0**	48.2	11.7	44.1	20.2	59.2
Transient Chessb.	**2.4**	26.0	16.5	31.8	15.7	75.3
Random Tree	**4.8**	131.6	17.5	25.2	30.5	77.8
LED-Drift	**3.9**	154.0	59.9	26.3	32.3	190.8
Mixed Drift	**6.5**	471.2	69.4	102.5	62.4	463.3
Artificial$\sum$	**42.7**	1316.5	280.1	379.9	404.9	1257.0
Outdoor	0.9	2.5	**0.7**	3.4	8.7	1.8
Weather	**0.5**	10.2	2.5	4.1	2.9	9.1
Electricity	**1.1**	18.4	4.4	11.2	5.7	19.2
Rialto	**4.4**	64.9	17.4	27.8	42.7	68.3
Airline	**8.6**	316.2	37.0	322.3	598.3	235.8
Cover Type	**18.8**	788.4	201.9	148.2	191.6	1195.7
Poker	**10.6**	485.7	91.1	204.8	86.7	502.0
PAMAP	**144.8**	12534.1	1223.3	288.0	390.9	8506.9
SPAM	**266.8**	48889.9	2142.9	2737.0	1510.3	15651.0
KDD99	**119.0**	22757.6	1520.6	456.5	511.8	8908.3
Real world $\sum$	**575.4**	85867.8	5241.7	4203.1	3349.6	35097.9
Overall $\sum$	**618.1**	87184.3	5521.8	4583.1	3754.5	36355.0

whereas those of NN-methods is constant. A decision tree is only growing in case of ambiguous labels within the leaves, therefore tasks with a high classification performance such as *PAMAP* or *KDD* cause a very slow growth and quick processing. However, large and noisy datasets lead to endlessly growing trees, which eventually become slow and inefficient.

Overall, SAM-E requires approximately eight times as much processing time as the tree-based ensembles. This factor is halved, if the exceptionally easy tasks of *PAMAP* and *KDD* are not considered. Naturally, the complexity of sliding-window approaches depends on the window size. For example, SAM-E is faster for the task *Outdoor* than the tree-based methods because in this small task the window size is limited to 400 instances (400 instances correspond to 10% of the dataset).

As already noted in Section 4.4.4, the NN-search of PAW is inefficient compared to ours. It is a single kNN-based classifier and requires a multiple of the run time of the SAM-E ensemble which consists of ten SAM classifiers that additionally perform more complex processing steps such as the cleaning, the STM adaptation and LTM clustering.

Some exemplary developments of the run time are given in Figure 4.18. Even though the VFDT is the quickest method for the *Airline* dataset, LVGB has the highest run time due to its drift detection (ADWIN), which is created for each learner and dominates the processing time.

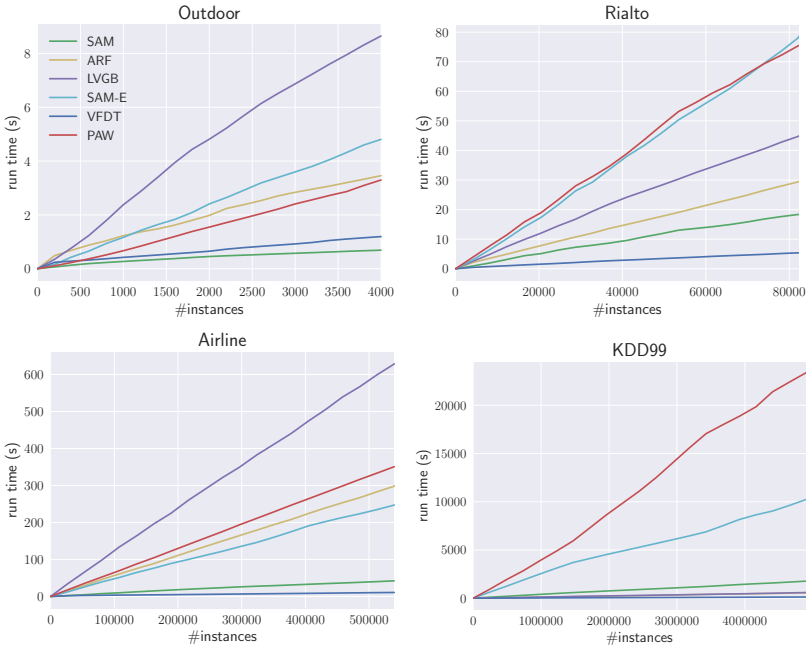

Figure 4.18: The development of the run time for some datasets. The complexity of the VFDT depends on the classification performance. Easy tasks such as *KDD99* lead to slowly growth of the trees, since most leaves have no ambiguous class label information. In contrast, the complexity of kNN-based approaches is independent from their error rate.

Scalability

The results achieved with 100 learners are given in Table 4.18. Increasing the number of members improves the performance of SAM-E further, even though only slightly. Noteworthy, LVGB performs worse with the larger ensemble because it only replaces one learner at a time in case of detected drift, which has a decreasing effect when the number of learners is increased. ARF has the largest performance gain due to its higher diversity (see Figure 4.17). However, increasing the size further does not lead to further improvement [10]. Overall, SAM-E remains dominant and even performs significantly better than LVGB.

4.5.5 *Discussion*

SAM-E combines the robust drift-handling algorithm SAM with the advantages of a highly diversified ensemble. The diversity is ensured by randomizing the feature space as well as the underlying kNN-search, which heavily affects SAM's cleaning operation enabling the preservation of partially contradicting concepts that might be useful at different points in time. Additionally, a drift detection, monitoring the performance of the ensemble, enables a faster adaptation in case

10 We repeated the experiments for ARF with $n = 250$, resulting in an average error rate of 18.53.

Table 4.18: Interleaved Test-Train error rates of the ensembles using $n = 100$ learners. The best results are marked in bold.

Dataset	ARF	LVGB	SAM-E
SEA Concepts	**11.39**	12.78	12.27
Rot. Hyperplane	15.27	12.74	**11.43**
Moving RBF	27.64	43.99	**11.48**
Inter. RBF	**2.67**	27.38	3.29
Moving Squares	34.38	33.24	**2.75**
Transient Chessb.	22.77	**2.5**	10.35
Random Tree	4.97	**3.74**	32.37
LED-Drift	27.1	**26.01**	34.49
Mixed Drift	17.35	22.24	**11.54**
Artificial ∅	18.17	20.51	**14.44**
Artificial ∅ rank	2.20	2.10	**1.70**
Outdoor	59.3	42.32	**9.02**
Weather	**20.93**	22.02	21.04
Electricity	19.74	18.84	**15.39**
Rialto	22.43	42.46	**15.54**
Airline	**33.3**	36.2	35.54
Cover Type	7.41	7.16	**4.75**
Poker	17.56	8.83	**8.72**
PAMAP	0.03	0.22	**0.02**
SPAM	7.4	8.89	**5.01**
KDD99	0.03	0.04	**0.01**
Real world ∅	18.81	18.70	**11.55**
Real world ∅ rank	2.11	2.67	**1.20**
Overall ∅	18.51	19.56	**12.92**
Overall ∅ rank	2.16	2.37	**1.47**
Nemenyi significance:	SAM-E $\succ$ LVGB		

of drift and explicitly filters learners with improper parametrization for the current situation. As SAM itself, our method is easy to use in practice, since the few hyperparameters can be robustly set in general without the necessity of dataset-specific tuning. We provide a parallel implementation which clearly reduces the processing time. It is open-source and will be integrated within the popular MOA framework[11], facilitating the comparison for other researchers. In the evaluation, we showed the effects of the different algorithmic building blocks and compared SAM-E with other state-of-the-art methods. It consistently outperformed all other approaches, some with statistical significance.

There are various ways to build on our work. Quite obvious extensions are the incorporation of online metric learning and / or feature drift approaches that can mitigate the typical susceptibility of kNN to scaling and noise; or approximate kNN-querying that can possibly dwindle computational load, enabling even larger ensembles with

11 https://moa.cms.waikato.ac.nz/

potential for more randomization.

A more profound extension is to develop the architecture towards a semantic or episodic memory. SAM-E was a first step to create more complex memory architectures as the projection in different sub-spaces and the varying granularity in which inconsistent information is removed generate memories with different views on the current situation. However, these memories are mainly based on randomization and have no specific meaning. It would be interesting to create diverse memories in a more purposeful way, in terms of a semantic representation, where memories explicitly capture different semantic concepts and may even act on varying time scales.

5 REAL-WORLD APPLICATIONS

Summary Three real-world applications of incremental learning are presented in this chapter. The tasks are completely different as they include object recognition, driver-intent prediction as well as rapid motion classification, highlighting the wide range of applicability. Thereby, we showcase previously proposed methods such as ILVQ or SAM. Another focus is to combine personalized learning with online adaptation, which is analyzed in the frame of two different scenarios.

Source Code

- Python sources of the Incremental Learning Vector Quantization (ILVQ) with the COst MinimizatiOn Sampling (COSMOS) (Losing et al., 2015) is available at https://github.com/vlosing/incrementalLearning.
- Original C++ sources of Online Random Forest (ORF) (Saffari et al., 2009) are provided at https://github.com/amirsaffari/online-random-forests.
- The Outdoor dataset is available at https://github.com/vlosing/incrementalLearning. Due to privacy issues, data of the driver maneuver prediction as well as human motion recognition cannot be published.

Parts of this chapter are based on:

- Losing, V., Hammer, B., & Wersing, H. (2015). Interactive online learning for obstacle classification on a mobile robot. In *2015 international joint conference on neural networks (ijcnn)* (pp. 1–8). IEEE.
- Losing, V., Hammer, B., & Wersing, H. (2017a). Personalized maneuver prediction at intersections. In *2017 ieee 20th international conference on intelligent transportation systems (itsc)* (pp. 1–6).
- Losing, V., Hammer, B., & Wersing, H. (2019). Personalized online learning of whole body motions using multiple inertial measurement units. In *Ieee international conference on robotics and automation (icra) 2019*.

INCREMENTAL learning offers various benefits for real-world applications. For instance, stream-wise processing fosters low time- and space complexities, enabling implementations for mobile devices with very limited amount of resources. Moreover, direct processing on the hardware increases the independence of the system as it is neither dependent on computational servers, e.g. cloud services, nor on the environment in terms of a network connection.

Incremental learning has been applied in plenty scenarios. Concretely, various applications put emphasis on the incremental integration of new data or classes into the system without considering non-stationary environments in particular. In this context, applications in the computer vision domain are quite common. Tackled tasks include object classification (Losing et al., 2015; Kirstein et al., 2005; Bai, Ren, Zhang, & Zhou, 2015), object detection (Opelt, Pinz, & Zisserman, 2006; Lu et al., 2014; Dou, Li, Qin, & Tu, 2015), tracking (Cai et al., 2014; Li, Dick, Shen, Van Den Hengel, & Wang, 2013) or gaze estimation (Sugano, Matsushita, Sato, & Koike, 2008).

An incremental algorithm that learns and adapts a low-dimensional eigenspace representation to reflect appearance changes of the target was presented by Lim, Ross, Lin, and Yang (2005). Embedded within a Markov Chain Monte Carlo framework it is applied to track objects. Kirstein, Wersing, Gross, and Körner (2012) presented an approach for

lifelong-learning of object categories. Category-specific representations are incrementally learned and combined with an LVQ-based classifier.

Incremental learning was also applied in teaching human gestures to a humanoid robot (Calinon & Billard, 2007). In detail, motion data, captured from IMU sensors, is projected in a latent space and encoded in an incrementally adapted Gaussian Mixture Model (GMM). Kulić, Ott, Lee, Ishikawa, and Nakamura (2012) used incremental learning in an unsupervised way to obtain and update full-body motion primitives from motion capture data. In more detail, they first partitioned the data based on stochastic segmentation and then extracted primitives with an incremental clustering algorithm.

Several real-world applications used incremental learning to cope with non-stationary environments. Common tasks are spam filtering (Fdez-Riverola, Iglesias, Diaz, Méndez, & Corchado, 2007; Méndez, Fdez-Riverola, Iglesias, Diaz, & Corchado, 2006) and anomaly detection in the context of cyber security (Lane & Brodley, 1998; Maggi, Robertson, Kruegel, & Vigna, 2009). Further applications include anti-biotic resistance analysis (Tsymbal, Pechenizkiy, Cunningham, & Puuronen, 2006), financial distress prediction (Sun & Li, 2011) or place recognition within dynamic environments (Luo, Pronobis, Caputo, & Jensfelt, 2007).

Incremental learning is also appealing for the purpose of personalization which denotes the modification of a system towards the characteristics of an individual user. Two different modes of personalization have been distinguished (Fischer, 2001; Hasenjäger & Wersing, 2017):

1. *Active customization* by the user, e.g. by making selections and setting parameters.

2. *Adaptive systems* where the usage history is employed to estimate user preferences and situation statistics to adjust parameters and behavior.

In context of this thesis, the term personalization mainly refers to the second type. The general idea is that the focus on one person drastically reduces the variance within the data, enabling a better performance with a smaller amount of data. In other words, products/services trying to serve a broad range of customers have to make sacrifices in order to perform well on average. In contrast, the focus on one single user enables a very specific adaptation to its individual demands. Hence, personalization is particularly crucial in the case of highly diverse customers. The process of personalizing a product is known to foster emotional bonding which directly correlates with the user's effort during the adaptation (Mugge, Schoormans, & Schifferstein, 2009). Personalization also leads to higher brand loyalty, making it especially attractive for manufacturers (Ball, Coelho, & Vilares, 2006).

As incremental learning usually happens within the application, it is particularly suited for online adaptation to one specific user, its environment or behavior. Furthermore, the major problem of inter-person generalization is completely avoided, facilitating the task as well as

the computational complexity, since post-processing steps such as normalization or temporal integration can often be omitted. However, adaptive personalization has been mainly investigated on basis of offline models (Weiss & Lockhart, 2012; Medrano, Plaza, Igual, Sánchez, & Castro, 2016; Neto, de Souza Baptista, & Campelo, 2016; Bifulco, Pariota, Simonelli, & Di Pace, 2013; Butakov & Ioannou, 2015; Harsham et al., 2015; Orth et al., 2017). Regarding personalized recommendation, there are partially incremental systems, i.e. they update user-profiles on the fly which directly affect the recommendations, but their underlying knowledge base relies on batch processing. E.g. Google News uses collaborative filtering for its news recommendations (Das, Datar, Garg, & Rajaram, 2007). In real-time, user statistics such as clicked news stories are updated and considered within an updated news feeds. However, the main module of the recommendation is based on static user categories that are extracted via batch processing, as they comprise the behavior of all users within the last few months. Applications which mainly rely on incremental machine learning methods for the purpose of personalization were only recently considered by Losing, Hammer, and Wersing (2017a, 2019) and both were published in context of this thesis.

In this chapter, three applications of incremental learning are presented and discussed, thereby tackling the following major challenges:

- Given an application scenario, how can incremental learning be efficiently integrated and how does it perform in terms of classification performance and model complexity?

- What are the benefits of personalized online learning in comparison to average offline learning for certain settings?

In the first application, a mobile robot is performing object recognition in a garden environment. It is an interactive learning scenario, since the user can label the visual stream at any time during application to trigger the incremental adaptation. The outdoor environment makes this task particularly challenging because changing lighting conditions impact substantially the visual representation, i.e. there is concept drift in the data. We apply the ILVQ learning architecture from Section 3.2 and compare it with other state-of-the-art methods.

In regard to personalization, two different learning schemes are applied and analyzed. The first uses batch learning and represents the nowadays dominant approach of pretraining a static model at the company. Customers may be able to perform a rough configuration based on few predefined templates. However, the underlying model remains largely unchanged. Usually, the training process at the company relies on a large dataset, trying to cover all likely situations that may be faced in practice. Consequently, a model is generated which maximizes the average performance for multiple users. This learning scheme is referred to as *average offline learning*.

In contrast, *personalized online learning* refers to a model which is delivered without any prior knowledge to the customer and directly

learns during application. Hence, its objective is to maximize its performance on the personalized input it encounters in practice, allowing a flexible adaptation to individual demands.

We compare both schemes within two different applications. One focuses on the behavior prediction of a driver at intersections encountered on the daily commute (Section 5.2). Such predictions can be used to provide situation-specific assistance in terms of warnings or notifications. The objective of the second scenario is to classify human motions as quickly as possible based on IMU signals (Section 5.3). Fast motion classification is a prerequisite for several applications, however, we are particularly interested in supporting humans to execute their current motion by actively controlling assistive devices such as prosthesis or exoskeletons.

5.1 INTERACTIVE ONLINE LEARNING ON A MOBILE ROBOT

In this Section, the proposed learning architecture consisting of the ILVQ in combination with COSMOS prototype placement strategy (see Section 3.2) is applied in a real-world application. In particular, a flexible scheme for interactive learning, a predestined application for incremental algorithms (Amershi & Cakmak, 2014) is investigated. An online recognition architecture that is capable of interactive learning of up to 50 objects in short time was presented in (Kirstein et al., 2005). Here we will rely on a similar architecture but incorporate and investigate a richer online learning model. Vision based incremental learning on a mobile robot was also performed by Luo et al. (2007) as well as by Filliat (2008). However, these systems perform indoors and use different algorithms such as incremental SVM or the Bag of Words scheme (Salton & McGill, 1986). We demonstrate our framework within a real-time-learning scenario which focuses on the challenging task of outdoor object classification on a mobile robot. Hereby, the framework is paired with two mobile apps, creating an easy-to-use online learning system.

5.1.1 *Application Setup*

In our scenario, an autonomous robot is exploring a garden environment (Fig. 45.1) in a random scheme. The user interacts in real-time with the robot by labeling approached objects via an iPad. Labeled objects are incrementally incorporated into the model and learned immediately enabling a direct reaction of the system. New objects can be introduced at any time.

Whenever an object is approached, the robot stops in front of it within a certain distance. If the user does not provide a label, the robot announces the recognized class. Unknown objects as well as uncertain classifications are expressed explicitly. Object specific actions are only executed in case of confident classifications. Actions may include oral comments as well as driving behaviors such as avoiding or driving over.

Figure 5.1: Typical scene of the interactive scenario. The robot drives randomly on the grass area and encounters various objects. The user follows the image-stream on the iPad and can label approached objects.

Objects are always avoided whenever they are classified as unknown or uncertain. Since the garden border is treated as any other object, but coupled strictly with avoidance, the robot stays within the grass area.

We used the Pioneer platform of Adept MobileRobots with a front-mounted Playstation Eye camera, which is directed on the ground and captures the scene with a frame rate of 120 *Hz*. Computation is done by a Lenovo ThinkPad, also mounted on the robot. A color-based grass segmentation algorithm detects obstacles whenever their representation deviates significantly from an environment model. Since this model is adapted dynamically, a high range of color variety can be handled.

The interactive scenario including live-labeling is showcased in the video available at https://github.com/vlosing/datasets/blob/master/stationary/Outdoor/outdoorRobot.avi.

The framework was initially presented by Losing (2014). The publication gives a detailed description wit respect to the image filtering, the App-based interaction and the faced challenges of computer vision in an outdoor scenario.

5.1.2 *Experiments*

In our interactive scenario, various consecutive images of the same object are recorded in constant pose and more or less constant lighting conditions. If the user decides to label this object sequence, all belonging samples are trained in the recorded order. In case of classification, the whole sequence is unknown for the classifier. Therefore, it is important to analyze the generalization to new object sequences, since this is the practically relevant case. It is particularly interesting considering the given small variance within one sequence compared to the high variance across different sequences. Hence, we trained the learning architecture approach-wise. However, the order of the sequences was random.

Table 5.1: Results for the Outdoor dataset with using whole sequences either as training or testing in comparison to a completely random partitioning. In both cases, 60% of the data were used for training, 30% for testing.

Sequence-wise	Test error	Train error	#Nodes
COSMOS	**38.62**	8.96	232.0
Closest	40.42	11.96	245.8
Cluster	40.82	11.04	235.6
Voronoi	41.19	10.78	**231.3**

Random	Test error	Train error	#Nodes
COSMOS	**18.42**	14.06	**234.8**
Closest	21.97	16.77	253.4
Cluster	21.45	16.94	247.0
Voronoi	18.82	14.38	236.6

We are interested to evaluate the proposed prototype-placement strategy *COSMOS* in comparison to other state-of-the-art placement schemes. All prototype placement strategies are in detail described in Section 3.2. Table 5.1 shows the test error for seven training sequences per object, achieved by each placement strategy. Also depicted is the outcome for training a random proportion of 60% of the data. The heavy performance decline on the test set for the approach-wise training is striking, indicating that generalization to completely unseen sequences is more challenging. Consequently, the chosen feature representation is not robust enough against the different variations contained in the dataset. A lot of directly illuminated objects share a similar representation with a big concentration on the yellow bin and the variance of the representation across different objects is often smaller than those of the same object across different approaches. Nevertheless, a high accuracy is achieved when images are shuffled randomly. Even though a high variance is given for the dataset, the difference of images within one sequence is usually small. As soon as one image of each sequence is contained in the training set, the major part of variance is covered and high rates are possible.

There are also more prototypes inserted in the approach-wise training which is caused by more errors during training. One cause is the dependence of the GLVQ updates on a random order due to the underlying stochastic gradient descent minimization. But the main reason is that, during training, the generalization from sequence to sequence is limited too. Therefore, whenever a new sequence is trained, the classifier makes more mistakes until prototypes are added.

COSMOS achieves the best results, but compared to the artificial datasets (Section 3.2), the performance difference between placement strategies is rather small. The image dataset contains many times more classes but less samples per class. Therefore, the choice among these

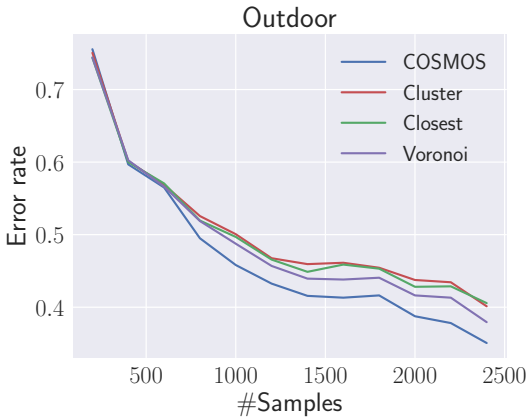

Figure 5.2: Learning curve (test error) for approach-wise training on the Outdoor dataset.

Table 5.2: Comparison against other classifier using the approach-wise training. *COSMOS-GMLVQ* additionally learns the metric of the input space and is therefore more powerful than COSMOS-GLVQ. The test error (TE) and the number of used nodes are given after an increasing amount of used training examples. Pairs of TE and the number of nodes which are not Pareto dominated are marked in bold. Results of a batch kNN are given as reference, because LVQ and SAM are based on nearest neighbor as well.

Method	TE/#Nodes 500 samples	TE/#Nodes 1500 samples	TE/#Nodes 2400 samples
COSMOS-GLVQ	57.1/116	41.4/446	35.7/949
COSMOS-GMLVQ	**55.9/100**	**40.7/377**	**32.9/774**
ISVM	59.4/363	42.5/777	34.8/1347
SAM	59.4/414	47.5/975	35.7/1408
Batch kNN	58.0/500	41.0/1500	34.6/2400

samples is more limited per class, which leaves little leeway to choose good or bad candidates. Furthermore, the data in the high-dimensional space is not broadly distributed but rather strongly clustered. Hence, the already limited choice makes additionally not a big difference. Figure 5.2 depicts the learning curve. All placing strategies perform similar but *COSMOS* leads continuously throughout the training.

Comparison Against the ISVM and SAM

We evaluate *COSMOS* also in combination with the Generalized Matrix LVQ (GMLVQ) (Schneider et al., 2009), which is more powerful because it incorporates metric learning of the input space. Table 5.2 shows the accuracy as well as the stored number of prototypes / support vectors after an increasing number of training examples. This experiment is based on the approach-wise evaluation. Our algorithm performs better than ISVM on the Outdoor dataset and uses thereby a less complex model. The combination with the GMLVQ has always a higher

accuracy and less nodes compared to the GLVQ one. Additionally the complexity, i.e. number of support vectors generated by ISVM, cannot be limited in a straight-forward way unlike for incremental LVQ variants. The metric learning allows our architecture to achieve even better results than the batch kNN algorithm, which uses the whole training set as model representation.

We also applied our SAM architecture within this task. The memory limit was set to a maximum of 1500 instances, matching the model complexity of ISVM. Generally, SAM is designed for learning on large data streams. In this rather small task, some of its essential mechanisms as the adaptive compression within the LTM have basically no effect. Also the adaptive switching of the knowledge base between the STM and LTM does not matter, since the evaluation is done in offline scheme as the models are first constructed based on a training set and evaluated afterward on a separate test set. Nonetheless, Table 5.2 illustrates that SAM achieves comparable results to those of the ISVM, GLVQ and the batch kNN.

5.1.3 *Discussion*

We demonstrated our learning architecture which combines the GM-LVQ with the COSMOS prototype placement strategy within an interactive online learning scenario on a mobile robot. Objects, lying outdoor on the lawn, are trained in real-time and instantly incorporated into the model representation, enabling a direct system reaction. The used RG-chromaticity color histogram turned out to be insufficiently robust to deal with changes due to various lighting conditions. Generalization to completely unseen image sequences, as necessary for the scenario, is therefore only partially possible. However, the representation is not coupled to our learning architecture and can be easily exchanged. One particularly attractive feature representation is the one obtained by Deep Learning (Simonyan & Zisserman, 2014).

On the basis of the scenario we recorded a challenging object classification benchmark and made it available to the public. Here, we showed that our system outperforms the ISVM, a proven state-of-the-art method, by achieving a higher classification performance with a substantially sparser model.

In this section, the incremental adaptation of model complexity was demonstrated within a real-world application. In detail, the model started from scratch without any parameters and evolved steadily as more and more training data was used, adaptively adjusting its own complexity to the demands of the task. The learning of the model was mirrored in a steadily decreasing error rate the more parameters were allocated. Thereby, adjusting model parameters based on cost-minimization turned out to be particularly effective for prototype-based learning.

5.2 PERSONALIZED MANEUVER PREDICTION

This section focuses on exploring the potential of personalized incremental learning in the context of driver maneuver prediction at intersections.

Current advanced driver assistance systems are designed to deliver robust performance over an average range of driving conditions and driver profiles. Consequently, drivers are often dissatisfied because the assistance offered does not match their expectations and preferred driving style. Additionally, the frequently reoccurring driving situations experienced by one particular individual driver constitute only a small fraction of all possible situations. Both factors, individual driver characteristics and reoccurring driving situations provide a great potential for an optimization of the assistance system from an average system to a better adapted, personalized one.

In the automotive context, adaptive personalization based on offline estimation of an appropriate parametrized driver model has been recently considered for real-time route prediction (Neto et al., 2016), adaptive cruise control (Bifulco et al., 2013; Butakov & Ioannou, 2015), predictive Human Machine Interaction (Harsham et al., 2015) and cooperative assistance-on-demand (Orth et al., 2017).

Recently, the application of more generic non-parametric machine learning models caught more interest in the context of advanced driver assistance systems (ADAS) and autonomous driving (Kuefler, Morton, Wheeler, & Kochenderfer, 2017). Their application is, however, often limited to cases of available big datasets necessary for training deep architectures. In this section, we show that generic online learning architectures capable of incremental learning from few training data can be employed for efficient personalization of maneuver prediction as a subsystem of an integrated ADAS.

Tactical maneuver prediction with a horizon of about 2-5 seconds is a highly relevant sub-function for controlling warnings and active safety systems in a car (see the review of Doshi and Trivedi (2011)). Approaches can be based on driver sensing and intention estimation (Rodemerk, Winner, & Kastner, 2015) or just taking GPS traces for trajectory estimation (Liebner, Klanner, Baumann, Ruhhammer, & Stiller, 2013; Klingelschmitt, Platho, Groß, Willert, & Eggert, 2014). A parametric behavior model for curvature-dependent velocity profiles of straight driving and right turns is estimated based on intersection crossing training data (altogether 245 approaches) by Liebner et al. (2013). Klingelschmitt et al. (2014) proposed the anticipated velocity at stop line (AVS) feature, defined as $AVS = v^2 + 2da$, where v denotes the velocity, a the acceleration and d the distance to the intersection. Using a small amount of data (34 approaches on seven intersections) they showed that this information alone is a strong indicator for the drivers intention approaching an intersection. An online learning approach of feature-based maneuver prediction was proposed by Wiest et al. (2015) and applied in a limited setting of two intersections.

In this contribution, we propose a model-free data-driven approach

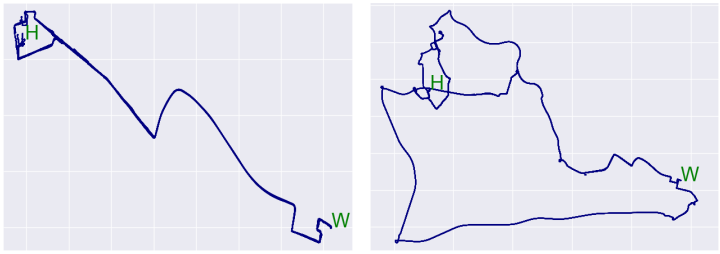

Figure 5.3: All traces of two different drivers in the direction from home to work. The location of the drivers home as well as the working place are marked by "H" and "W" respectively. In contrast to the rather fix route on the left, the commute on the right incorporates multiple alternative routes.

to maneuver prediction, capable of incremental online learning. Compared to other feature-based contributions, our simple approach scales to distinctly more intersections (285 with 5043 approaches) and does not rely on specific filtering, nor on manual labeling of real-world data. Based on a previous analysis of incremental learning architectures (Losing et al., 2018b), we choose an appropriate architecture and demonstrate the performance gain that can be obtained by personalized adaptation of the prediction. We also stress the benefits of personalized context features, which can be easily obtained in the personalized context and lead to a further performance boost.

5.2.1 *Dataset*

Our dataset was extracted from recordings of the daily work-home commute of eleven different drivers. Personalization is particularly useful in this setting, since each driver takes daily an individual and usually similar route. This repeated pattern can be exploited by state-of-the-art machine learning methods and can provide a robust prediction function after only a few commutes. We used the mobile App TrackAddict[1] in combination with an IPhone 5 to record the data. The GPS trace itself was delivered by the Dual XGPS 160, which in contrast to the IPhone delivers the data at a rate of 10 Hz. TrackAddict provides additional raw data to each GPS coordinate such as time, velocity, gyroscope values as well as the corresponding video. However, we utilize only the time (s), velocity ($\mathrm{km\,h^{-1}}$) and the GPS coordinates (°) for our analysis.

The variance of the daily route varies from driver to driver as it is illustrated by Figure 5.3. Some drivers are taking only small detours, whereas others choose among several different alternatives.

Preprocessing

We removed too short streams ($<$ 5 min) as well as those with a too low GPS rate resulting from failures of the GPS receiver. Only

1 http://racerender.com/TrackAddict/Features.html

intersection approaches were extracted from the raw data. We used a maximum prediction horizon of four seconds leading to approximately 40 data points per approach. This time is sufficient to provide situation-dependent assistance at intersections (Doshi & Trivedi, 2011).

Relevant Intersections and Potential Stop Points

A stream of driver d corresponding to a commute ride is given by a sequence $S^d = [p_1, \ldots, p_m]$ of m recorded tuples $p = (time, vel, long, lat)$. A potential stop point within such a given stream is determined by a measurement with velocity smaller than some predefined value α, whereby we always take the first such measurement in a row, and we make sure a certain velocity β is reached before the next stop can be encountered. That means, a potential stop is found at time step t_i if $vel_{t_i} \leq \alpha$, $vel_{t_i-1} > \alpha$, and there exists some t' before the time of the next stop point $\hat{t}$ such that $t_i < t' < \hat{t}$ and $vel_{t'} \geq \beta$. In case of t_i being the last stop point of a stream, the latter condition is omitted. This way, every commute ride S_r^d of a driver d yields a set of potential stop points $O_r^d := \{o_1, \ldots, o_{k_r}\}$. For every driver d, we collect all commute sequences $S_{\text{total}}^d := \bigcup_r S_r^d$ and the corresponding set of potential stop points $O_{\text{total}}^d := \{o_i \mid \exists r\, o_i \in O_r^d\}$.

Since we are interested in the prediction of driver behavior at *intersections*, we determine all intersections of the observed drives within the corresponding map area[2]. Thereby, intersections are simply represented by their GPS-coordinates $I = (long, lat)$. To reduce the amount of intersections and also to generate a challenging as well as balanced dataset, we consider only intersections at which the driver has stopped at least once. Further, we align a potential stop point o_j to an intersection I_j, provided the point o_j is the closest stop point to I_j and the distance is smaller than a predefined value γ, measured in the Euclidean distance of the GPS signals[3]. All potential stop points which are not aligned to an intersection are irrelevant.

Automatic Labeling

For every commute, we identify the relevant intersections on its way. We identify the parts of the commute ride, which are within a distance of at most 20 m to the intersection. For these events, we distinguish the following classes:

- *straight* (cross without stopping)
- *stop*
- *turn* (turn left or right without stopping)

All datapoints of one approach are labeled as the same class. An approach is labeled as *stop* provided it is contained in the list of stop

2 We extracted the intersections from OpenStreetMap (OpenStreetMap contributors, 2017)

3 We chose the parameter values $\alpha = 5\,\text{km}\,\text{h}^{-1}$, $\beta = 20\,\text{km}\,\text{h}^{-1}$ and $\gamma = 20\,\text{m}$ to generate the dataset.

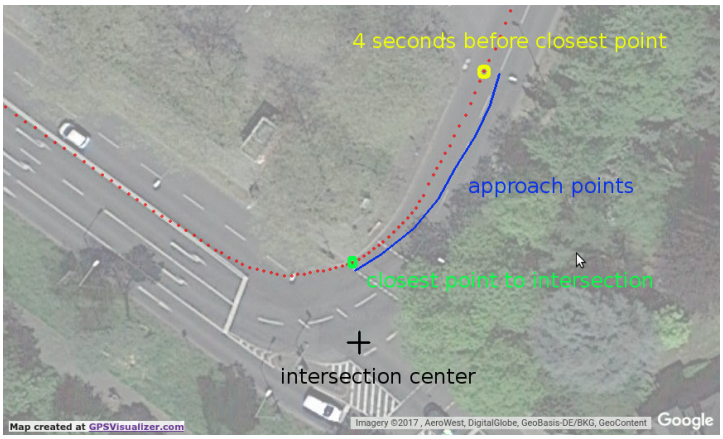

Figure 5.4: A typical right turn approach. The approach data contains all points four seconds backwards from the closest point to the intersection.

points as described above. In this case, it is irrelevant whether the car goes straight or takes a turn after the initial stop. A *stop* approach contains the data sequence four seconds backwards from the stop point. Please note, that our learning task is not the same as learning the daily route of the driver. The car can stop at any day at any intersection caused by e.g. traffic lights, preceding cars or pedestrians, which makes the subsequently taken direction extraneous. For the remaining events, we automatically determine the type based on the following geometric considerations. Two lines are fitted to the GPS trace, before and after the intersection. An approach is labeled as *turn* when the angle in between the two lined is larger than 30°. Otherwise, it is labeled as *straight*. The *turn* and *straight* data contain the data sequence four seconds backwards from the closest point to the intersection. Figure 5.4 illustrates an exemplary intersection *turn* approach.

A typical GPS stream with labeled intersection approaches is shown in Figure 5.5. Even though we optimized the parameters of the automatic labeling, our data driven approach comes at the cost of a minor amount of label noise. This mainly concerns approaches of the *straight* and *turn* class, which are sometimes hard to discriminate from each other, due to the arbitrary different intersection layouts. Furthermore, the imprecision of the GPS signal adds to the complexity.

Dataset Characteristics

The main characteristics of the resulting datasets are given in Table 5.3. The number of approaches of each driver varies naturally within the dataset, depending on the amount of recorded streams, the length of the commute as well as whether the corresponding route is located in a rural or urban area. Table 5.4 illustrates the largely similar class distribution of the approaches with *turn* having the smallest share.

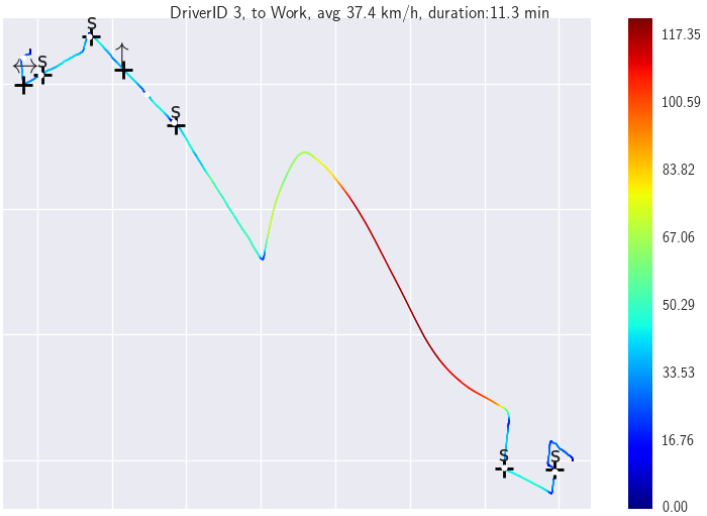

Figure 5.5: A GPS trace from home to work. The crosses mark the relevant intersection for this trace, whereas white dots denote stopping positions. Please note that some intersections are excluded due to the fact that the driver did not stop at them on all streams in one direction. The labels of the approaches are given by ↑ = straight, ↔ = turn, s = stop. The color of the trace encodes the velocity in $\mathrm{km\,h^{-1}}$.

Table 5.3: The dataset broken down to the individual drivers, contributing to the dataset in different proportions. Driver ten has with 22.4% the highest share, whereas driver three with 2.7% the smallest one. Each approach is usually represented by 40 single datapoints.

DriverID	#Streams	#Intersections	#Approaches	#Datapoints
1	31	35	809	32664
2	23	17	240	9722
3	14	27	260	5427
4	37	26	561	22702
5	12	32	211	8510
6	30	17	288	11674
7	35	16	414	16752
8	28	30	642	25814
9	24	13	379	15303
10	57	61	1103	44648
11	21	11	136	5482
$\sum$	312	285	5043	198698

Table 5.4: The class distribution of the dataset.

Class	#Approaches	#Datapoints	Proportion (%)
Straight	1819	72194	36.33
Stop	2062	80985	40.76
Turn	1162	45519	22.91

Table 5.5: The evaluated models with corresponding feature sets. Vel = Velocity, Acc = Acceleration, Dist = Distance to intersection, I-Lat= Intersection-Latitude, I-Lon = Intersection-Longitude.

Abbreviation	Model	Learning	Features
AVG_{AVS}	Logistic Regression	offline	AVS,Dist
AVG	Random Forest	offline	AVS,Vel,Acc,Dist
PERS	Online Random Forest	online	AVS,Vel,Acc,Dist
$PERS_C$	Online Random Forest	online	AVS,Vel,Acc,Dist,I-Lat,I-Lon

5.2.2 *Experiments*

Average offline models are compared against personalized online models in the supervised classification setting. Thereby, the objective is to predict one out of C intersection maneuvers, represented by the target variable $y \in \{c_1, \ldots, c_C\}$, on the basis of a set of features $\mathbf{x} \in \mathbb{R}^n$, which characterizes the ego vehicle state using attributes such as velocity, acceleration or GPS-coordinates.

Four different models were evaluated. Table 5.5 depicts their characteristics as well as the respective set of features. At each point in time of an intersection approach the feature vector $\mathbf{x}_t$, possibly containing the velocity, acceleration, AVS (Klingelschmitt et al., 2014) and the GPS coordinates of the intersection, is used to calculate the prediction of the corresponding model. Average offline models were evaluated in a leave-one-driver-out scheme. Precisely, they are tested with the data of one specific driver, whereas those of the remaining drivers is used for training. This is done repeatedly such that each driver is used for testing once. We mainly utilize on- and offline variants of the popular Random Forest (RF) (Breiman, 2001) to enable a fair comparison. The RF is a well known state-of-the-art learning algorithm, delivering highly competitive results (Fernández-Delgado et al., 2014; Losing et al., 2018b) and is easy to apply out of the box. We additionally use as baseline a Logistic Regression (Bishop, 2006) model with the AVS feature and the distance to the intersection as it was proposed by Klingelschmitt et al. (2014). The other offline model is a RF with an extended feature set including next to the AVS feature, also the velocity, acceleration as well as the distance to the intersection.

We conducted also experiments coupling the Logistic Regression model with the extended feature set, as well as the RF with the AVS feature only. However, the Logistic Regression model did not profit from the additional features and the RF performed on average about 5% worse than AVG_{AVS}. Consequently, we omit these results in our analysis.

Both incremental models are instances of the Online Random Forest (ORF) (Saffari et al., 2009). One is using the same feature set as the RF, whereas the other additionally incorporates personalized context information in terms of the GPS coordinates of the corresponding intersection. Personalized context features have the advantage of being rather easily to obtain for a specific user and can substantially boost

the individual prediction after only a few examples. In the context of the average user, however, they often have no specific meaning at all and can even deteriorate the performance or require a huge amount of training examples to be beneficial. In our case, for example, the intersection GPS coordinates would only boost the performance of the average model if it incorporates examples for each intersection approached from all directions. This requires a tremendous amount of data, by far more than contained in our dataset, even though the routes of the drivers are locally related.

We evaluate the incremental models in the online learning setting (see Section 3.1.2). However, since the data is temporally ordered and has a high degree of label autocorrelation we perform the approach-wise scheme. Precisely, the model has to predict all samples of one approach, before the corresponding labels get revealed. We trained from the scratch one online model for every driver in single pass. Meaning, the online models are utilized without any form of pre-training and only access the data of one specific driver.

Results

Figure 5.6a shows the resulting error rates for different prediction horizons.

Clearly, the prediction gets easier the closer the driver is to the intersection. The method AVG performs on average similar to AVG_{AVS} even though it uses additional features. However, the AVS feature is basically a compression of the velocity, acceleration and intersection distance for the purpose of intersection intent inference. Our experiments confirm its usefulness in this context.

Both online models substantially outperform their offline counterparts, underlining the benefits of a personalized prediction within this setting. This is particularly remarkable considering the severely smaller amount of available training data as well as the usual performance advantage of offline models. The error rate may even decrease further with larger training sets. Moreover, it is shown that the addition of the intersection GPS coordinates boosts the performance throughout the whole prediction horizon.

The advantage of the online models is even more pronounced if their performance is measured after a certain amount of training data has been seen. Figure 5.6b contrasts the mean error rate to those achieved on the second half of each drivers data. The error rate is distinctly lower for the second half of the data, due to the naturally higher amount of mistakes done at the beginning of learning.

The learning curves of the online models are shown in Figure 5.6c. Precisely, it depicts the error rate depending on the number of trained approaches for a specific intersection. The personalized predictors require only a small amount of training data to compete with the average ones. In fact, they are already more accurate at the third time they approach the same intersection. Hence, only one / two days of commute tours are sufficient to gain an advantage with the personalized models in our scenario. PERS seems to be converged

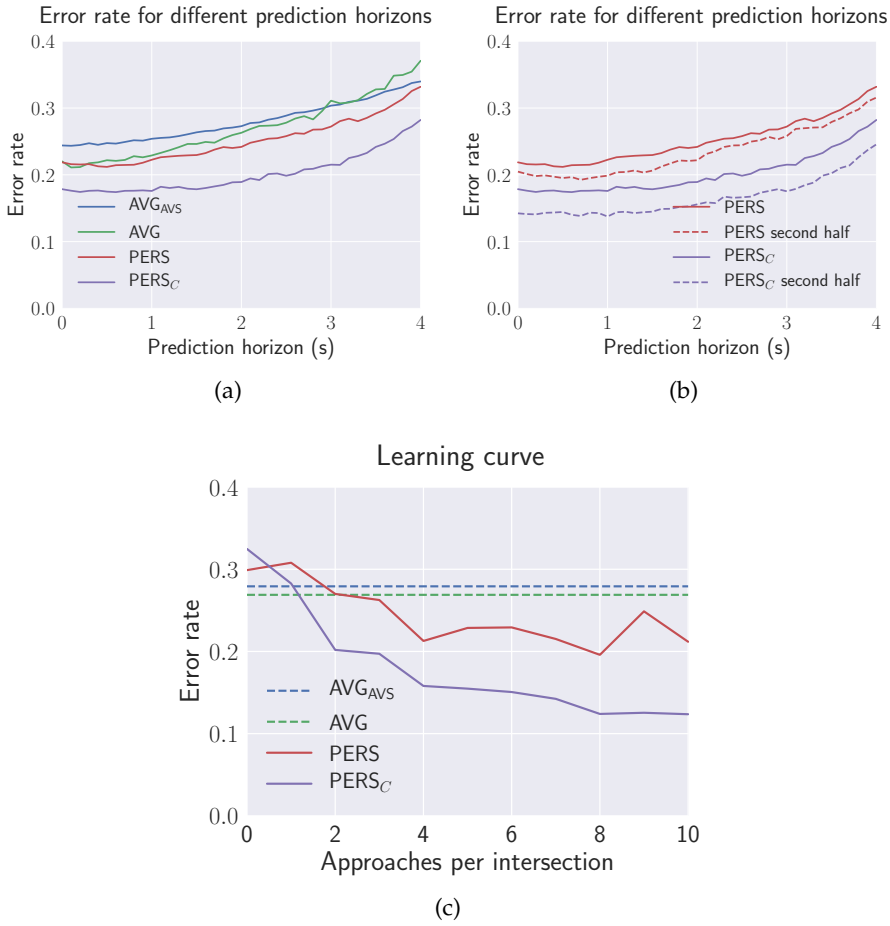

Figure 5.6: (a) The mean error rate of the evaluated models depending on the prediction horizon. (b) The mean error rate as well as the one considering only the second half of each drivers data, achieved by both online models. The error rate is distinctly lower for the second half, because more mistakes are done at the beginning of learning. (c) Learning curve of the personalized online models. The average error rate is given depending on the number of experienced approaches for the specific intersection. The personalized predictors are quickly more accurate than average ones. The average performance of the offline models is given via dotted lines.

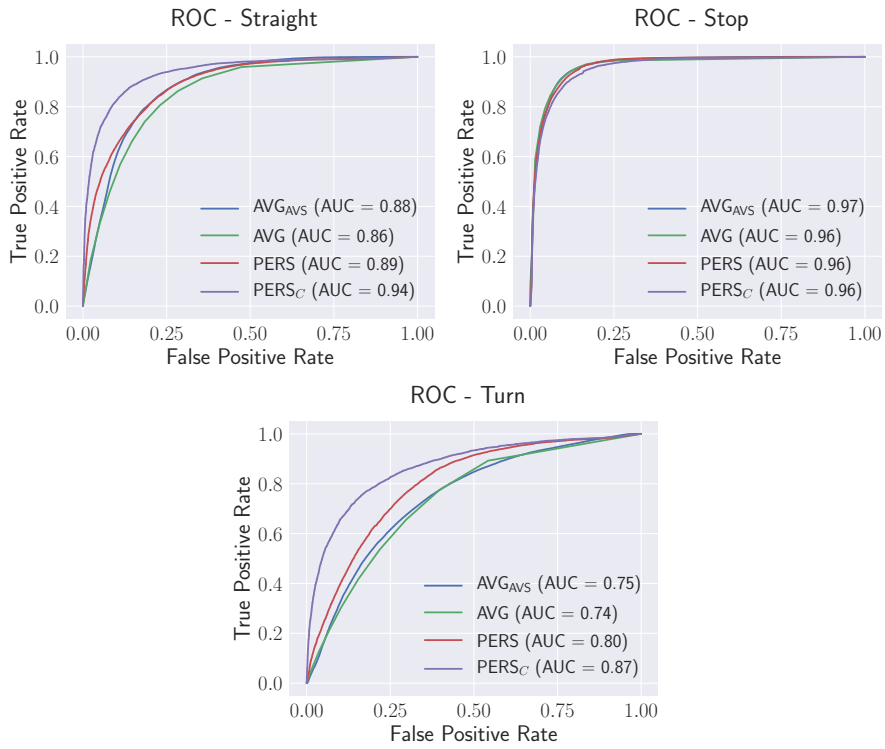

Figure 5.7: ROC curves of the models for all classes. The online models are superior for the *straight* and *turn* class, whereas all models have a similar ROC for the *stop* class.

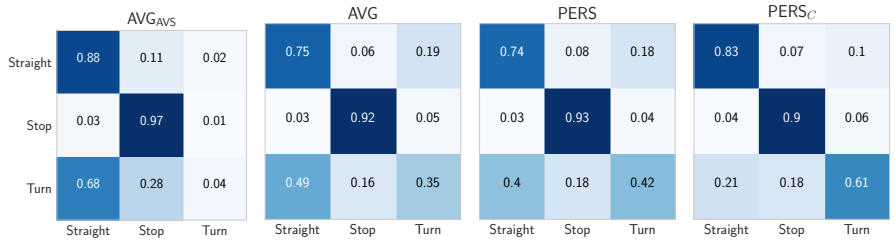

Figure 5.8: Confusion matrices of the models. Most confusions occur between the *straight* and *turn* class. However, the context features distinctly enhance the discrimination. AVG$_{AVS}$ has constantly a low confidence for the *turn* class as it is rarely used for prediction.

after approximately four approaches, whereas PERS$_C$ keeps improving with additional data.

Figure 5.7 depicts the receiver operating characteristic (ROC) curves for all classes and Figure 5.8 shows the confusion matrices of the models.

Most confusions occur between the *straight* and *turn* class, whereas the *stop* approaches are classified with high accuracy. AVG$_{AVS}$ has the lowest error rate for the *straight* and *stop* class. However, particularly the ROC curve for the *straight* class illustrates that it is not necessarily the best model. Rather, these low error rates come at the cost of a poor

accuracy for the minority class *turn*, constantly predicted with a low confidence, and therefore, rarely used. Nonetheless, the ROC curve for the *turn* class shows that AVG_{AVS} is quite able to reasonably predict this class with an appropriate tuning of the confidence thresholds.

The personalized models perform particularly well for the *straight* and *turn* class because of several reasons. Clearly, they profit from the fact that the class distribution of a specific driver is often different than those of the average driver. Furthermore, a driver may approach intersections in a specific way, facilitating the personalized classification.

$PERS_C$ is by far the best model in general. It is able to implicitly generate an intersection specific prediction model, due to its access to the intersection GPS coordinates.

Generalization of Personalized Models

One interesting question is whether the personalized models are able to learn a driver-specific way of approaching intersections in general. Therefore, we analyze the personalized model in a leave-one-intersection-out as well as in a leave-one-approach-out experiment and compare its performance with those of an average model. For a fair comparison we use for both models the extended feature set (AVS, Vel, Acc, Dist). The average model achieves in both experiments an error rate of 0.29, whereas the personalized model performs slightly worse in the leave-one-intersection-out experiment (error: 0.30), but clearly better in the leave-one-approach-out setting (error: 0.24). Therefore, it is not confirmed that the personalized model learns a generic driver-specific way of approaching intersections. Rather, we can conclude that the advantage of the personalized models is mainly based on already seen intersections approached by the specific driver.

Results of SAM and ILVQ

We also evaluated our SAM and ILVQ architecture to assess their potential as personalized models within this task. Thereby, we use the same architecture for ILVQ as in the outdoor robotics application described in Section 5.1 (*COSMOS* prototype-placement strategy with metric learning on basis of GMLVQ). Both algorithms are distance-based classifiers, hence, each dimension was normalized, using a small sample (10% of the data) of the personalized data, to compensate for different feature scales. Furthermore, the learning rate of ILVQ was optimized on the small sample as well. Figure 5.9 shows the resulting error rates for different prediction horizons for the personalized model as well as the one using context information. ILVQ and SAM achieve competitive results in comparison to ORF, underlining their robustness for real-world scenarios. SAM was even applied out of the box without any tuning of the hyperparameters. The models are particularly competitive when the context features are used. The task contains mainly virtual drift in terms of label autocorrelation as well as a repetitive order of approached intersections. Since real drift is not present, incrementally growing models designed for stationary envi-

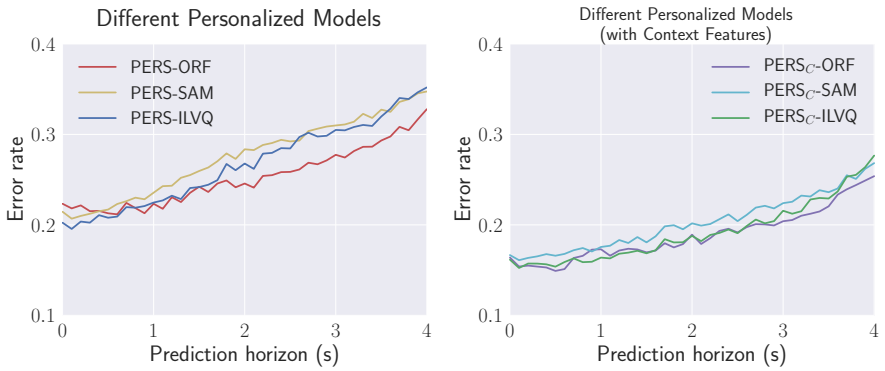

Figure 5.9: The mean error rate of different personalized models depending on the prediction horizon. The results on the left are based on the feature set of velocity, acceleration and distance to intersection. Performances which include also the context features in form of GPS coordinates of the corresponding intersection are depicted on the right.

ronments, as ILVQ or ORF, achieve a high performance. The absence of real drift is also the explanation that SAM performs slightly worse. In such a case, there is little conflict between current and past concepts and therefore SAM's preservation of information is mainly based on the successive compression of the long-term memory. Here, it discards valuable information in form of the supervised signal by relying on unsupervised clustering.

5.2.3 *Discussion*

This section highlighted the benefits of a personalized incremental learning approach in the setting of intersection maneuver prediction. It is a particularly suitable setting for personalized incremental learning because of the following reasons. First, the prediction of intersection behavior allows an accurate and automatic extraction of ground truth information in retrospective. Second, the focus on the daily commute enables a quick adaptation of personalized online models, since corresponding trajectories have rather a low variance and can be extracted on a daily basis.

A dataset containing the GPS traces of the daily work-home commute, driven by eleven different drivers, was recorded, covering altogether 285 intersections with corresponding 5043 approaches. Our dataset is severely larger and more diverse than those in comparable contributions. We applied a simple, model-free as well as data-driven approach, providing an accurate maneuver prediction. In contrast to state-of-the art techniques, it does not rely on explicit and sophisticated ego vehicle modeling, nor on manually labeled data. Precisely, we compare the prediction error rate of offline models trained in a leave-one-driver-out scheme with incremental personalized models, trained in online fashion exclusively with the data of one specific driver.

The efficient personalized models turned out to be more accurate after only a small amount of training data than their offline counterparts. Precisely, they are already more accurate after one or two commute tours. Furthermore, we highlighted that personalized context data such as the intersection GPS coordinates, often only viable and useful for the specific user, additionally increase the performance.

Our simple, model-free as well as data-driven approach leads to an accurate maneuver prediction and, in comparison to state-of-the art techniques, does neither rely on explicit and sophisticated ego vehicle modeling nor on manually labeled data. However, in the case of available precise lane-level maps, an explicit modeling of intersections may have advantages with respect to a better generalization to areas driven for the first time (Liebner et al., 2013). In subsequent experiments, we showed that the advantage of the personalized models is not due to a generic driver-specific way of approaching intersections, but rather based on already seen intersections approached by the specific driver.

5.3 PERSONALIZED HUMAN ACTION CLASSIFICATION

In this section, the benefits of personalized online learning are investigated within the task of online action classification. We apply the Xsens bodysuit (Roetenberg, Luinge, & Slycke, 2009) to record a rich and large motion database, incorporating eighteen different classes. In a preliminary study, we first utilize a greedy feature selection approach to determine the crucial IMUs that are necessary to achieve a high classification performance. Subsequently, we thoroughly compare average offline machine learning models with online personalized ones. Our experiments show that online models require a small amount of data to outperform average user systems, particularly if only the raw sensor data is used, often a necessity for applications with strictly limited computational resources. The performance advantage of the online modes increases further with more personal training data.

The classification of human actions is crucial for diverse application scenarios such as surveillance, human-machine interactions and pervasive health care (Aggarwal & Ryoo, 2011; Poppe, 2010). In contrast to the tasks of activity recognition or offline action classification, where the classification is performed on the basis of a fully observed sequence, online action classification aims to recognize the motion on the fly and as quickly as possible. This crucial difference opens up a spectrum of further application scenarios. One that we are particularly interested in is the control of wearable devices which actively support the motion of users. Such hardware may support physical rehabilitation or assist handicapped persons by means of an improved prosthesis control for example (Paaßen, Schulz, Hahne, & Hammer, 2017; Kong & Jeon, 2006). Another application area are working environments where repetitive and strenuous motions such as bending and kneeling are frequently demanded (Pratt, Krupp, Morse, & Collins, 2004; Muramatsu, Kobayashi, Sato, Jiaou, & Hashimoto, 2011).

Usually, action classification is done with skeleton data extracted from vision-based sensors (Garcia-Hernando & Kim, 2017; Li et al., 2016; Gaidon, Harchaoui, & Schmid, 2011; Wang & Schmid, 2013; Jain, v. Gemert, Jégou, Bouthemy, & Snoek, 2014). However, the accessibility of visual data is limited to environments equipped with the necessary sensors. In case of wearable devices, integrating inertial measuring units (IMU) is a viable and elegant solution to obtain a continuous data stream independent from environmental properties. IMUs have been mostly used for activity recognition tasks, which aim to discriminate high-level actions such as walking, cycling and swimming, often on the basis of commercial everyday hardware like smart-phones, smart-watches or fitness-tracker wristbands (Kwapisz, Weiss, & Moore, 2010; Weiss, Timko, Gallagher, Yoneda, & Schreiber, 2016; Nelson, Verhagen, & Noordzij, 2016). In contrast, we apply the Xsens bodysuit which incorporates 17 IMUs spread across the body, providing a precise measurement of postures with a high sample rate, enabling motion recognition in a fine-grained way.

Commonly, online action classification has been tackled with offline machine learning methods. A static model is generated by a large amount of labeled examples and then applied to a hold-out set (Garcia-Hernando & Kim, 2017; Li et al., 2016; Gaidon et al., 2011; Wang & Schmid, 2013; Jain et al., 2014). However, we aim for a system that adapts to the personal behavior patterns of its user in real-time and on the fly based on inertial measurements. Ideally, a continuous collaboration between the system and user is triggered where both alternatively adjust to each other and maximize the utility of the system. A static model is simply not suitable in such a scenario. Hence, we apply online machine learning algorithms which continuously adapt to the incoming data stream. Since the data is processed one-by-one, these algorithms are able to handle infinite streams and thereby guarantee a low time and space complexity. Therefore, even limited computational hardware is able to process the stream locally without access to cloud services, allowing an application independent from internet connectivity and offering complete data privacy at the same time.

The potential of personalization in the context of motion recognition was mostly analyzed with static models. Weiss and Lockhart (2012) investigated personalization in an activity recognition task on the basis of accelerometer-data obtained from smart phones. In their experiments, personalized models were more accurate with a clearly smaller amount of training data. Medrano et al. used personalized models for the task of human fall-detection and achieved a higher performance in comparison to average user models (Medrano et al., 2016). Our contribution differs from the mentioned work, since we combine personalization with online learning to classify fine-grained motion classes on the basis of a large data foundation obtained with multiple IMUs. We first perform a forward feature selection to determine a small sub-set of IMUs which are the most valuable ones in terms of classification performance. Subsequently, we thoroughly compare average offline machine learning models with online personalized ones.

Our experiments show that online models require a small amount of data to outperform average user systems, particularly if only the raw sensor data is used, often a necessity for applications with strictly limited computational resources. The advantage of the online models increases further with more training data.

5.3.1 *Online Action Classification*

The focus is the evaluation of average offline models as well as personalized online models in the online action classification task (Li et al., 2016). A stream $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_t}\}$ of feature vectors (IMU sensor measurements in our case) arrives one after another. The goal is to determine whether a frame $\mathbf{x_t}$ at time t belongs to an action among the predefined C action classes. In contrast to action detection tasks, there are no situations in which $\mathbf{x_t}$ does not belong to any predefined action. A model predicts the action class in the form of:

$$y_t^\star = \underset{y_t \in \{1,\ldots,C\}}{\arg\max} \; P(y_t | \mathbf{x}_1, \ldots, \mathbf{x}_t).$$

Naturally, online action classification is more challenging than offline action classification, since methods are not allowed to peek in the future and must instantaneously determine the action of $\mathbf{x}_t$.

5.3.2 *Dataset*

The data recording as well as the resulting dataset is described in this section.

Recording Setup

We used the popular Xsens bodysuit with 17 IMUs, measuring linear and angular motions with a triad of gyroscopes and accelerometers, distributed on different body locations as shown in Figure 5.10. The sample rate of 120 Hz is sufficient to track even highly dynamic motions as common in sports (Tessendorf, Gravenhorst, Arnrich, & Tröster, 2011). We used a sub-sampled rate of of 60 Hz. Six additional sensor positions are interpolated, resulting in altogether 23 different *segments*. Altogether, 13 different *feature types* such as acceleration, joint-angles or orientation are provided for each segment and are encoded with three or four dimensions. Various filters are used to provide also integrated feature types such as velocity and position. However, these are known to be prone to drift due to the inevitable sensor bias (Damgrave & Lutters, 2009).

Feature Extraction and Normalization

We translated the position of all segments such that the pelvis is in the origin of the X-Y coordinates. Xsens offers kinematic data as 3-dimensional vectors. We enriched the data by adding further feature

Figure 5.10: The Xsens body suit with 17 IMU sensors. The sternum IMU is occluded due to the image perspective. We used the wireless version. Source: https://www.xsens.com/.

types which encode only the magnitude of these. Furthermore, a normalized version of each feature is added which is obtained by mean-subtraction and scaling to unit-variance. This has been done per person to improve generalization across different subjects. In total, up to 32 different feature types are available for each of the 23 segments, resulting in 1472 dimensions per sample.

Action variety

Four different subjects performed nine movement sequences consisting of several single actions. These sequences were repeated 10-20 times. In total, sixteen fine-grained action classes are present in the data[4]. The recordings were done in one session for each subject and the data was manually labeled. Figure 5.11 depicts some action sequences. The action class distribution of the dataset is depicted in Figure 5.12. The data is mainly dominated by five different walking actions as well as the standing class, which frequently occurs in-between. Altogether, the dataset encodes 2755 actions represented by 329021 single instances.

5.3.3 *Experiments*

Analogous to the evaluation in Section 5.2.2, we compare an offline average user model against an online personalized one, which we denote as *average* (AVG) and *personalized* (PERS) from now on. The *average* model is trained in leave-one-subject-out scheme. Precisely, it is tested with the data of one specific subject, whereas data of the

4 The action classes are : stand, walk forward, walk backwards, walk sideways, walk curve, turn, squat down, squat up, lunge down, lunge up, stand with object, walk forward with object, walk backwards with object, turn with object, put object down, squat up with object)

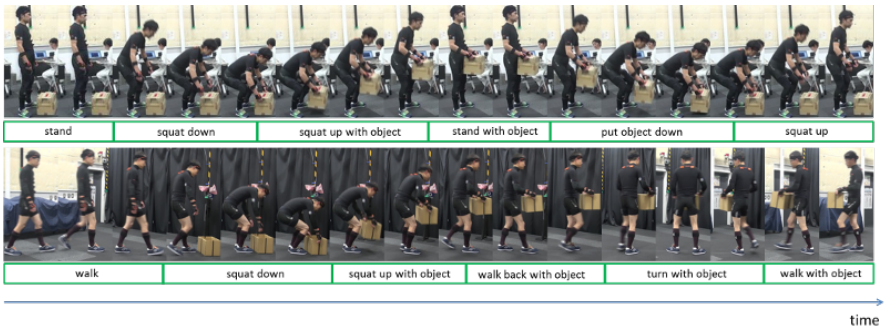

Figure 5.11: Exemplary sub-parts of sequences with different subsequent actions. The fine-grained ground truth is depicted in the green boxes. The classes are often ambiguous in the transition period from one action to another.

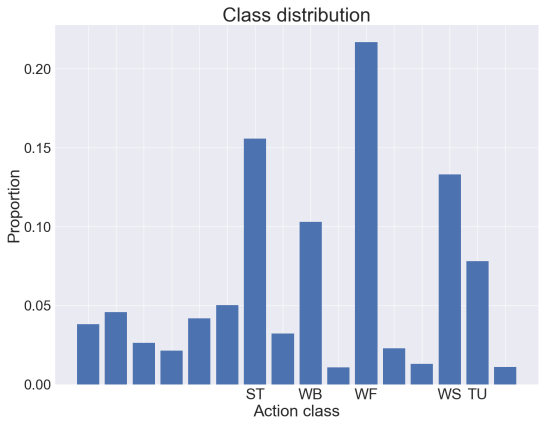

Figure 5.12: The imbalanced class distribution of the recorded data. For reasons of clarity, only the dominating classes are labeled in the chart. ST = stand, WB = walk backwards, WF = walk forward, WS = walk sideways, TU= turn around.

remaining three subjects is used for training. This is done repeatedly such that each subject is used for testing once. Personalized models are evaluated in the online learning setting, as described in Section 3.1.2. The model classifies first the label of one sample and uses it afterward for model adaption. This is done for all samples in the dataset. As in the application of the driver maneuver prediction in Section 5.2.2, there is once again a high degree of label autocorrelation, since each action consists of a number of samples with the same class. Therefore, we perform an action-wise evaluation. Precisely, the model classifies all samples of one action, before the corresponding labels get revealed. The personalized models are trained from scratch, one online model for every subject in single pass. Consequently, online models are utilized without any form of pre-training and only access the data of one subject. Please note, that we calculate both errors (off- and online) using the same data for testing, but the online algorithms continuously adapt their model to the test subject.

We apply on- and offline variants of the popular Random Forest

Table 5.6: The three most important feature types.

Priority	AVG	PERS
1	position (scaled)	sensor-acceleration
2	magnitude-angular-velocity (scaled)	sensor-angular-velocity
3	velocity (scaled)	center-of-mass

(RF) (Breiman, 2001; Saffari et al., 2009) to enable a possibly fair comparison. Concretely, we use decision forests consisting of 50 trees and rely on the class entropy as impurity function (Fayyad & Irani, 1992).

Encoding Recent Motion Data

Action classification is known to be more accurate when the feature vector encodes not only the current sensor state, but the recent motion history. We performed preliminary experiments to compare the effect of different feature configurations. We evaluate the performance for encoding only the current sensor state versus stacking the features of the last 30 frames (∼0.5 s)) versus encoding the features of the last 30 frames via the five highest values of the discrete cosine transformation (DCT) (Ahmed, Natarajan, & Rao, 1974). The highest performance is achieved by the DCT encoding even though it uses a substantially lower amount of dimensions in comparison to stacking. Based on these results, we use the DCT encoding for the rest of the experiments[5]. The detailed results are listed in the appendix (Table A.1).

Feature Selection

The Xsens body suit offers various feature types for each segment. To reduce the number of input dimensions, we performed for each model feature selection, determining the most valuable dimensions in two steps. We use forward feature selection (Abe, 2010) to minimize the classification error. Concretely, we start with an empty set and iteratively add the feature type / segment which minimizes the error the most until all of them are used. In the first step, we extract the most valuable feature types (see Section 5.3.2), thereby using all segments and in the second we determine the most valuable segments using only the previously extracted feature types. The normalized feature types were not offered to the *personalized* model, since they have no effect on person-specific and scale-invariant models. Figure 5.13 shows the error rate depending on the number of feature types / segments. It can be seen, that only few feature types and segments are necessary to reach a high performance. Using all feature types is even harmful which is probably caused by overfitting. The personalized model relies on less data and is therefore more affected. Based on these results, we decide to use the three best feature types together with the five most valuable segments. Table 5.6 lists the most valuable feature types for both models. The chosen types by the *average* model are mostly tempo-

5 The discrete Fourier transformation (DFT) (Harris, 1978) was evaluated as well, however, it performed worse than the DCT

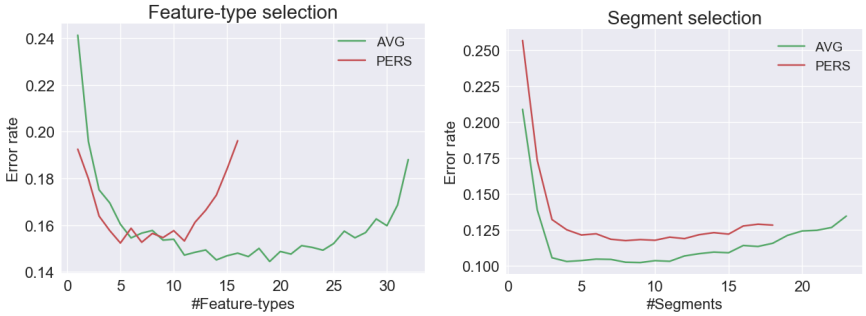

Figure 5.13: Forward feature selection in two steps: First, the best feature types are determined using all segments (top). Subsequently, the most valuable segments are determined on the basis of the three best feature types (bottom). The *personalized* model had no access to the normalized features because they have no effect on its performance. The number of evaluated segments depends on the chosen feature types in the first step. Hence, the number of evaluated segments is different for both models.

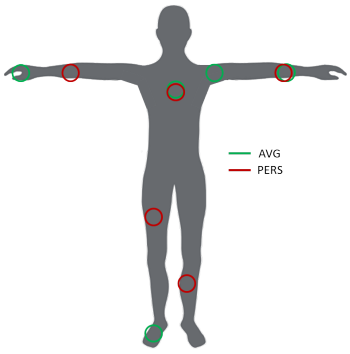

Figure 5.14: The favored sensor locations of both models. There is no excessively covered body part. Instead, the sensors are equally distributed across the whole body.

rally integrated and normalized, highlighting that the person-specific normalization improves the inter-person generalization. In contrast, the *personalized* model prefers rather the raw sensor data. Figure 5.14 displays the preferred sensors. Both models equally distribute the segments across the body, indicating that our motion dataset profits from an encoding of the whole body posture.

Results

The reported results are based on the selected features (Section 5.3.3) which are encoded via the five highest values of the corresponding DCT (Section 5.3.3). Figure 5.15 shows the average error rate of the *personalized* and *average* model depending on the relative action progress. As expected, the models are the most uncertain during the transition period from one action to another, since the data is very similar then for consecutive action classes. Also the ground truth is most inconsistent then because it is hard to define the exact moment one action

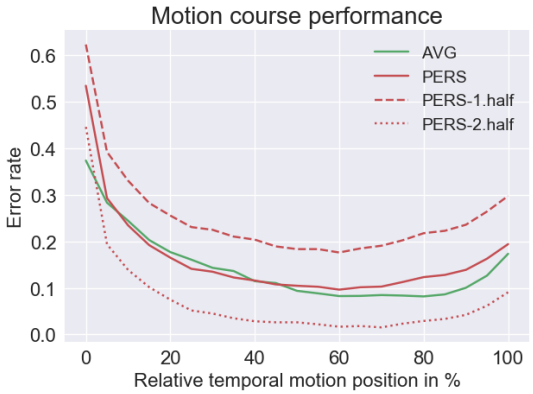

Figure 5.15: The classification performance depending on the relative progress of the action. The models are less accurate in the transitions between the actions.

ends and another starts. The models are particularly uncertain at the beginning of motions because the feature vector encodes the recent past. Consequently, a lot of variance is within the data at this stage, since different action classes can transition into the same class.

The overall results of both models are similar. However, the *personalized* model is continuously adapting, therefore, we analyze it in more detail. Assuming the data of a given subject contains l single motions, we break the set into approximately equally sized sets l_1 and l_2, i.e. $|l_1| \approx |l_2| \approx \frac{|l|}{2}$, and measure the performance on each of those independently. The performance of the first half is the incremental error achieved on l_1 by an online model starting from scratch without any prior knowledge and continuously adapting on the basis of l_1. Averaging the results over all subjects leads to the curve in the plot. As expected, it performs clearly worse in comparison to the *average* model. The performance of the second half is the incremental error achieved on the second half of the motions l_2, but this time the model has already seen the data of the first half l_1. In this case, we observe that the *personalized* model significantly outperforms the *average* one as soon it has seen enough motions of the respective person. The average performance of the *personalized* model can be expected to converge at least towards the error rate achieved for the second half of the data with additional motion sequences.

The learning curves are depicted in Figure 5.16. The performance of the *personalized* model is measured by averaging the performance over the last 3600 instances ($\sim$1 min), whereas the performance of the *average* model is always measured by classifying the whole data of the hold-out-subject. The *average* model has access to significantly more data because it is trained on three subjects. Nonetheless, the learning curve of the *personalized* model is not only steeper, but it reaches a distinctly lower error rate, highlighting the benefits of personalized learning in this field.

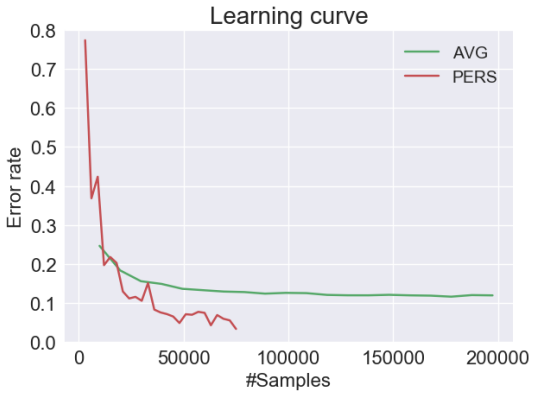

Figure 5.16: The learning curve of both models. The *personalized* model achieves a lower error with significantly less data.

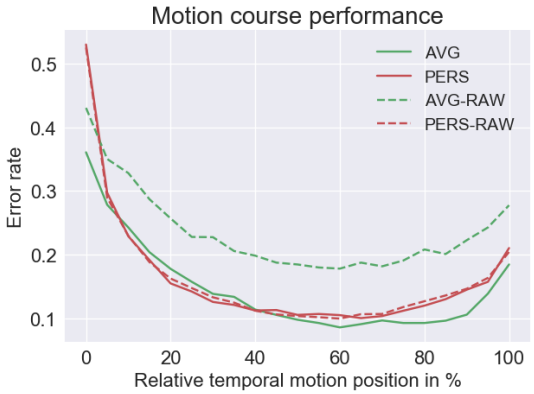

Figure 5.17: The error rate depending on the relative progress of the action. The performance achieved with the raw features is compared with the one including integrated information. The distinct sensor signals across the subjects reduce the generalization ability of the *average* model.

Raw Sensor Data

It is a matter of time until the integration of even low sensor bias signals leads inevitably to drift, if no recalibration is periodically performed. One way to avoid the continuous bias accumulation is to use the raw sensor signals without any integration. Also applications where the integration is not possible due to limited computational resources have to rely on the raw data. Figure 5.17 depicts the classification performance when only the raw sensor data is used in comparison to the one using also integrated features[6].

The inter-subject generalization ability of the *average* model significantly drops if only the sensor data is used, because of the person-specific nature of the raw signals. The *personalized* model is basically not affected, since it uses in both cases mostly the raw signals. Hence, personalized learning is even more valuable when the input is limited

6 Similar to the selection process in Section 5.3.3, we determined the most relevant features of the raw sensor data.

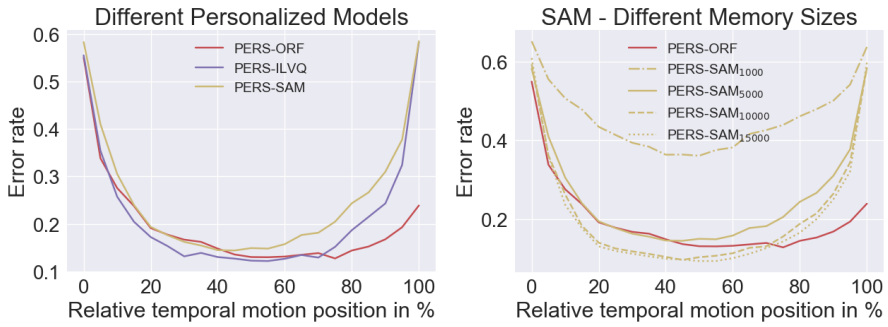

Figure 5.18: The classification performance depending on the relative progress of the action on the basis of different personalized classifiers.

to the raw sensor data.

Results of SAM and ILVQ

We also evaluated the SAM and ILVQarchitecture as personalized models within this task. Thereby, we use the same architecture for ILVQ as in the outdoor robotics application described in Section 5.1 (*COSMOS* prototype-placement strategy with metric learning on basis of GMLVQ). Both algorithms are distance-based classifiers, hence, each dimension was normalized, using a small sample (10% of the data) of the personalized data, to compensate for different feature scales. Furthermore, the learning rate of ILVQ was estimated on a small sample as well. Both distance-based models perform the classification exclusively on the basis of the current state. In contrast to the ORF, they did not benefit from stacking cosine transformation of the recent past, therefore their prediction is based on the current sample only. Figure 5.18 shows on the left the resulting error rates depending on the relative progress of the action. ILVQ achieves competitive results in comparison to ORF. In fact it has even a better performance when the ORF is limited to make a prediction based on the current sample as well. SAM performs slightly worse which is mainly caused by its unsupervised compression scheme as indicated by Figure 5.18 on the right: An increase of the memory limit, which results in fewer compression cycles, leads to a drastically increased performance. As already noted in Sections 4.4.6, 5.2.2, using the supervised signal within the compression seems a promising step to improve the SAM architecture.

5.3.4 *Discussion*

In this section, we analyzed the application of personalized online machine learning models in the task of online action classification. Using the Xsens body suit, we recorded a large motion database with over 2750 actions. Four different subjects repetitively performed various motion sequences which are categorized into sixteen different classes. The data was enriched with normalization techniques to improve the inter-subject generalization. We performed a forward feature selec-

tion, which showed that only a few IMUs are necessary to obtain a high performance. Using state-of-the-art machine learning algorithms, we compared personalized online models against average-user offline models. It turns out that personalized models are very efficient learners yielding better results with a small amount of data which indicates that the motions are indeed performed in a personalized way. The performance gain is particularly pronounced when only the raw sensor signals are used without any integration. The advantage is likely to increase with additional data.

The analysis is based on pre-labeled data. Since the long-term goal is to utilize personalized online models in a real-world application, it is required to obtain online ground truth information. In Section 3.1.2, several feasible ways of obtaining such information are discussed. Regarding the application at hand, the approach of using an offline model which delivers the labels with a tolerable delay seems the most promising one. Additionally, manually provided labels could be incorporated as well. There are further interesting possibilities to extend the current analysis. In particular, an investigation of the personal data variance between different recording sessions is crucial to obtain a robust estimate of the expected benefits in practice due to personalized online learning. Another promising starting point is the fusion of offline average models with online personalized ones to avoid a cold start at the beginning.

Our study constitutes only a first step on the way to apply personalized online learning in the task of actively supporting human motions. However, the results show that such models can offer added value, in terms of an online adaptation to individual motion sequences. Thereby, they achieve a higher performance on basis of less complex models, requiring only few training data. Hardware driven by such approaches could enable fascinating adaptation cycles between human and system, which both continuously adapt to each other to maximize the system's utility. This is especially interesting in lifelong-learning scenarios such as the support of permanently handicapped people with highly personalized prostheses.

6 CONCLUSION

In this thesis, we advocated incremental machine learning as an attractive alternative to the widespread offline learning in the domain of supervised classification. We contributed various new algorithms and consistently published the source code or even integrated the methods within the MOA framework. Hence, our research is highly transparent and enables straightforward reproducibility, facilitating the work for fellow researchers.

First, we targeted **stationary environments**, tackling the common problem of practitioners to select suitable incremental algorithms for a given task. Guidance was provided in terms of an extensive survey that compared state-of-the-art methods in key properties. The main findings were compactly presented in form of an overview table, streamlining the choice of a suitable algorithm. We addressed the stability-plasticity dilemma in case of the LVQ by contributing the first incremental prototype-placement strategy which is not based on heuristics, but instead directly minimizes the cost function. This mathematically sound approach is especially able to handle overlapping distributions, a task where heuristics mostly fail. Methods based on growing models have to decide on the fly when to add new model parameters. Inevitably, a trade-off arises between adaptation speed and model complexity / run time. In this regard, an improvement for the VFDT was provided. We replaced its regular scheme of split evaluations, which is based on an predefined interval. Instead, we applied a data-driven approach that uses local information to predict the split-time. It drastically reduced the run time without affecting the classification performance.

Subsequently, we focused on **non-stationary environments**, where the underlying distribution may change in arbitrary ways. A novel method was proposed, which is able to detect the drift characteristics within a dataset. Such information gives more insight into the task at hand and facilitates the choice of a proper learning algorithm. Concretely, we extracted characteristics of commonly used real-world datasets to assess their suitability as a benchmark for concept drift evaluation. One major contribution is the novel SAM architecture, a biologically inspired algorithm that can handle different types of concept drift. Its key idea is to establish a consistency between a short- and long-term memory, leading to an innovative way of integrating new knowledge in the current representation. The architecture is well-suited for lifelong-learning applications because the level of abstraction is adaptively increased, continuously balancing memory demands and classification performance. The robustness was further increased on the basis of a bagging ensemble that applies a set of highly diverse SAMs to boost the performance even more.

We also shed light on possible **application scenarios** and discussed viable options to obtain ground truth information on the fly, a neces-

sary prerequisite for supervised incremental learning. In particular, we applied the proposed algorithms within three completely different real-world tasks, highlighting the wide range of applicability. One task utilized prototype-based learning within an outdoor scenario on a mobile robot. The potential of personalized online learning was assessed in two different tasks. One was about predicting the driver behavior at intersections encountered on the daily commute, whereas the other tackled online motion recognition based on IMU signals. Adaptive personalized learning turned out to be beneficial in both scenarios, offering multiple advantages over static average-user models, demanding further research and a stronger consideration in practice.

6.1 HIGH-LEVEL INSIGHTS

In the course of the thesis, we could confirm established learning principles and even provide new insights. These are summarized in the following.

- Our proposed prototype-placement strategy confirms that learning on basis of formally backed optimization is preferable over heuristics most of the times.
- The idea of local split-time prediction showcases that local adaptation guided by local information has various advantages over globally predefined mechanisms.
- Various building blocks of SAM architecture are based on error minimization in retrospective. This mechanism offers a valuable handle to dynamically adapt the system, leading to a higher robustness and simpler usability in comparison to systems based on static and predefined configurations.
- Dedicated memory models turn out to be effective in various scenarios. This concept can be transferred to applications in terms of fine-grained situation-specific memories or models, that are applied according to the given situation.
- Seamless blending between memory and model represents an efficient way of explicitly editing both, memory and model, at the same time. The robust handling of the stability-plasticity dilemma by such an architecture indicates that tight coupling between memory and model on basis of the same representation is an effective mechanism for adaptive learning.

6.2 OUTLOOK

Although this thesis addressed various challenges by corresponding contributions, there are still many open questions remaining that offer interesting opportunities for future research.

One step was taken towards the path of extracting drift characteristics from a given dataset. Various aspects are not yet determined such as the timing and the frequency of occurring drift, corresponding drift patterns, e.g. abrupt versus incremental as well as the presence of reoccurring concepts. These properties could be indirectly inferred from the performance development over time, but also other approaches relying on data visualization or distribution models could be viable.

We exclusively treated incremental learning for supervised classification. Research on incremental learning in the context of regression is relatively rare, even though there is a myriad of possible applications. Therefore, it constitutes an appealing research direction. Some contributions of this thesis can easily be transferred into the regression setting. For instance, the SAM architecture can be reformulated by defining conflicting instances in terms of a target distance. In the same way, the split-time prediction for decision trees can be transferred, by replacing the class impurity function.

The SAM architecture provides a way of establishing a consistency between two memories. However, consistency is continuously obtained based on the current concept only, and information that has been classified once as inconsistent is irreversibly erased. Especially in case of reoccurring concepts or lifelong-learning scenarios, information may become inconsistent with the current belief but turn out to be relevant again at later stage. Even though the extension to an ensemble yields a variety of SAM architectures, it only partially solves this problem. An explicit construction of multiple memories that capture an own consistency seems more viable. Analogous to the human episodic memory (Tulving et al., 1972), they could cover different time periods or even be hierarchically arranged, operating on different time scales. Similarly, the construction of semantic-specific memories that are kept consistent over time, potentially acting on different levels of abstraction, seems an interesting idea that merits pursuing.

Our work regarding personalized online learning can be followed up by transferring such a system to the actual hardware and providing further empirical evidence of the benefits within the concrete applications. We demonstrated that personalized online learning is feasible on the basis of few data samples in an efficient way. It allows systems to operate on their own without the necessity of a network connection. However, further steps are required to develop such approaches beyond the prototype stage. It is mandatory to ensure the safe application of such systems and to provide performance guarantees. Currently, learning occurs from scratch solely depending on the individual input. If the system gets initially misleading data instances, we have to make sure that it recovers very quickly, or even ignores them from the beginning. The input can also be very noisy or even completely unexpected, where still a safe reaction is mandatory. One part of the solution could be to integrate rejections as output, allowing the system to ignore data instances where its confidence is not high enough (Scheme, Hudgins, & Englehart, 2013; Fischer, Hammer, & Wersing, 2015b). Another important building block is to have fall-back mechanisms which could rely on robust models acquired via batch

learning on large datasets. Such models can be thoroughly tested to ensure a baseline performance. In particular, they can mitigate the "cold-start issue" of personalized models, which initially have a low performance.

High transparency is another important aspect to increase trust in and acceptance of autonomous systems (Cramer et al., 2008). This implies a high transparency of the online adaptation process in our case. If the user understands the basic functionality of the system it can explicitly provide inputs that develop the system into a desired direction. Transparency could be provided based on explicit explanations, as it is often done by recommender systems (Tintarev & Masthoff, 2007), or on model visualizations (Verbert, Parra, Brusilovsky, & Duval, 2013; Schulz et al., 2015).

APPENDIX

A.1 DETAILED RESULTS

A.1.1 *Motion Classification*

For the sake of completeness, we report the average error rates of all experiments in Table A.1.

Table A.1: Average error rates of all experiments. Results of the table at the top are based on all available features including post-integrations and normalizations. Whereas, the second table lists the error rates that were achieved using only the raw sensor signals. In both cases, three feature types and five segments were used for both models.

Feature set	#Dimensions		Error rate		Error rate	
	AVG	PERS	AVG	PERS	PERS-1.half	PERS-2.half
Single frame	33	27	17.72	18.60	17.34	9.86
Stacked	990	810	11.68	14.51	22.95	6.07
DCT	165	135	11.57	13.46	21.35	5.58

Feature set	#Dimensions		Error rate		Error rate	
	AVG	PERS	AVG	PERS	PERS-1.half	PERS-2.half
Single frame	35	35	24.67	17.23	23.74	10.80
Stacked	1050	1050	19.00	14.88	21.83	7.94
DCT	175	175	17.90	14.27	21.51	7.03

A.1.2 *Practice-oriented survey*

In the following we provide the results of the experiments grouped by each dataset to facilitate the comparison between the three different settings.

	Accuracy								Complexity							
Border	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Setting 1	**99.4**	98.4	97.6	96.8	96.5	96.0	35.5	96.4	797	251	3.7k	193	1.9k	750	**9**	12
Setting 2	**99.4**	98.8	96.3	97.1	96.4	96.2	33.7	96.4	1.1k	1.5k	8.0k	286	2.6k	875	**9**	12
Setting 3	**98.5**	97.6	94.0	94.7	88.4	88.0	37.5	94.4	-	-	-	-	-	-	-	-
Overlap	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Setting 1	82.2	81.0	**83.7**	83.0	81.7	83.4	67.6	66.4	10k	7.2k	2.3k	235	1.9k	900	**12**	16
Setting 2	81.8	80.7	81.2	82.1	81.0	**82.2**	68.4	66.4	10.9	9.2 k	31k	496	1.9k	546	**12**	16
Setting 3	**81.7**	78.8	78.2	81.1	72.7	74.8	67.9	67.5	-	-	-	-	-	-	-	-
Letter	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Setting 1	97.0	**97.1**	93.2	93.9	87.0	70.0	56.4	63.4	131k	140k	168k	14k	51k	8.4k	**442**	832
Setting 2	96.4	**97.5**	92.2	95.1	84.9	67.4	53.9	63.4	163	173k	473k	11k	70 k	6.5k	**442**	832
Setting 3	91.3	**92.7**	75.4	88.4	79.3	35.4	41.0	64.2	-	-	-	-	-	-	-	-
SUSY	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Setting 1	DNF	DNF	**79.5**	79.0	DNF	DNF	78.7	73.5	DNF	DNF	86.0	51.2	DNF	DNF	**19**	72
Setting 2	DNF	DNF	**79.3**	73.4	DNF	DNF	77.0	73.5	DNF	DNF	1.5M	453k	DNF	DNF	**19**	72
Setting 3	DNF	DNF	**79.3**	78.5	DNF	DNF	78.7	73.5	-	-	-	-	-	-	-	-
Outdoor	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Setting 1	70.9	**71.4**	71.0	66.9	68.5	70.9	23.2	60.5	43k	42k	8.8k	11k	6.2k	12k	**880**	1.7k
Setting 2	**71.9**	69.0	57.6	63.2	69.4	71.6	25.7	60.5	40k	53k	925	11 k	7.3k	8.3k	**880**	1.7k
Setting 3	**86.4**	82.3	34.2	82.6	68.5	73.3	18.0	65.0	-	-	-	-	-	-	-	-
COIL	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Setting 1	**96.5**	93.2	92.9	94.3	89.2	91.5	12.4	90.0	58k	93k	61k	18k	9.2k	36k	**2.2k**	4.2k
Setting 2	**95.6**	92.5	89.9	93.6	88.1	83.6	12.1	90.0	180k	113k	14k	17 k	6.1k	15k	**2.2k**	4.2k
Setting 3	75.4	66.3	66.6	**79.1**	58.7	63.1	9.6	70.2	-	-	-	-	-	-	-	-

	Off-line accuracy								Complexity							
DNA	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Setting 1	**94.9**	**94.9**	89.6	92.1	90.5	88.8	93.0	88.4	237k	190k	5.0k	33k	1.6k	55k	**543**	1.1k
Setting 2	**95.2**	95.1	88.2	90.8	92.0	86.7	93.3	88.4	333k	327k	500	42k	1.8k	76k	**543**	1.1k
Setting 3	**89.5**	**89.5**	73.1	84.6	67.9	49.1	84.7	86.1	-	-	-	-	-	-	-	-
	Accuracy								Complexity							
USPS	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Setting 1	95.4	**95.6**	92.5	91.4	90.3	92.1	89.0	75.4	710k	520k	33k	15k	9.6k	106k	**2.6k**	5.1k
Setting 2	**95.6**	94.7	91.4	91.4	88.9	89.7	89.8	75.4	744k	978k	61k	91k	12k	50k	**2.6k**	5.1k
Setting 3	**96.7**	96.6	84.5	92.7	86.6	88.8	88.5	76.0	-	-	-	-	-	-	-	-
Isolet	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Setting 1	96.2	**96.7**	92.5	92.0	90.0	91.9	91.5	80.1	2.8M	3.4M	31k	21k	**12.0k**	322k	16k	32k
Setting 2	**96.5**	96.4	92.0	89.4	90.7	88.5	89.6	80.1	6.2M	4.2M	123k	67k	**14k**	159k	16k	32k
Setting 3	**93.6**	92.9	69.2	84.7	76.3	80.7	74.3	75.2	-	-	-	-	-	-	-	-
MNIST	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Setting 1	DNF	**98.5**	94.3	94.8	92.4	89.1	86.0	55.6	DNF	14M	111k	315k	73k	397k	**7.9k**	16k
Setting 2	DNF	**98.1**	95.0	94.3	91.3	84.2	87.7	55.6	DNF	16M	253k	1.2M	162k	169k	**7.9k**	16k
Setting 3	DNF	**97.5**	87.1	90.8	89.0	86.5	83.7	56.5	-	-	-	-	-	-	-	-
Gisette	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}	ISVM	LASVM	ORF	ILVQ	LPP_{CART}	IELM	SGD_{Lin}	NB_{Gauss}
Setting 1	**98.0**	97.9	94.6	93.0	94.2	91.4	93.1	75.4	7.0M	6.2M	4.7k	263k	**2.5k**	2.5M	5.0k	20k
Setting 2	97.8	**97.9**	92.7	94.1	94.0	83.8	95.4	75.4	7.0M	6.7M	16k	470k	**3.0k**	1.0M	5.0k	20k
Setting 3	96.3	**96.4**	90.3	91.1	86.7	80.5	92.1	74.0	-	-	-	-	-	-	-	-

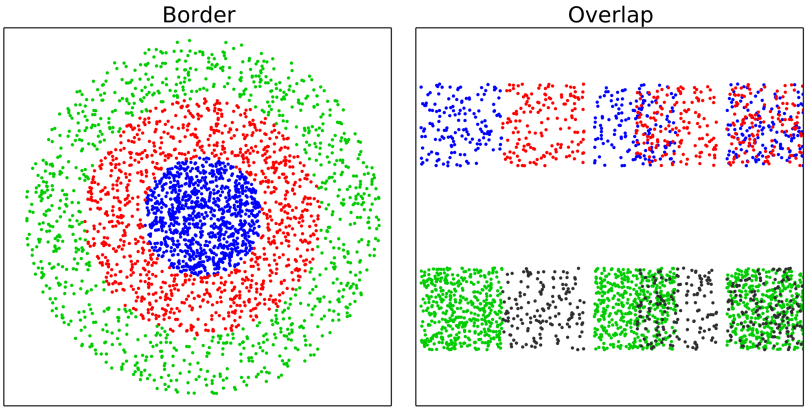

Figure A.1: The artificial data sets Border (on the left left) and Overlap (right). Different classes are coded by different colors.

A.2 DATASETS

A.2.1 *Artificial*

Radial basis function (RBF) Gaussian distributions with random initial positions, weights and standard deviations are generated in d-dimensional space. The weight controls the partitioning of the examples among the Gaussians. We used the RBF generator in MOA (100 dim., 50 classes, 100 centroids) (Losing et al., 2018a). This dataset is i.i.d.

Random Tree A random decision tree is constructed by randomly splitting along the attributes as well as assigning random classes to each leaf. Numeric and nominal attributes are supported and the tree depth can be predefined. Instances are generated by uniform sampling along each attribute. Traversing the tree with the instance determines the corresponding class label. This dataset is i.i.d. and was initially proposed by Domingos and Hulten (2000). It is regarded as an easy task for decision trees. We used the Random tree generator in MOA (100 numeric dim., 100 nominal dim., 25 classes).

Border This two-dimensional dataset consists of three circular classes and is depicted in Figure A.1 on the left. Each class is represented by 1000 uniformly distributed points (Losing et al., 2015).

Overlap The dataset is depicted in Figure A.1 on the right. Squares of uniformly distributed classes are overlapping each other with various degrees. The upper row classes have the same densities whereas, the green class below is three times denser than the black one (Losing et al., 2015).

Interchanging RBF Fifteen Gaussians with random covariance matrices are replacing each other every 3000 samples (Losing et al.,

2016c). Thereby, the number of Gaussians switching their position increases each time by one until all are simultaneously changing their location. This allows to evaluate an algorithm in the context of abrupt drift with increasing strength. Altogether 66 abrupt drifts are occurring within this dataset.

Moving RBF Gaussian distributions with random initial positions, weights and standard deviations are moved with constant speed v in d-dimensional space. The weight controls the partitioning of the examples among the Gaussians. We used the Random RBF generator in MOA (Bifet, Holmes, Kirkby, & Pfahringer, 2010) with the same parametrization as by Bifet et al. (2013) (10 dimensions, 50 Gaussians, 5 classes, v=0.001).

LED-Drift Each instance is reperesented by 24 boolean features with 17 of them being irrelevant. The remaining seven features correspond to segments of a seven-segment LED display. The goal is to predict the digit displayed on the LED display, where each feature has a 10% chance of being inverted. Drift is generated by swapping the relevant features with irrelevant ones. We used the LED-Drift generator in MOA (7 drifting dimensions, 10% noise).

Poker One million randomly drawn poker hands are represented by five cards each encoded with its suit and rank. The class is the poker hand itself such as one pair, full house and so forth. The original form available at UCI (Dheeru & Karra Taniskidou, 2017) has no drift, since the poker hands are static and instances are randomly generated. However, we used the version presented by Bifet et al. (2013), containing virtual drift via sorting the instances by rank and suit.

SEA Concepts This dataset was proposed by Street and Kim (2001) and consists of 50000 instances with three attributes of which only two are relevant. The two class decision boundary is given by $f_1 + f_2 = \theta$, where f_1, f_2 are the two relevant features and θ a predefined threshold. Abrupt drift is simulated with four different concepts, by changing the value of θ every 12500 samples. Also included are 10% of noise.

Rotating Hyperplane A hyperplane in d-dimensional space is defined by the set of points x that satisfy $\sum_{i=1}^{d} w_i x_i = w_0$. The position and orientation of the hyperplane are changed by continuous addition of a term δ to the weights $w_i = w_i + \delta$. We used the Random Hyperplane generator in MOA with the same parametrization as proposed by Bifet et al. (2013) (10 dimensions, 2 classes, δ=0.001).

Moving Squares Four equidistantly separated, squared uniform distributions are moving in horizontal direction with constant speed (Losing et al., 2016c). The direction is inverted whenever the leading square reaches a predefined boundary. Each square represents a different class. The added value of this dataset is the

predefined time horizon of 120 examples before old instances may start to overlap current ones. This is especially useful for dynamic sliding window approaches, allowing to test whether the size is adjusted accordingly.

Transient Chessboard Virtual drift is generated by revealing successively parts of a chessboard (Losing et al., 2016c). This is done square by square randomly chosen from the whole chessboard such that each square represents an own concept. Every time after four fields have been revealed, samples covering the whole chessboard are presented. This reoccurring alternation penalizes algorithms tending to discard former concepts. To reduce the impact of classification by chance we used eight classes instead of two.

Mixed Drift The datasets Interchanging RBF, Moving Squares and Transient Chessboard are arranged next to each other and samples of these are alternately introduced (Losing et al., 2016c). Therefore, incremental, abrupt and virtual drift are occurring at the same time, requiring a local adaptation to different drift types.

A.2.2 *Real-world*

MNIST The MNIST is the most popular database of handwritten digits and was introduced by Lecun et al. (1998). The digits have been size-normalized and centered in grayscaled images of 28×28 pixels. The objective is to classify the images into the 10 digits. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

MNIST-8M Loosli et al. (2007) used pseudo-random deformations and translations to extended the well known MNIST database (Lecun et al., 1998) to eight million instances. The ten handwritten digits are encoded in 782 binary features. The dataset is already shuffled in its original publication and therefore contains no concept drift.

Letter The objective is to categorize black-and-white images, represented by 16 primitive numerical attributes, into one of the 26 capital letters in the English alphabet (Frey & Slate, 1991). The character images were based on 20 different fonts and each letter was randomly distorted.

SUSY The objective is to classify whether simulated particle collisions lead to the generation of new supersymmetric particles or not and was initially published by Baldi et al. (2014). The data has been produced using Monte Carlo simulations. Each instance consists of eight kinematic features and 10 derived high-level features.

COIL Each of the 100 different objects is depicted in 72 different views in 128×128 pixel RGB images (S. A. Nene & Murase, 1996). These are encoded within a 21-dimensional RG-Chromaticity (Jain & Li, 2005) space. As proposed by Wallraven et al. (2003), a subset of 17 views per object, resulting in views every $20\,^{\circ}$, are used for training. The remaining 55 views are used for the evaluation.

DNA Splice junctions are points on a DNA sequence at which "superfluous" DNA is removed during the process of protein creation in higher organisms (Noordewier, Towell, & Shavlik, 1991). Exons are the parts of the DNA sequence retained after splicing and introns the spliced out part. The objective is to categorize a given DNA sequence into the 3 classes: exon/intron boundaries (EI), intron/exon boundaries(IE), neither of them.

USPS This dataset consist of grayscale images with 16×16 pixels (Hull, 1994). The images contain handwritten digits and the goal is to categorize them into the corresponding 10 classes.

Isolet The goal is to recognize the spoken letters of the alphabet. Various common features from the audio processing domain such as spectral coefficient,; contour features, sonorant features, pre-sonorant features, and post-sonorant features are encoding the signal (Cole & Fanty, 1990). The recording was done based on 150 different subjects.

Gisette The data set was is based on the MNIST database. The goal is to distinguish between the written number 4 and 9. The original data were modified for the purpose of the feature selection challenge (Guyon, Gunn, Nikravesh, & Zadeh, 2006). In particular, pixels were samples at random in the middle top part of the feature containing the information necessary to disambiguate both numbers and higher order features were created as products of these pixels to plunge the problem in a higher dimensional feature space. Various noise features were added having no predictive power. The order of the features and patterns were randomized.

Forest Cover Type Assigns cartographic variables such as elevation, slope, soil type, ... of 30×30 meter cells to different forest cover types. Only forests with minimal human-caused disturbances were used, so that resulting forest cover types are more a result of ecological processes. It is often used as a benchmark for drift algorithms (Bifet et al., 2013; Gama et al., 2003; Oza & Russell, 2001).

Airline The Airline data set was inspired by the regression data set from Ikonomovska[1]. The task is to predict whether a given flight

1 https://kt.ijs.si/elena_ikonomovska/data.html

Figure A.2: Objects, positioned in a garden, are approached by a mobile robot under different lighting conditions. Each row shows the first, fifth and tenth image of one approach.

will be delayed or not based on seven attributes encoding various information on the scheduled departure. This dataset is often used to evaluate concept drift classifier and is temporally ordered.

HIGGS This task consists of eleven million simulated particle collisions and was initially published by Baldi et al. (2014). The goal of this binary classification problem is to distinguish between a signal process producing Higgs bosons and a background process. The data consist of low-level kinematic features recorded as well as some derived high-level indicators.

Outdoor We obtained this data set from images recorded by a mobile in a garden environment (Losing et al., 2015). The task is to classify 40 different objects, each approached ten times under varying lighting conditions affecting the color based representation (see Figure A.2). Each approach consists of 10 images and is represented in temporal order within the dataset. The objects are encoded in a normalized 21-dimensional RG-Chromaticity histogram.

SPAM Katakis et al. (2009) developed the Spam Corpus data set was on the basis of SpamAssassin[2] data collection covering an extended time period. The goal is to classify between spam or

2 http://spamassassin.apache.org/

Figure A.3: Images of the Rialto bridge recorded at different times of day. The color based representation of the buildings changes significantly during each of the 20 recorded days. Source of the images: https://www.skylinewebcams.com.

ham (not spam). Each attribute encodes the occurrence of one of the 39917 words within an e-mail.

Weather Elwell and Polikar (2011) introduced this dataset. In the period of 1949-1999 eight different features such as temperature, pressure wind speed etc. were measured at the Offutt Air Force Base in Bellevue, Nebraska. The target is to predict whether it is going to rain on a certain day or not. The dataset contains 18159 instances with an imbalance towards no rain (69%).

Electricity This problem is often used as a benchmark for concept drift classification. Initially described by Harries and Wales (1999), it was used thereafter for several performance comparisons (Baena-Garcıa et al., 2006; Kuncheva & Plumpton, 2008; Bifet et al., 2013; Gama et al., 2004). A critical note to its suitability as a benchmark was given by Žliobaite (2013). The dataset holds information of the Australian New South Wales Electricity Market, whose prices are affected by supply and demand. Each sample, characterized by attributes such as day of week, time stamp, market demand etc., refers to a period of 30 minutes and the class label identifies the relative change (higher or lower) compared to the last 24 hours.

Rialto Bridge Timelapse Ten of the colorful buildings next to the famous Rialto bridge in Venice are encoded in a normalized 27-dimensional RGB histogram (Losing et al., 2016c). We obtained the images from time-lapse videos captured by a webcam with fixed position. The recordings cover 20 consecutive days during may-june 2016. Continuously changing weather and lighting conditions affect the representation, generating natural concept drift as shown in Figure A.3.

Physical Activity Monitoring (PAMAP) This activity recognition task includes eighteen different activities performed by up to nine different subjects and comprises ten hours of recorded data in total Reiss and Stricker (2012). The features are obtained from three inertial measurement units (IMU) and a heart rate monitor. The IMUs have a sampling rate of 100Hz and are located on the chest, the dominant wrist and ankle. It is a frequently used benchmark dataset (Reyes-Ortiz et al., 2016; Ordóñez & Roggen, 2016; Gan & Tao, 2015).

KDD99 This intrusion detection data set is well known and widely used to analyze the performance of data stream learning algorithms (Amini et al., 2014; Aggarwal et al., 2003). It simulates different types of cyber attacks and provides the highly imbalanced data in temporal order. Seventeen different classes are incorporated within the 5 million instances which are encoded with 41 attributes.

BIBLIOGRAPHY

Abe, S. (2010). Feature selection and extraction. In *Support vector machines for pattern classification* (pp. 331–341). Springer.

Acerbi, A., Ghirlanda, S., & Enquist, M. (2012). The logic of fashion cycles. *PLOS ONE*, *7*(3), 1–9.

Ade, R. & Deshmukh, P. (2013). Methods for incremental learning: A survey. *International Journal of Data Mining & Knowledge Management Process*, *3*(4), 119.

Aggarwal, C. C. (2014). *Data classification: Algorithms and applications* (1st). Chapman & Hall/CRC.

Aggarwal, C. C., Han, J., Wang, J., & Yu, P. S. (2003). A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on very large data bases - volume 29* (pp. 81–92). VLDB '03. Berlin, Germany: VLDB Endowment.

Aggarwal, J. & Ryoo, M. (2011). Human activity analysis: A review. *ACM Computing Surveys*, *43*(3), 16:1–16:43.

Aguirre, E., Mahr, D., Grewal, D., de Ruyter, K., & Wetzels, M. (2015). Unraveling the personalization paradox: The effect of information collection and trust-building strategies on online advertisement effectiveness. *Journal of Retailing*, *91*(1), 34–49.

Ahmed, N., Natarajan, T., & Rao, K. R. (1974). Discrete cosine transform. *IEEE Transactions on Computers*, *C-23*(1), 90–93.

Akaike, H. (1998). Information theory and an extension of the maximum likelihood principle. In *Selected papers of hirotugu akaike* (pp. 199–213). Springer.

Akata, Z., Perronnin, F., Harchaoui, Z., & Schmid, C. (2014). Good practice in large-scale learning for image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(3), 507–520.

Alexander, D. C. (2017). *Natural disasters*. Routledge.

Alippi, C., Boracchi, G., & Roveri, M. (2013). Just-in-time classifiers for recurrent concepts. *IEEE Transactions on Neural Networks and Learning Systems*, *24*(4), 620–634.

Alippi, C. & Roveri, M. (2008a). Just-in-time adaptive classifiers—part i: Detecting nonstationary changes. *IEEE Transactions on Neural Networks*, *19*(7), 1145–1153.

Alippi, C. & Roveri, M. (2008b). Just-in-time adaptive classifiers—part ii: Designing the classifier. *IEEE Transactions on Neural Networks*, *19*(12), 2053–2064.

Amershi, S. & Cakmak, M. (2014). Power to the people: The role of humans in interactive machine learning. *AI Magazine*.

Amini, A., Wah, T. Y., & Saboohi, H. (2014). On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology*, *29*(1), 116–141.

Arthur, D. & Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual acm-*

siam symposium on discrete algorithms (pp. 1027–1035). Society for Industrial and Applied Mathematics.

Atkinson, R. C. & Shiffrin, R. M. (1968). Human memory: A proposed system and its control processes. *The psychology of learning and motivation*, *2*, 89–195.

Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. *Computer networks*, *54*(15), 2787–2805.

Baena-Garcıa, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., & Morales-Bueno, R. (2006). Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams* (Vol. 6, pp. 77–86).

Bai, X., Ren, P., Zhang, H., & Zhou, J. (2015). An incremental structured part model for object recognition. *Neurocomputing*, *154*, 189–199.

Baldi, P., Sadowski, P., & Whiteson, D. (2014). Searching for Exotic Particles in High-Energy Physics with Deep Learning. *Nature Commun. 5*. arXiv: 1402.4735 [hep-ph]

Ball, D., Coelho, P. S., & Vilares, M. J. (2006). Service personalization and loyalty. *Journal of services marketing*, *20*(6), 391–403.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157–166.

Bennett, W. L. (2012). The personalization of politics: Political identity, social media, and changing patterns of participation. *The ANNALS of the American Academy of Political and Social Science*, *644*(1), 20–39.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509–517.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems* (pp. 2546–2554).

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., & Cox, D. D. (2015). Hyperopt: A python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, *8*(1), 014008.

Bermejo, S., Cabestany, J., & Payeras-Capellà, M. (1998). A new dynamic lvq-based classifier and its application to handwritten character recognition. In *European symposium on artificial neural networks ESANN*.

Bickel, S., Brückner, M., & Scheffer, T. (2009). Discriminative learning under covariate shift. *Journal of Machine Learning Research*, *10*(Sep), 2137–2155.

Biehl, M., Bunte, K., & Schneider, P. (2013). Analysis of flow cytometry data by matrix relevance learning vector quantization. *PLoS ONE*.

Bifet, A. & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Siam international conference on data mining (sdm)* (Vol. 7, p. 2007). SIAM.

Bifet, A. & Gavaldà, R. (2009). Adaptive learning from evolving data streams. In *Advances in intelligent data analysis viii: 8th international symposium on intelligent data analysis*.

Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010). Moa: Massive online analysis. *The Journal of Machine Learning Research*, *11*, 1601–1604.

Bifet, A., Holmes, G., & Pfahringer, B. (2010). Leveraging bagging for evolving data streams. In *Machine learning and knowledge discovery in databases* (pp. 135–150). Springer.

Bifet, A., Holmes, G., Pfahringer, B., & Frank, E. (2010). Fast perceptron decision tree learning from evolving data streams. In *Advances in knowledge discovery and data mining* (pp. 299–310). Berlin, Heidelberg: Springer Berlin Heidelberg.

Bifet, A., Pfahringer, B., Read, J., & Holmes, G. (2013). Efficient data stream classification via probabilistic adaptive windows. In *Proceedings of the 28th annual acm symposium on applied computing* (pp. 801–806). SAC '13. Coimbra, Portugal: ACM.

Bifet, A., Zhang, J., Fan, W., He, C., Zhang, J., Qian, J., ... Pfahringer, B. (2017). Extremely fast decision tree mining for evolving data streams. In *Proc. of the 23rd sigkdd international conference on knowledge discovery and data mining*. KDD '17. Halifax, NS, Canada.

Bifulco, G. N., Pariota, L., Simonelli, F., & Di Pace, R. (2013). Development and testing of a fully adaptive cruise control system. *Transportation Research Part C: Emerging Technologies*, *29*, 156–170.

Biggio, B., Corona, I., Nelson, B., Rubinstein, B. I., Maiorca, D., Fumera, G., ... Roli, F. (2014). Security evaluation of support vector machines in adversarial environments. In *Support vector machines applications* (pp. 105–153). Springer.

Bikhchandani, S. & Sharma, S. (2000). Herd behavior in financial markets. *IMF Staff papers*, *47*(3), 279–310.

Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Bonferroni, C. E. (1936). *Teoria statistica delle classi e calcolo delle probabilita*. Libreria internazionale Seeber.

Bordes, A., Ertekin, S., Weston, J., & Bottou, L. (2005). Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, *6*, 1579–1619.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of compstat'2010* (pp. 177–186). Springer.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1993). Classification and regression trees.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks.

Brent, R. (1973). *Algorithms for minimization without derivatives*. Dover Books on Mathematics.

Bunte, K., Schneider, P., Hammer, B., Schleif, F.-M., Villmann, T., & Biehl, M. (2012). Limited rank matrix learning, discriminative dimension reduction and visualization. *Neural Networks*, *26*, 159–173.

Butakov, V. & Ioannou, P. (2015). Personalized driver/vehicle lane change models for ADAS. *IEEE Transactions on Vehicular Technology*, *64*(10), 4422–4431.

Cai, Z., Wen, L., Lei, Z., Vasconcelos, N., & Li, S. Z. (2014). Robust deformable and occluded object tracking with dynamic graph. *IEEE Transactions on Image Processing*, *23*(12), 5497–5509.

Calinon, S. & Billard, A. (2007). Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the acm/ieee international conference on human-robot interaction* (pp. 255–262). ACM.

Carlevarino, A., Martinotti, R., & Metta, G. (2000). An incremental growing neural network and its application to robot control. In *International joint conference on neural networks. 2000*.

Carolis, B. D., Ferilli, S., & Redavid, D. (2015). Incremental learning of daily routines as workflows in a smart home environment. *ACM*, *4*(4), 20:1–20:23.

Carpenter, G. A., Grossberg, S., & Reynolds, J. (1991). Artmap: A self-organizing neural network architecture for fast supervised learning and pattern recognition. In *Ijcnn-91-seattle international joint conference on neural networks* (Vol. 1, 863–868 vol.1).

Carpenter, G. A. & Grossberg, S. (1987a). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer vision, graphics, and image processing*, *37*(1), 54–115.

Carpenter, G. A. & Grossberg, S. (1987b). Art 2: Self-organization of stable category recognition codes for analog input patterns. *Applied optics*, *26*(23), 4919–4930.

Cauwenberghs, G. & Poggio, T. (2001). Incremental and decremental support vector machine learning. In *Advances in neural information processing systems* (pp. 409–415).

Cesa-Bianchi, N., Freund, Y., Haussler, D., Helmbold, D. P., Schapire, R. E., & Warmuth, M. K. (1997). How to use expert advice. *Journal of the ACM (JACM)*, *44*(3), 427–485.

Cesa-Bianchi, N. & Lugosi, G. (2006). *Prediction, learning, and games*. Cambridge university press.

Chang, C.-C. & Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, *2*.

Charikar, M., Chekuri, C., Feder, T., & Motwani, R. (2004). Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, *33*(6), 1417–1440.

Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. *Mobile Netw. and Appl. 19*(2).

Cole, R. & Fanty, M. (1990). Spoken letter recognition. In *Proceedings of the workshop on speech and natural language* (pp. 385–390). HLT '90. Hidden Valley, Pennsylvania: Association for Computational Linguistics.

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273–297.

Cover, T. M. & Hart, P. E. (1967). Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, *13*(1), 21–27.

Cramer, H., Evers, V., Ramlal, S., Van Someren, M., Rutledge, L., Stash, N., ... Wielinga, B. (2008). The effects of transparency on trust in and acceptance of a content-based art recommender. *User Modeling and User-Adapted Interaction*, *18*(5), 455.

Damgrave, R. G. J. & Lutters, D. (2009). The drift of the xsens moven motion capturing suit during common movements in a working environment. In *Proceedings of the 19th cirp design conference–competitive design*. Cranfield University Press.

Das, A. S., Datar, M., Garg, A., & Rajaram, S. (2007). Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th international conference on world wide web* (pp. 271–280). ACM.

Dasu, T., Krishnan, S., Venkatasubramanian, S., & Yi, K. (2006). An information-theoretic approach to detecting changes in multidimensional data streams. *Interfaces*.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Res*, *7*, 1–30.

Dheeru, D. & Karra Taniskidou, E. (2017). UCI machine learning repository.

Diehl, C. P. & Cauwenberghs, G. (2003). Svm incremental learning, adaptation and optimization. In *Proceedings of the international joint conference on neural networks, 2003.* (Vol. 4, 2685–2690 vol.4).

Ditzler, G. & Polikar, R. (2013). Incremental learning of concept drift from streaming imbalanced data. *ieee transactions on knowledge and data engineering*, *25*(10), 2283–2301.

Ditzler, G., Roveri, M., Alippi, C., & Polikar, R. (2015). Learning in nonstationary environments: A survey. *Computational Intelligence Magazine, IEEE*, *10*(4), 12–25.

Domingos, P. & Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth acm sigkdd international conference on knowledge discovery and data mining* (pp. 71–80). ACM.

Domingos, P. & Hulten, G. (2003). A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, *12*(4), 945–949.

Domingos, P. & Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, *29*(2).

Doshi, A. & Trivedi, M. M. (2011). Tactical driver behavior prediction and intent inference: A review. In *Intelligent transportation systems (itsc), 2011 14th international ieee conference on* (pp. 1892–1897). IEEE.

Dou, J., Li, J., Qin, Q., & Tu, Z. (2015). Moving object detection based on incremental learning low rank representation and spatial constraint. *Neurocomputing*, *168*.

Downs, T., Gates, K., & Masters, A. (2002). Exact simplification of support vector solutions. *Journal of Machine Learning Res*, *2*, 293–297.

Dreyfus, S. E. & Dreyfus, H. L. (1980). *A five-stage model of the mental activities involved in directed skill acquisition*. California Univ Berkeley Operations Research Center.

Dudai, Y. (2004). The neurobiology of consolidations, or, how stable is the engram? *Annual Review of Psychology*, *55*, 51–86.

Elwell, R. & Polikar, R. (2009). Incremental learning in nonstationary environments with controlled forgetting. In *2009 international joint conference on neural networks* (pp. 771–778).

Elwell, R. & Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, *22*(10), 1517–1531.

Ertekin, S., Bottou, L., & Giles, C. L. (2011). Nonconvex online support vector machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(2), 368–381.

Fayyad, U. M. & Irani, K. B. (1992). The attribute selection problem in decision tree generation. In *Aaai* (pp. 104–110).

Fdez-Riverola, F., Iglesias, E. L., Diaz, F., Méndez, J. R., & Corchado, J. M. (2007). Applying lazy learning algorithms to tackle concept drift in spam filtering. *Expert Systems with Applications*, *33*(1), 36–48.

Feng, Y. & Gallego, G. (1995). Optimal starting times for end-of-season sales and optimal stopping times for promotional fares. *Management science*, *41*(8), 1371–1391.

Fernández-Delgado, M., Cernadas, E., Barro, S., & Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, *15*, 3133–3181.

Filliat, D. (2008). Interactive learning of visual topological navigation. In *2008 ieee/rsj international conference on intelligent robots and systems* (pp. 248–254).

Firestein, S. (2012). *Ignorance: How it drives science*. OUP USA.

Fischer, G. (2001). User modeling in human-computer interaction. *User Modeling and User-Adapted Interaction*, *11*(1-2), 65–86.

Fischer, L., Hammer, B., & Wersing, H. (2015a). Certainty-based prototype insertion/deletion for classification with metric adaptation. In *Proceedings of the european symposium on artificial neural networks ESANN* (pp. 7–12).

Fischer, L., Hammer, B., & Wersing, H. (2015b). Efficient rejection strategies for prototype-based classification. *Neurocomputing*, *169*, 334–342.

Forlizzi, J. & DiSalvo, C. (2006). Service robots in the domestic environment: A study of the roomba vacuum in the home. In *Proceedings of the 1st acm sigchi/sigart conference on human-robot interaction* (pp. 258–265). ACM.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, *3*(4), 128–135.

Freund, Y. & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119–139.

Freund, Y., Schapire, R., & Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, *14*(771-780), 1612.

Frey, P. W. & Slate, D. J. (1991). Letter recognition using holland-style adaptive classifiers. *Machine Learning*, *6*, 161.

Friedman, J. H. (1997). On bias, variance, 0/1—loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, *1*(1), 55–77.

Fu, Z., Wu, D.-A. J., Ross, I., Chung, J. M., Mamelak, A. N., Adolphs, R., & Rutishauser, U. (2019). Single-neuron correlates of error monitoring and post-error adjustments in human medial frontal cortex. *Neuron*, *101*(1), 165–177.e5.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, *17*(4).

Gaber, M. M., Zaslavsky, A., & Krishnaswamy, S. (2005). Mining data streams: A review. *ACM Sigmod Record*, *34*(2), 18–26.

Gaidon, A., Harchaoui, Z., & Schmid, C. (2011). Actom sequence models for efficient action detection. In *Cvpr 2011* (pp. 3201–3208).

Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In *Advances in artificial intelligence–sbia 2004* (pp. 286–295). Springer.

Gama, J., Rocha, R., & Medas, P. (2003). Accurate decision trees for mining high-speed data streams. In *Proceedings of the ninth acm sigkdd international conference on knowledge discovery and data mining* (pp. 523–528). ACM.

Gama, J., Sebastião, R., & Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Machine Learning*, *90*(3), 317–346.

Gama, J., Žliobaite, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, *46*(4), 44.

Gan, J. & Tao, Y. (2015). Dbscan revisited: Mis-claim, un-fixability, and approximation. In *Proc. of the 2015 sigmod international conference on management of data* (pp. 519–530). SIGMOD '15. Melbourne, Victoria, Australia.

García Molina, J. F., Zheng, L., Sertdemir, M., Dinter, D. J., Schönberg, S., & Rädle, M. (2014). Incremental learning with svm for multimodal classification of prostatic adenocarcinoma. *PLOS ONE*, *9*(4), 1–14.

Garcia-Hernando, G. & Kim, T.-K. (2017). Transition forests: Learning discriminative temporal transitions for action recognition and detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 407–415.

Garcia-Martin, E. (2017). Extraction and energy efficient processing of streaming data, dissertation.

Gepperth, A. & Hammer, B. (2016). Incremental learning algorithms and applications. In *Proceedings of the european sympoisum on artificial neural networks (ESANN)*.

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, *63*(1), 3–42.

Giraud-Carrier, C. (2000). A note on the utility of incremental learning. *Ai Communications*, *13*(4), 215–223.

Gomes, H., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., ... Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning*, *106*(9), 1469–1495.

Grbovic, M. & Vucetic, S. (2009). Learning vector quantization with adaptive prototype addition and removal. In *2009 international joint conference on neural networks* (pp. 994–1001).

Griffis, J., Allendorfer, J., & P. Szaflarski, J. (2016). Voxel-based gaussian naïve bayes classification of ischemic stroke lesions in individual t1-weighted mri scans. *Journal of Neuroscience Methods*, *257*, 97–108.

Grossberg, S. (1976a). Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors. *Biological cybernetics*, *23*(3), 121–134.

Grossberg, S. (1976b). Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors. *Biological cybernetics*, *23*(3), 121–134.

Grossberg, S. (1988). Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural networks*, *1*(1), 17–61.

Gu, B., Sheng, V. S., Tay, K. Y., Romano, W., & Li, S. (2015). Incremental support vector learning for ordinal regression. *IEEE Transactions on Neural networks and learning systems*, *26*(7), 1403–1416.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, *29*(7), 1645–1660.

Guyon, I., Gunn, S., Nikravesh, M., & Zadeh, L. A. (2006). *Feature extraction: Foundations and applications (studies in fuzziness and soft computing)*. Berlin, Heidelberg: Springer-Verlag.

Harries, M. & Wales, N. S. (1999). Splice-2 comparative evaluation: Electricity pricing.

Harris, F. J. (1978). On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, *66*(1), 51–83.

Harsham, B., Watanabe, S., Esenther, A., Hershey, J., Le Roux, J., Luan, Y., ... Potluru, V. (2015). Driver prediction to improve interaction with in-vehicle hmi. In *Proc. workshop on digital signal processing for in-vehicle systems (dsp)*.

Hasenjäger, M. & Wersing, H. (2017). Personalization in advanced driver assistance systems and autonomous vehicles: A review. In *2017 ieee 20th international conference on intelligent transportation systems (itsc)* (pp. 1–7).

Hasselmo, M. E. (2006). The role of acetylcholine in learning and memory. *Current opinion in neurobiology*, *16*(6), 710–715.

Hastie, T. & Tibshirani, R. (1996). Discriminant analysis by gaussian mixtures. *Journal of the Royal Statistical Society. Series B (Methodological)*, 155–176.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

He, H., Chen, S., Li, K., & Xu, X. (2011). Incremental learning from stream data. *IEEE Transactions on Neural Networks*, *22*(12), 1901–1914.

Helsper, E. J. (2010). Gendered internet use across generations and life stages. *Communication research*, *37*(3), 352–374.

Higgins, C., Duxbury, L., & Lee, C. (1994). Impact of life-cycle stage and gender on the ability to balance work and family responsibilities. *Family Relations*, 144–150.

Hilbert, M. & López, P. (2011). The world's technological capacity to store, communicate, and compute information. *Science (New York, N.Y.) 332*, 60–5.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(8), 832–844.

Holmes, G., Kirkby, R., & Pfahringer, B. (2005). Stress-testing hoeffding trees. In *Knowledge discovery in databases: Pkdd 2005: 9th european conference on principles and practice of knowledge discovery in databases, porto, portugal, october 3-7, 2005. proc.*

Holmes, G., Richard, K., & Pfahringer, B. (2005). Tie-breaking in hoeffding trees. In *Proc. of the workshop w6 the second international workshop on knowledge discovery in data streams*.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, *2*(5), 359–366.

Hsieh, C.-J., Si, S., & Dhillon, I. S. (2014). A divide-and-conquer solver for kernel support vector machines. In *Proceedings of the 31st international conference on international conference on machine learning - volume 32* (pp. I-566–I-574). ICML'14. Beijing, China: JMLR.org.

Hull, J. J. (1994). A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *16*(5), 550–554.

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. In *Proc. of the seventh sigkdd international conference on knowledge discovery and data mining*.

Jaber, G., Cornuéjols, A., & Tarroux, P. (2013). Online learning: Searching for the best forgetting strategy under concept drift. In *Neural information processing* (pp. 400–408). Springer.

Jain, A. K. & Li, S. Z. (2005). *Handbook of face recognition*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Jain, M., v. Gemert, J., Jégou, H., Bouthemy, P., & Snoek, C. G. M. (2014). Action localization with tubelets from motion. In *2014 ieee conference on computer vision and pattern recognition* (pp. 740–747).

Jain, P., Kulis, B., Dhillon, I. S., & Grauman, K. (2009). Online metric learning and fast similarity search. In *Advances in neural information processing systems 21* (pp. 761–768). Curran Associates, Inc.

Jarvis, P. (2012). *Towards a comprehensive theory of human learning*. Routledge.

Jin, R. & Agrawal, G. (2003). Efficient decision tree construction on streaming data. In *Proc. of the ninth sigkdd international conference on knowledge discovery and data mining*. KDD '03. Washington, D.C.

Jin, Y. & Hammer, B. (2014). Computational intelligence in big data. *Computational Intelligence Magazine, IEEE*, *9*(3).

Johnson, R. & Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems* (pp. 315–323).

Joshi, P. & Kulkarni, P. (2012). Incremental learning: Areas and methods-a survey. *International Journal of Data Mining & Knowledge Management Process*, *2*(5), 43.

Katakis, I., Tsoumakas, G., Banos, E., Bassiliades, N., & Vlahavas, I. (2009). An adaptive personalized news dissemination system. *Journal of Intelligent Information Systems*, *32*(2), 191–212.

Kifer, D., Ben-David, S., & Gehrke, J. (2004). Detecting change in data streams. In *Proceedings of the thirtieth international conference on very large data bases-volume 30* (pp. 180–191). VLDB Endowment.

Kirkby, R. B. (2007). *Improving hoeffding trees* (Doctoral dissertation, The University of Waikato).

Kirstein, S., Wersing, H., Gross, H.-M., & Körner, E. (2012). A life-long learning vector quantization approach for interactive learning of multiple categories. *Neural Networks*, *28*, 90–105.

Kirstein, S., Wersing, H., & Körner, E. (2005). Rapid online learning of objects in a biologically motivated recognition architecture. In *Pattern recognition* (pp. 301–308). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kivinen, J. & Warmuth, M. K. (1997). Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, *132*(1), 1–63.

Klingelschmitt, S., Platho, M., Groß, H.-M., Willert, V., & Eggert, J. (2014). Combining behavior and situation information for reliably estimating multiple intentions. In *Ieee intelligent vehicles symposium (iv)*.

Klinkenberg, R. & Joachims, T. (2000). Detecting concept drift with support vector machines. In *Proceedings of the seventeenth international conference on machine learning* (pp. 487–494). ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Kohavi, R. (1996). Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proc. of the second international conference on knowledge discovery and data mining*. KDD'96. Portland, Oregon.

Kohonen, T. (2001). *Self-organizing maps* (3rd). Springer series in information sciences, 30. Berlin: Springer.

Kolter, J. Z. & Maloof, M. A. (2007). Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research*, *8*, 2755–2790.

Kong, K. & Jeon, D. (2006). Design and control of an exoskeleton for the elderly and patients. *IEEE/ASME Transactions on mechatronics*, *11*(4), 428–432.

Kourtellis, N., Morales, G. D. F., Bifet, A., & Murdopo, A. (2016). Vht: Vertical hoeffding tree. In *2016 ieee international conference on big data (big data)*.

Kuefler, A., Morton, J., Wheeler, T., & Kochenderfer, M. (2017). Imitating driver behavior with generative adversarial networks. *arXiv preprint arXiv:1701.06699*.

Kulić, D., Ott, C., Lee, D., Ishikawa, J., & Nakamura, Y. (2012). Incremental learning of full body motion primitives and their sequencing through human motion observation. *The International Journal of Robotics Research*, *31*(3), 330–345.

Kuncheva, L. I. & Žliobaite, I. (2009). On the window size for classification in changing environments. *Intelligent Data Analysis*, *13*(6), 861–872.

Kuncheva, L. I. & Plumpton, C. O. (2008). Adaptive learning rate for online linear discriminant classifiers. In *Structural, syntactic, and statistical pattern recognition* (pp. 510–519). Springer.

Kwapisz, J. R., Weiss, G. M., & Moore, S. A. (2010). Activity recognition using cell phone accelerometers. In *Proceedings of the fourth international workshop on knowledge discovery from sensor data* (pp. 10–18).

De-la-Torre, M., Granger, E., Radtke, P. V., Sabourin, R., & Gorodnichy, D. O. (2015). Partially-supervised learning from facial trajectories for face recognition in video surveillance. *Information Fusion*, *24*, 31–53.

Lakshminarayanan, B., Roy, D. M., & Teh, Y. W. (2014). Mondrian forests: Efficient online random forests. In *Advances in neural information processing systems* (pp. 3140–3148).

Lane, T. & Brodley, C. E. (1998). Approaches to online learning and concept drift for user identification in computer security. In *Proceedings of the fourth international conference on knowledge discovery and data mining* (pp. 259–263). KDD'98. New York, NY: AAAI Press.

Laskov, P., Gehl, C., Krüger, S., & Müller, K.-R. (2006). Incremental support vector learning: Analysis, implementation and applications. *Journal of Machine Learning Res*, *7*.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. of the IEEE*, *86*(11).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436.

LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., ... Sackinger, E., et al. (1995). Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks* (Vol. 60, pp. 53–60). Perth, Australia.

Li, C., Zhang, Y., & Li, X. (2009). Ocvfdt: One-class very fast decision tree for one-class classification of data streams. In *Proc. of the*

third international workshop on knowledge discovery from sensor data. SensorKDD '09. Paris, France.

Li, X., Dick, A., Shen, C., Van Den Hengel, A., & Wang, H. (2013). Incremental learning of 3d-dct compact representations for robust visual tracking. *IEEE transactions on pattern analysis and machine intelligence*, *35*(4), 863–881.

Li, Y., Lan, C., Xing, J., Zeng, W., Yuan, C., & Liu, J. (2016). Online human action detection using joint classification-regression recurrent neural networks. In *Computer vision – eccv 2016* (pp. 203–220). Cham: Springer International Publishing.

Liang, N., Huang, G., Saratchandran, P., & Sundararajan, N. (2006). A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, *17*(6), 1411–1423.

Liddle, A. R. (2007). Information criteria for astrophysical model selection. *Monthly Notices of the Royal Astronomical Society: Letters*, *377*(1), L74–L78.

Liebner, M., Klanner, F., Baumann, M., Ruhhammer, C., & Stiller, C. (2013). Velocity-based driver intent inference at urban intersections in the presence of preceding vehicles. *IEEE Intelligent Transportation Systems Magazine*, *5*(2), 10–21.

Lim, J., Ross, D. A., Lin, R.-S., & Yang, M.-H. (2005). Incremental learning for visual tracking. In *Advances in neural information processing systems* (pp. 793–800).

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2*(4), 285–318.

Loosli, G., Canu, S., & Bottou, L. (2007). Training invariant support vector machines using selective sampling. In *Large scale kernel machines*.

Losing, V. (2014). Master thesis: Online learning on a mobile robot.

Losing, V., Hammer, B., & Wersing, H. (2015). Interactive online learning for obstacle classification on a mobile robot. In *2015 international joint conference on neural networks (ijcnn)* (pp. 1–8). IEEE.

Losing, V., Hammer, B., & Wersing, H. (2016a). Choosing the best algorithm for an incremental on-line learning task. In *2016 24th european symposium on artificial neural networks (esann)*.

Losing, V., Hammer, B., & Wersing, H. (2016b). Dedicated memory models for continual learning in the presence of concept drift. In *Advances in neural information processing systems (nips) 29, continual learning workshop*.

Losing, V., Hammer, B., & Wersing, H. (2016c). Knn classifier with self adjusting memory for heterogeneous concept drift. In *2016 ieee 16th international conference on data mining (icdm)* (pp. 291–300).

Losing, V., Hammer, B., & Wersing, H. (2017a). Personalized maneuver prediction at intersections. In *2017 ieee 20th international conference on intelligent transportation systems (itsc)* (pp. 1–6).

Losing, V., Hammer, B., & Wersing, H. (2017b). Self-adjusting memory: How to deal with diverse drift types. In *Proceedings of the twenty-*

sixth international joint conference on artificial intelligence, IJCAI-17 (pp. 4899–4903).

Losing, V., Hammer, B., & Wersing, H. (2018a). Enhancing very fast decision trees with local split-time predictions. In *2018 ieee 16th international conference on data mining (icdm)*.

Losing, V., Hammer, B., & Wersing, H. (2018b). Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, *275*.

Losing, V., Hammer, B., & Wersing, H. (2018c). Tackling heterogeneous concept drift with the self-adjusting memory (sam). *Knowledge and Information Systems*, *54*(1), 171–201.

Losing, V., Hammer, B., & Wersing, H. (2019). Personalized online learning of whole body motions using multiple inertial measurement units. In *Ieee international conference on robotics and automation (icra) 2019*.

Losing, V., Hammer, B., Wersing, H., & Bifet, A. (2019). Tackling concept drift with a diverse self-adjusting memory ensemble. In *Submitted to international conference on data engineering (icde) 2019*.

Lou, W., Wang, X., Chen, F., Chen, Y., Jiang, B., & Zhang, H. (2014). Sequence based prediction of dna-binding proteins based on hybrid feature selection using random forest and gaussian naïve bayes. *PLOS ONE*, *9*(1), 1–10.

Lu, Y., Boukharouba, K., Boonært, J., Fleury, A., & Lecoeuche, S. (2014). Application of an incremental svm algorithm for on-line human recognition from video surveillance using texture and color features. *Neurocomputing*, *126*, 132–140.

Luo, J., Pronobis, A., Caputo, B., & Jensfelt, P. (2007). Incremental learning for place recognition in dynamic environments. In *2007 ieee/rsj international conference on intelligent robots and systems* (pp. 721–728).

Ma, S., Li, X., Ding, Y., & Orlowska, M. E. (2007). A recommender system with interest-drifting. In *Web information systems engineering – wise 2007* (pp. 633–642). Berlin, Heidelberg: Springer Berlin Heidelberg.

Maaten, L. v. d. & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(Nov), 2579–2605.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *Proc. of the fifth berkeley symposium on mathematical statistics and probability* (Vol. 1). University of California Press.

Maggi, F., Robertson, W., Kruegel, C., & Vigna, G. (2009). Protecting a moving target: Addressing web application concept drift. In *International workshop on recent advances in intrusion detection* (pp. 21–40). Springer.

Margineantu, D. D. & Dietterich, T. G. (1997). Pruning adaptive boosting. In *Proceedings of the fourteenth international conference on machine learning* (pp. 211–218). ICML '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Martinetz, T. M. & Schulten, K. J. (1991). A "neural gas" network learns topologies. In *Proceedings of the international conference on*

artificial neural networks 1991 (pp. 397–402). Amsterdam; New York: North-Holland.

Medrano, C., Plaza, I., Igual, R., Sánchez, Á., & Castro, M. (2016). The effect of personalization on smartphone-based fall detectors. *Sensors*, *16*(1), 117.

Méndez, J. R., Fdez-Riverola, F., Iglesias, E. L., Diaz, F., & Corchado, J. M. (2006). Tracking concept drift at feature selection stage in spamhunting: An anti-spam instance-based reasoning system. In *European conference on case-based reasoning* (pp. 504–518). Springer.

Metsis, V., Androutsopoulos, I., & Paliouras, G. (2006). Spam filtering with naive bayes-which naive bayes? In *Third conference on email and anti-spam (ceas* (pp. 27–28).

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, *63*(2), 81.

Minsky, M. & Papert, S. (1972). *Perceptrons: An introduction to computational geometry*. Mit Press.

Mitchell, T. M. (1997). *Machine learning* (1st ed.). New York, NY, USA: McGraw-Hill, Inc.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., G Bellemare, M., ... Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–33.

Mojet, J., Christ-Hazelhof, E., & Heidema, J. (2001). Taste perception with age: Generic or specific losses in threshold sensitivity to the five basic tastes? *Chemical senses*, *26*(7), 845–860.

Mugge, R., Schoormans, J. P., & Schifferstein, H. N. (2009). Emotional bonding with personalised products. *Journal of Engineering Design*, *20*(5), 467–476.

Muramatsu, Y., Kobayashi, H., Sato, Y., Jiaou, H., & Hashimoto, T. (2011). Quantitative performance analysis of exoskeleton augmenting devices - muscle suit - for manual worker. *International Journal of Automation Technology*, *5*, 559–567.

Nellore, K. & Hancke, G. P. (2016). A survey on urban traffic management system using wireless sensor networks. *Sensors*, *16*(2), 157.

Nelson, E. C., Verhagen, T., & Noordzij, M. L. (2016). Health empowerment through activity trackers: An empirical smart wristband study. *Computers in human behavior*, *62*, 364–374.

Nene, S. A., Nayar, S. K., & Murase, H. (1996). *Columbia object image library (coil-100)*.

Neto, F. D. N., de Souza Baptista, C., & Campelo, C. E. (2016). A user-personalized model for real time destination and route prediction. In *Intelligent transportation systems (itsc), 2016 ieee 19th international conference on* (pp. 401–407). IEEE.

Noordewier, M. O., Towell, G. G., & Shavlik, J. W. (1991). Training knowledge-based neural networks to recognize genes in dna sequences. In *Advances in neural information processing systems 3* (pp. 530–536). Morgan-Kaufmann.

Novikoff, A. (1962). On convergence proofs of perceptrons. In *Proceedings of the symposium on the mathematical theory of automata* (Vol. 12, 2, pp. 615–622).

Opelt, A., Pinz, A., & Zisserman, A. (2006). Incremental learning of object detectors using a visual shape alphabet. In *Proceedings of the 2006 ieee computer society conference on computer vision and pattern recognition-volume 1* (pp. 3–10). IEEE Computer Society.

OpenStreetMap contributors. (2017). Planet dump retrieved from https://planet.osm.org. https://www.openstreetmap.org.

Ordóñez, F. J. & Roggen, D. (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, *16*(1).

Orth, D., Kolossa, D., Sarria Paja, M., Schaller, K., Pech, A., & Heckmann, M. (2017). A maximum likelihood method for driver-specific critical-gap estimation. In *2017 ieee intelligent vehicles symposium (iv)*.

Oza, N. C. (2005). Online bagging and boosting. In *Ieee transactions on systems, man, and cybernetics* (Vol. 3, pp. 2340–2345). IEEE.

Oza, N. C. & Russell, S. (2001). Experimental comparisons of online and batch versions of bagging and boosting. In *Proceedings of the seventh acm sigkdd international conference on knowledge discovery and data mining* (pp. 359–364). ACM.

Paaßen, B., Schulz, A., Hahne, J., & Hammer, B. (2017). An EM transfer learning algorithm with applications in bionic hand prostheses. In *Proceedings of the 25th european symposium on artificial neural networks (esann 2017)* (pp. 129–134). Bruges: i6doc.com.

Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, *41*(1/2), 100–115.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pernici, F. & Del Bimbo, A. (2014). Object tracking by oversampling local features. *IEEE transactions on pattern analysis and machine intelligence*, *36*(12), 2538–2551.

Pfahringer, B., Holmes, G., & Kirkby, R. (2008). Handling numeric attributes in hoeffding trees. In *Advances in knowledge discovery and data mining*.

Platt, J. (1998). *Sequential minimal optimization: A fast algorithm for training support vector machines*.

Polikar, R., Upda, L., Upda, S. S., & Honavar, V. (2001). Learn++: An incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *31*(4), 497–508.

Poppe, R. (2010). A survey on vision-based human action recognition. *Image Vision Computing*, *28*(6), 976–990.

Pratt, J. E., Krupp, B. T., Morse, C. J., & Collins, S. H. (2004). The roboknee: An exoskeleton for enhancing strength and endurance during walking. In *Ieee international conference on robotics and automation, 2004. proceedings. icra '04. 2004* (Vol. 3, 2430–2435 Vol.3).

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... Ng, A. Y. (2009). Ros: An open-source robot operating system. In *Icra workshop on open source software* (Vol. 3, *3.2*, p. 5). Kobe, Japan.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*.

Raghupathi, W. & Raghupathi, V. (2014). Big data analytics in healthcare: Promise and potential. *Health information science and systems*, *2*(1), 3.

Read, J., Bifet, A., Pfahringer, B., & Holmes, G. (2012). Batch-incremental versus instance-incremental learning in dynamic and evolving data. In *International symposium on intelligent data analysis* (pp. 313–323). Springer.

Reed, R. D. & Marks, R. J. (1998). *Neural smithing: Supervised learning in feedforward artificial neural networks*. Cambridge, MA, USA: MIT Press.

Reiss, A. & Stricker, D. (2012). Introducing a new benchmarked dataset for activity monitoring. In *2012 16th international symposium on wearable computers* (pp. 108–109).

Reyes-Ortiz, J.-L., Oneto, L., SamÃ, A., Parra, X., & Anguita, D. (2016). Transition-aware human activity recognition using smartphones. *Neurocomputing*, *171*(Supplement C), 754–767.

Richtárik, P. & Takáč, M. (2016). Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, *156*(1), 433–484.

Rodemerk, C., Winner, H., & Kastner, R. (2015). Predicting the driver's turn intentions at urban intersections using context-based indicators. In *Intelligent vehicles symposium (iv), 2015 ieee* (pp. 964–969). IEEE.

Roetenberg, D., Luinge, H., & Slycke, P. (2009). Xsens mvn: Full 6dof human motion tracking using miniature inertial sensors. *3*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65–386.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation*. DTIC Document.

S. A. Nene, S. K. N. & Murase, H. (1996). Columbia object image library (coil-100).

Saffari, A., Leistner, C., Santner, J., Godec, M., & Bischof, H. (2009). On-line random forests. In *2009 ieee 12th international conference on computer vision workshops, iccv workshops* (pp. 1393–1400).

Salperwyck, C. & Lemaire, V. (2011). Learning with few examples: An empirical study on leading classifiers. In *The 2011 international joint conference on neural networks* (pp. 1010–1019).

Salton, G. & McGill, M. J. (1986). *Introduction to modern information retrieval*. New York, NY, USA: McGraw-Hill, Inc.

Sapienza, M., Cuzzolin, F., & Torr, P. H. (2014). Learning discriminative space–time action parts from weakly labelled videos. *International journal of computer vision*, *110*(1), 30–47.

Sato, A. & Yamada, K. (1995). Generalized learning vector quantization. In *Proceedings of the 8th international conference on neural information processing systems* (pp. 423–429). NIPS'95. Denver, Colorado: MIT Press.

Scheme, E. J., Hudgins, B. S., & Englehart, K. B. (2013). Confidence-based rejection for improved pattern recognition myoelectric control. *IEEE Transactions on Biomedical Engineering*, *60*(6), 1563–1570.

Schiaffino, S., Garcia, P., & Amandi, A. (2008). Eteacher: Providing personalized assistance to e-learning students. *Computers & Education*, *51*(4), 1744–1754.

Schlimmer, J. C. & Granger, R. H. (1986). Incremental learning from noisy data. *Machine Learning*, *1*(3), 317–354.

Schlimmer, J. & Fisher, D. (1986). A case study of incremental concept induction. (pp. 496–501).

Schneider, P., Biehl, M., & Hammer, B. (2009). Adaptive relevance matrices in learning vector quantization. *Neural Computing*, *21*(12), 3532–3561.

Schulz, A., Gisbrecht, A., & Hammer, B. (2015). Using discriminative dimensionality reduction to visualize classifiers. *Neural Processing Letters*, *42*(1), 27–54.

Shalev-Shwartz, S., Singer, Y., Srebro, N., & Cotter, A. (2011). Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical Programming*, *127*(1), 3–30.

Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, *5*(1), 3–55.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., van den Driessche, G., … Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*, 484–489.

Silverman, B. W. (2018). *Density estimation for statistics and data analysis*. Routledge.

Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*.

Spencer, R. (2013). The square kilometre array: The ultimate challenge for processing big data. In *Iet seminar on data analytics 2013: Deriving intelligence and value from big data*.

Stephens, Z. D., Lee, S. Y., Faghri, F., Campbell, R. H., Zhai, C., Efron, M. J., … Robinson, G. E. (2015). Big data: Astronomical or genomical? *PLOS Biology*, (7).

Street, W. N. & Kim, Y. (2001). A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh acm sigkdd international conference on knowledge discovery and data mining* (pp. 377–382). KDD '01. San Francisco, California: ACM.

Sugano, Y., Matsushita, Y., Sato, Y., & Koike, H. (2008). An incremental learning method for unconstrained gaze estimation. In *European conference on computer vision* (pp. 656–667). Springer.

Sun, J. & Li, H. (2011). Dynamic financial distress prediction using instance selection for the disposal of concept drift. *Expert Systems with Applications*, *38*(3), 2566–2576.

Tang, J., Deng, C., & Huang, G.-B. (2016). Extreme learning machine for multilayer perceptron. *IEEE transactions on neural networks and learning systems*, *27*(4), 809–821.

Tang, J., Deng, C., Huang, G.-B., & Zhao, B. (2015). Compressed-domain ship detection on spaceborne optical image using deep neural network and extreme learning machine. *IEEE Transactions on Geoscience and Remote Sensing*, *53*(3), 1174–1185.

Tessendorf, B., Gravenhorst, F., Arnrich, B., & Tröster, G. (2011). An imu-based sensor network to continuously monitor rowing technique on the water. In *2011 seventh international conference on intelligent sensors, sensor networks and information processing* (pp. 253–258).

Ting, S., Ip, W., & Tsang, A. H. (2011). Is naive bayes a good classifier for document classification? *International Journal of Software Engineering and Its Applications*, *5*(3), 37–46.

Tintarev, N. & Masthoff, J. (2007). A survey of explanations in recommender systems. In *2007 ieee 23rd international conference on data engineering workshop* (pp. 801–810). IEEE.

Tseng, M. M. & Piller, F. (2011). *The customer centric enterprise: Advances in mass customization and personalization*. Springer Science & Business Media.

Tsymbal, A., Pechenizkiy, M., Cunningham, P., & Puuronen, S. (2006). Handling local concept drift with dynamic integration of classifiers: Domain of antibiotic resistance in nosocomial infections. In *19th ieee international symposium on computer-based medical systems, 2006.* (pp. 679–684). IEEE.

Tulving, E. et al. (1972). Episodic and semantic memory. *Organization of memory*, *1*, 381–403.

U.S. Department of Transportation, F. H. A. (2007). The national intersection safety problem.

UTGOFF, P. E. (1988). Id5: An incremental id3. In J. Laird (Ed.), *Machine learning proceedings 1988* (pp. 107–120). San Francisco (CA): Morgan Kaufmann.

Van Laarhoven, P. J. & Aarts, E. H. (1987). Simulated annealing. In *Simulated annealing: Theory and applications* (pp. 7–15). Springer.

Verbert, K., Parra, D., Brusilovsky, P., & Duval, E. (2013). Visualizing recommendations to support exploration, transparency and controllability. In *Proceedings of the 2013 international conference on intelligent user interfaces* (pp. 351–362). ACM.

Wallraven, Caputo, & Graf. (2003). Recognition with local features: The kernel recipe. In *Proceedings ninth ieee international conference on computer vision* (257–264 vol.1).

Wang, A., Wan, G., Cheng, Z., & Li, S. (2009). An incremental extremely random forest classifier for online learning and tracking. In *2009 16th ieee international conference on image processing (icip)*.

Wang, H. & Schmid, C. (2013). Action recognition with improved trajectories. In *2013 ieee international conference on computer vision* (pp. 3551–3558).

Watkin, T. L., Rau, A., & Biehl, M. (1993). The statistical mechanics of learning a rule. *Reviews of Modern Physics*, *65*(2), 499.

Weiss, G. M., Timko, J. L., Gallagher, C. M., Yoneda, K., & Schreiber, A. J. (2016). Smartwatch-based activity recognition: A machine learning approach. In *2016 ieee-embs international conference on biomedical and health informatics (bhi)* (pp. 426–429).

Weiss, G. M. & Lockhart, J. W. (2012). The impact of personalization on smartphone-based activity recognition. In *Aaai workshop on activity context representation: Techniques and languages* (pp. 98–104).

Welch, B. L. (1947). The generalization of student's' problem when several different population variances are involved. *Biometrika*, *34*(1/2).

Werbos, P. (1974). Beyond regression : New tools for prediction and analysis in the behavioral sciences.

Wersing, H. & Körner, E. (2003). Learning optimized features for hierarchical models of invariant object recognition. *Neural Computation*, *15*(7).

Widmer, G. & Kubat, M. (1992). Learning flexible concepts from streams of examples: Flora2, 463–467.

Widmer, G. & Kubat, M. (1993). Effective learning in dynamic environments by explicit context tracking. In *European conference on machine learning* (pp. 227–243). Springer.

Widmer, G. & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine learning*, *23*(1), 69–101.

Wiest, J., Karg, M., Kunz, F., Reuter, S., Kreßel, U., & Dietmayer, K. (2015). A probabilistic maneuver prediction framework for self-learning vehicles with application to intersections. In *Intelligent vehicles symposium (iv), 2015 ieee* (pp. 349–355). IEEE.

Wilcox, R. R. (2012). *Introduction to robust estimation and hypothesis testing*. Academic Press.

Wolpert, D. H. (2002). The supervised learning no-free-lunch theorems. In *Soft computing and industry* (pp. 25–42). Springer.

Yang, H. & Fong, S. (2011). Optimized very fast decision tree with balanced classification accuracy and compact tree size. In *The 3rd international conference on data mining and intelligent information technology applications*.

Yang, R. & Newman, M. W. (2013). Learning from a learning thermostat: Lessons for intelligent systems for the home. (pp. 93–102). UbiComp '13. Zurich, Switzerland: ACM.

Ye, Y., Squartini, S., & Piazza, F. (2013). Online sequential extreme learning machine in nonstationary environments. *Neurocomputing*, *116*, 94–101.

Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things Journal*, *1*(1).

Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhang, H. (2004a). The optimality of naive bayes. *AAAI*, *1*(2), 3.

Zhang, T. (2004b). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the*

twenty-first international conference on machine learning (p. 116). ACM.

Zhao, J., Wang, Z., & Park, D. S. (2012). Online sequential extreme learning machine with forgetting mechanism. *Neurocomputing*, *87*, 79–89.

Žliobaite, I. (2010). Learning under concept drift: An overview. eprint: arXiv:1010.4784

Žliobaite, I. (2013). How good is the electricity benchmark for evaluating concept drift adaptation. *CoRR*, *abs/1301.3524*.